COMPUTATIONAL TOOLS FOR PRELIMINARY MATERIAL DESIGN OF METALS AND POLYMER-CERAMIC NANO COMPOSITES

Approved by:

Dr. Karl Jacob, Advisor  
School of Materials Science and Engineering  
Georgia Institute of Technology

Dr. David McDowell, Co-Advisor  
School of Mechanical Engineering  
Georgia Institute of Technology

Dr. Ting Zhu  
School of Mechanical Engineering  
Georgia Institute of Technology

Dr. Wallace Carr  
School of Materials Science and Engineering  
Georgia Institute of Technology

Dr. Donggang Yao  
School of Materials Science and Engineering  
Georgia Institute of Technology

Date Approved: January 3, 2014
This dissertation is dedicated to my grandmother who has helped me my entire time in grad school and is happy I am finally done with my homework.
ACKNOWLEDGEMENTS

I wish to acknowledge the following people and organizations who have helped in the writing of this dissertation: David McDowell, Karl Jacob, Ting Zhu, Helen and Steven Kraus, Sonia Oxman, CCMD, NSF, John Terracina, Fred Cook, Hamid Garmestani, Seung Soon Jang, Wallace Carr, Anselm Griffin, Donggang Yao, Nicolas Hoffmann, Alex Fenheim, Michael Kraus and Leroy Emkin.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>x</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xii</td>
</tr>
<tr>
<td>LIST OF SYMBOLS AND ABBREVIATIONS</td>
<td>xv</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>xix</td>
</tr>
<tr>
<td><strong>CHAPTER</strong></td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Preliminary Material Design</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Interatomic Potentials</td>
<td>1</td>
</tr>
<tr>
<td>1.3 Polymer-Metal Oxide Nano Composites</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Computational Tools</td>
<td>3</td>
</tr>
<tr>
<td>1.5 References</td>
<td>4</td>
</tr>
<tr>
<td>2 LITERATURE REVIEW</td>
<td>6</td>
</tr>
<tr>
<td>2.1 Metal Potential Fitting</td>
<td>6</td>
</tr>
<tr>
<td>2.1.1 Interatomic Potentials</td>
<td>6</td>
</tr>
<tr>
<td>2.1.2 Metal Fitting Software</td>
<td>8</td>
</tr>
<tr>
<td>2.1.3 Software Tools</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Polymer Initialization</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1 Freely Jointed Chain</td>
<td>10</td>
</tr>
<tr>
<td>2.2.2 Freely Rotating Chain</td>
<td>10</td>
</tr>
<tr>
<td>2.2.3 Self Avoiding Walk</td>
<td>11</td>
</tr>
<tr>
<td>2.2.4 Optimization Algorithms</td>
<td>11</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.2.5 Nano Composite</td>
<td>12</td>
</tr>
<tr>
<td>2.3 Molecular Dynamics Bonding</td>
<td>12</td>
</tr>
<tr>
<td>2.4 Nano Composite Theory</td>
<td>14</td>
</tr>
<tr>
<td>2.4.1 Bonding</td>
<td>15</td>
</tr>
<tr>
<td>2.4.2 Young’s Modulus</td>
<td>16</td>
</tr>
<tr>
<td>2.5 PMMA-Metal Oxide Nano Composites</td>
<td>16</td>
</tr>
<tr>
<td>2.5.1 Thermal Stability and Mechanical Properties</td>
<td>16</td>
</tr>
<tr>
<td>2.5.2 Nano Particle Reactivity</td>
<td>18</td>
</tr>
<tr>
<td>2.5.3 MD Simulations</td>
<td>19</td>
</tr>
<tr>
<td>2.5.4 Thermal Property Relationship</td>
<td>20</td>
</tr>
<tr>
<td>2.5.5 Material Variables Effects</td>
<td>23</td>
</tr>
<tr>
<td>2.6 References</td>
<td>24</td>
</tr>
<tr>
<td>3 METHODOLOGY</td>
<td>29</td>
</tr>
<tr>
<td>3.1 Objectives</td>
<td>29</td>
</tr>
<tr>
<td>3.1.1 Developing Metallic Potentials</td>
<td>29</td>
</tr>
<tr>
<td>3.1.2 Studying PMMA-Metal Oxide Nano Composites</td>
<td>29</td>
</tr>
<tr>
<td>3.1.3 Producing Computational Tools</td>
<td>29</td>
</tr>
<tr>
<td>3.2 Metal EAM Potential Fitting Algorithm</td>
<td>30</td>
</tr>
<tr>
<td>3.2.1 Framework Design</td>
<td>30</td>
</tr>
<tr>
<td>3.2.2 Test Case</td>
<td>32</td>
</tr>
<tr>
<td>3.2.3 Testing Potential</td>
<td>37</td>
</tr>
<tr>
<td>3.3 Bonding Algorithm</td>
<td>41</td>
</tr>
<tr>
<td>3.3.1 Composite</td>
<td>41</td>
</tr>
<tr>
<td>3.3.2 Particle Surface</td>
<td>42</td>
</tr>
<tr>
<td>3.3.3 Molecules</td>
<td>42</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.5 Glass Transition Temperature</td>
<td>91</td>
</tr>
<tr>
<td>6.6 References</td>
<td>92</td>
</tr>
<tr>
<td>7 PMMA-ALUMINA NANO COMPOSITE</td>
<td>94</td>
</tr>
<tr>
<td>7.1 PMMA Properties</td>
<td>94</td>
</tr>
<tr>
<td>7.2 Molecular Energy Difference Between PMMA and Composite</td>
<td>96</td>
</tr>
<tr>
<td>7.3 Effects of Filler on Composite Properties</td>
<td>97</td>
</tr>
<tr>
<td>7.4 Effects of Density on Young’s Modulus</td>
<td>105</td>
</tr>
<tr>
<td>7.5 Effects of Molecular Energy on Young’s Modulus</td>
<td>109</td>
</tr>
<tr>
<td>7.6 Generalized Model</td>
<td>110</td>
</tr>
<tr>
<td>7.7 References</td>
<td>116</td>
</tr>
<tr>
<td>8 CONCLUSIONS AND FUTURE WORK</td>
<td>118</td>
</tr>
<tr>
<td>APPENDIX A: CALCULATING DIFFERENT ANGULAR SYSTEMS FOR INVERSIONS</td>
<td>122</td>
</tr>
<tr>
<td>APPENDIX B: PRODUCING INITIAL PMMA STRUCTURE FOR MD</td>
<td>129</td>
</tr>
<tr>
<td>APPENDIX C: RNG ALGORITHM IN PERIODIC SPACE</td>
<td>142</td>
</tr>
<tr>
<td>APPENDIX D: BULK PMMA MD SIMULATIONS AND PROPERTIES</td>
<td>143</td>
</tr>
<tr>
<td>APPENDIX E: PMMA-ALUMINA POTENTIAL TABLE CODE</td>
<td>158</td>
</tr>
<tr>
<td>APPENDIX F: NANO COMPOSITE MD SIMULATIONS AND PROPERTIES</td>
<td>164</td>
</tr>
<tr>
<td>APPENDIX G: BONDING ALGORITHM</td>
<td>205</td>
</tr>
<tr>
<td>APPENDIX H: METAL POTENTIAL FITTING</td>
<td>261</td>
</tr>
<tr>
<td>APPENDIX I: METAL POTENTIAL TESTS</td>
<td>473</td>
</tr>
<tr>
<td>APPENDIX J: MANUAL FOR BONDING ALGORITHM</td>
<td>487</td>
</tr>
<tr>
<td>APPENDIX K: MANUAL FOR RNG ALGORITHM</td>
<td>498</td>
</tr>
<tr>
<td>APPENDIX L: MANUAL FOR METAL POTENTIAL FITTING</td>
<td>505</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 2.1: Methods to create polymer metal oxide nano composites. 15
Table 2.2: Bonding process between PMMA and alumina. 19
Table 3.1: Property calculations selected for Fitting and Testing modules. 33
Table 3.2: Property values used for fitting and testing nickel potentials and the calculated values of the nickel potential selected by the Main Controller module. 34
Table 3.3: Different values of 0K lattice constants for copper and aluminum. 39
Table 3.4: Parameters for Equation 3.6. 39
Table 3.5: Nickel lattice constants for two different relationships at room temperature and 0K. 40
Table 3.6: Bonding process in the MD simulation box. 46
Table 3.7: Parameters for dihedral Equation 4.10 and 4.11. 48
Table 3.8: Parameters for inversion Equation 4.12 and improper Equation 4.13. 49
Table 3.9: Potential parameters for alumina. 57
Table 3.10: Parameters for the Born-Mayer Huggins potential for alumina in LAMMPS. 57
Table 3.11: Binding potential parameters describing the interaction between aluminum and the oxygen in PMMA. 58
Table 3.12: Non binding potential parameters describing the interaction between aluminum and carbon in PMMA and the interaction between oxygen in alumina and the atoms in PMMA. 59
Table 3.13: Computational experimental design of PMMA-alumina nano composite. 60
Table 3.14: Wall Potential Parameters. 61
Table 4.1: Elastic constants at 0K for different potentials and their calculated Young’s modulus using Equation 4.8. 74
Table 6.1: Experimental values, calculated values, and statistical analysis of the radius of gyration, Young's modulus, and glass transition temperature.

Table 6.2: Radius of gyration calculated using Equation 6.1 for the end to end distance equal to the simulation box length and the length of the diagonal on the side of the box and the radius of gyration calculated through MD simulations.

Table 7.1: The calculated values and statistical analysis of the box length independent properties of pure PMMA.

Table 7.2: The Young's modulus (Mpa) of the un-bonded composite by composition and RNG seed values.

Table 7.3: How the sign of the parameters in Equation 7.1 affects the change of modulus with increasing density.

Table 7.4: Experimental material failure information from Ash et al².

Table 7.5: Interphase thickness of different compositions and bonding levels for the PMMA-alumina nano composite.
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Material scale performance relationship.</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Experimental-simulation cycle.</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>The three parts of the framework and how they connect.</td>
<td>30</td>
</tr>
<tr>
<td>3.2</td>
<td>The design of the potential fitting software</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Lattice constant temperature relationships between different aluminum potentials and a theoretical derived equation.</td>
<td>36</td>
</tr>
<tr>
<td>3.4</td>
<td>Region where bonding can occur around the nanoparticle surface atom.</td>
<td>43</td>
</tr>
<tr>
<td>3.5</td>
<td>Representation of PMMA bonded to the nano particle surface.</td>
<td>44</td>
</tr>
<tr>
<td>3.6</td>
<td>United Atom Method representation of PMMA where c’s are carbon and o’s are oxygen.</td>
<td>47</td>
</tr>
<tr>
<td>3.7</td>
<td>Diagram of mirror based RNG algorithm before new unit and addition vector have been reflected across the mirror plane.</td>
<td>52</td>
</tr>
<tr>
<td>4.1</td>
<td>Lattice constant temperature relationships of different nickel potentials and experimental/theoretical derived equations.</td>
<td>68</td>
</tr>
<tr>
<td>4.2</td>
<td>The Young’s modulus of different nickel potentials and those calculated from elastic constants for various temperatures.</td>
<td>70</td>
</tr>
<tr>
<td>4.3</td>
<td>EAM pair potentials for nickel.</td>
<td>72</td>
</tr>
<tr>
<td>4.4</td>
<td>EAM electron density potentials for nickel.</td>
<td>73</td>
</tr>
<tr>
<td>4.5</td>
<td>EAM embedding energy potentials for nickel.</td>
<td>74</td>
</tr>
<tr>
<td>4.6</td>
<td>Sum of residuals squared for the testing properties relative to the number of knots used for fitting.</td>
<td>76</td>
</tr>
<tr>
<td>4.7</td>
<td>Sum of residuals squared for the fitting properties relative to the number of knots used for fitting.</td>
<td>77</td>
</tr>
<tr>
<td>5.1</td>
<td>Splitting PMMA-alumina nano composite into separate components.</td>
<td>80</td>
</tr>
</tbody>
</table>
Figure 5.2: The different PMMA polymer chains interacting with the oxygen atoms on the nano particle surface.

Figure 5.3: Graphical explanation of bonding process between water, PMMA and alumina shown for the united atom structures.

Figure 6.1: Radius of gyration with respect to simulation box length.

Figure 6.2: Molecular energy with respect to simulation box length.

Figure 6.3: Young’s modulus with respect to simulation box length.

Figure 6.4: Density with respect to simulation box length.

Figure 6.5: Glass transition temperature with respect to simulation box length.

Figure 7.1: Young’s modulus of 10,000 g/mo PMMA with respect to box length.

Figure 7.2: Density of 10,000 g/mol PMMA with respect to box length.

Figure 7.3: Molecular energy relationship with simulation box length for pure PMMA and the unbonded PMMA-alumina nano composite.

Figure 7.4: Molecular energies of the unbonded and bonded PMMA-alumina nano composite as they change with composition.

Figure 7.5: Young’s modulus of unbonded, bonded, experimental and theoretical data with respect to composition.

Figure 7.6: The average density of peak one for the unbonded and bonded PMMA-alumina composite for different compositions.

Figure 7.7: The average density of peak two for the unbonded and bonded PMMA-alumina nano composite for different compositions.

Figure 7.8: The average density of peak three for the unbonded and bonded PMMA-alumina nano composite for different compositions.

Figure 7.9: The average density of the bulk material for the unbonded and bonded PMMA-alumina nano composite for different compositions.

Figure 7.10: The relationship between the peak 1 density and the Young’s modulus of the bonded and unbonded composite.

Figure 7.11: The relationship between the peak 2 density and the Young’s modulus of the bonded and unbonded composite.
Figure 7.12: The relationship between the peak 3 density and the Young’s modulus of the bonded and unbonded composite.

Figure 7.13: The relationship between the bulk density and the Young’s modulus of the bonded and unbonded composite.

Figure 7.14: The relationship between Young’s modulus and the molecular energy.

Figure 7.15: Relationship between S and % bonding.

Figure 7.16: Relationship between R and % bonding.

Figure 7.17: Relationship between S and R.
LIST OF SYMBOLS AND ABBREVIATIONS

E  energy
F  embedding energy
ρ  electron density
φ  pair potential
⟨R²⟩  average end to end distance of the chain squared
N  number of bonds in the chain
b  bond length
θ  bonding angle
Φ  dihedral angle
E'  storage modulus
E''  loss modulus
α  thermal diffusivity
C_p  constant pressure heat capacity
ρ  material density
k  thermal conductivity
α  linear thermal expansion
V  volume
T  temperature
a  lattice constant
C_v  constant volume heat capacity
M  Young’s modulus
σ  stress
\( \varepsilon \) strain

\( U \) potential energy

\( \chi \) improper angle

\( C_{11} \) 11 elastic constant

\( C_{12} \) 12 elastic constant

\( x \) position

\( r \) radius

\( d \) distance

\( n \) normal vector

\( q \) effective charge

\( p \) stress tensor

\( m \) mass

\( v \) velocity

\( r \) separation distance

\( f \) force

\( Y \) Young’s modulus

\( V_f \) volume fraction

\( \delta \) anchor density

\( E \) molecular energy

\( x \) distance from the center of the material

MD molecular dynamics

MC Monte Carlo

MM molecular mechanics

GULP General Utility Lattice Program

PMMA poly(methyl methacrylate)
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAMMPS</td>
<td>Large-scale Atomic/Molecular Massively Parallel Simulator</td>
</tr>
<tr>
<td>EAM</td>
<td>embedded atom method</td>
</tr>
<tr>
<td>BCC</td>
<td>body center cubic</td>
</tr>
<tr>
<td>HCP</td>
<td>hexagonal close packed</td>
</tr>
<tr>
<td>MEAM</td>
<td>modified embedded atom method</td>
</tr>
<tr>
<td>FCC</td>
<td>face centered cubic</td>
</tr>
<tr>
<td>RNG</td>
<td>random number generator</td>
</tr>
<tr>
<td>FE</td>
<td>finite element</td>
</tr>
<tr>
<td>UFF</td>
<td>universal force field</td>
</tr>
<tr>
<td>COMPASS</td>
<td>condensed-phase optimized molecular potentials for atomistic simulation studies</td>
</tr>
<tr>
<td>DSC</td>
<td>differential scanning calorimetry</td>
</tr>
<tr>
<td>DMA</td>
<td>dynamic mechanic analysis</td>
</tr>
<tr>
<td>TGA</td>
<td>thermal gravimetric analysis</td>
</tr>
<tr>
<td>LOI</td>
<td>limiting oxygen index</td>
</tr>
<tr>
<td>TTI</td>
<td>time-to-ignition</td>
</tr>
<tr>
<td>THR</td>
<td>total heat released</td>
</tr>
<tr>
<td>HRR</td>
<td>heat release rate</td>
</tr>
<tr>
<td>pHRR</td>
<td>peak heat release rate</td>
</tr>
<tr>
<td>TOF</td>
<td>time of flameout</td>
</tr>
<tr>
<td>t_pHRR</td>
<td>time to peak heat release</td>
</tr>
<tr>
<td>MLR</td>
<td>mass loss rate</td>
</tr>
<tr>
<td>TSR</td>
<td>total smoke release</td>
</tr>
<tr>
<td>TBP</td>
<td>total burning period</td>
</tr>
<tr>
<td>VMD</td>
<td>visual molecular dynamics</td>
</tr>
</tbody>
</table>
TEM  
	ransmission electron microscopy
SUMMARY

In this dissertation, algorithms for creating estimated potentials for metals and modeling of nano composites are developed. The efficacy of the algorithms for estimated potentials were examined. The algorithm was found to allow molecular dynamic and Monte Carlo modeling to be included in the potential building process. Additionally, the spline based equations caused issues with the elastic constants and Young’s modulus due to extra local minima. Two algorithms were developed for improved modeling of nano composites: one was a random number generation algorithm for initializing polymer, second was a bonding algorithm for controlling bonds between polymer and nano particle. Both algorithms were effective in their tasks.

Additionally, the algorithms for improved nano composite modeling were used for preliminary material design of PMMA metal oxide nano composite systems. The results from the molecular dynamic simulations show the bonding between polymer matrix and nanoparticle has a large effect on the Young’s modulus and if this bonding could be controlled, the tensile properties of PMMA-metal oxide nano composites could be tailored to the applications’ requirements. The simulations also showed bonding had caused changes in the density of the material which than effected the energy on the polymer chain and the Young’s modulus. A model was than developed showing the relationship between density and the chain energy, and density and the Young’s modulus. This model can be
used for a better understanding and further improvement of PMMA-metal oxide nano composites.
CHAPTER 1

INTRODUCTION

1.1 Preliminary Material Design

Preliminary material design for this dissertation is defined as using theory and experimental data to effectively screen possible materials to determine the probability of meeting application specific requirements. Figure 1.1 shows the relationship of material selection and processing paths that affect the final properties. The material chemistry and structure in Figure 1.1 can easily be controlled during simulations. These simulations are capable of calculating material-property relationships and delivering insight into fundamental mechanisms which can further be utilized for preliminary material design or later experimentation. In this dissertation, computational experiments using molecular dynamics were utilized to explore the utility of new computational tools for preliminary material design.

1.2 Interatomic Potentials

Interatomic potentials are necessary to calculate energies (and forces) in atomic simulations like Molecular Dynamics (MD), Molecular Mechanics (MM),
and Monte Carlo (MC). Potentials allow these simulations to calculate energies
and properties fast and efficiently. For many pure materials, reasonably accurate
potentials are already available. Though, sometimes specific potentials are
needed to address structures and properties of interest for a given material. In
this case, a suitable potential may not exist and will require fitting to the
candidate system of interest. For material composites and alloys, linear
combinations of pure potentials typically will not work; therefore a new potential
may need developing.

Rapidly estimated potentials can be useful for material design in initial
screening. This leads to less time creating new potentials and more time
exploring property relationships or examining properties of possible new
materials. For organic, ceramic, and metallic crystalline materials, the General
Utility Lattice Program (GULP) is capable of estimating potentials. Unfortunately,
the choices of fitting properties are limited which for metals may be an issue
depending on user demands. Additionally, GULP’s current fitting capabilities do
not include material properties that are calculated using MC or MD
simulations\textsuperscript{1,2,3}. Although including these simulations are not required for metallic
potentials, allowing their inclusion in estimated potentials may prove to be useful.
In order to improve metallic estimated potentials, a novel methodology needs
developing which allows user driven property selection and links to MC or MD
simulation software. This new methodology will be developed for the embedded
atom method. To test this new methodology, potentials for nickel were developed
and tested.

1.3 Polymer Metal Oxide Nano Composites

Polymer metal oxide nano composites are important structural materials
because the nano particles modify the strength and elasticity of the material while
typically adding new properties not inherent in the base polymer. Like traditional
polymer composites size, shape, and composition effect the final tensile
properties. Additionally, for nano composites, the interphase thickness and
density combined with the number of bonds between materials is believed to have an appreciable effect on the final tensile properties.

To understand the effects of bonding and the interface, material testing at the nanoscale is required. Although experiments are capable of testing at this scale, MD simulations can explore the effects of controlled bonding. Accurate control of bonding in simulations is extremely important for gleaning a theoretical model or cause and effect relationship. To produce this model/relationship, MD simulations were done examining the effects of the following design variables: mass percent of metal oxide, and percentage of bonds between polymer and metal oxide.

For these computational simulations, poly(methyl methacrylate) (PMMA) - aluminum oxide nano composite was examined. These simulations utilized potentials for alumina, PMMA, and their interactions which were compatible with the united atom method. The united atom method merges the hydrogen atoms into their attached atom on the polymer chain. The united atom method increases the accessible time-length scale of the simulations, which allows either larger simulation boxes or longer simulation times. For this experiment larger simulation boxes were required.

Additionally, software tools were required to complete the computational simulations. Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS)\textsuperscript{4,5}, which is distributed by Sandia National Laboratories, was used for the MD simulations. A new methodology and software tool was developed to produce initial polymer configurations in the composite. An algorithm was developed to control the bonding between the polymer and the nano particle in the composite. These tools allowed examining the effects of the variables to further our understanding of polymer metal oxide nano composites.

\textbf{1.4 Computational Tools}

Computational tools in this dissertation are defined as software tools, algorithms, equations, and potentials used for running simulations and data analysis. Figure 1.2 shows the relation of these tools in the experimental-
This cycle shows that improvements in computational tools leads to improvements in simulations and calculations. Additionally, the improved simulations and calculations leads to better models and theories, and new experiments. The new experiments are designed to either confirm the results or prove the models and theories from simulations and calculations. The improved models and theories lead to improved software tools and the cycle continues.

Figure 1.2: Experimental-simulation cycle.

Therefore computational tools are developed for this dissertation to advance the knowledge base from simulations and derive new models and theories where possible. The computational tools developed are an algorithm for bonding polymer nano composites, a software for initializing the polymer for MD simulations of composites, and a tool for estimating metallic potentials.

1.5 References


CHAPTER 2
LITERATURE REVIEW

2.1 Metal Potential Fitting

2.1.1 Interatomic Potentials

The embedded atom method (EAM) was developed approximately 30 years ago for modeling metallic materials\textsuperscript{1,2}. Since then, potentials have been developed for vanadium, copper, nickel, aluminum, silver, gold, and platinum\textsuperscript{1,3-15}. These potentials have been fit with a multitude of different equation sets and a few have even been fit with splines. The EAM does have one drawback; the electron density has no directional component, leading to difficulties producing accurate potentials for body center cubic (BCC)\textsuperscript{15} and hexagonal close packed (HCP) materials. The modified embedded atom method (MEAM) added directional components to the EAM\textsuperscript{16,17} which has allowed the MEAM to be accurately applied to both BCC and HCP materials\textsuperscript{18,19}. The MEAM also has a standard equation set and method for fitting the potentials\textsuperscript{20}.

2.1.1.1 EAM

Daw and Baskes created the EAM to fix a discrepancy caused by using pair potentials to describe metallic materials. The key discrepancy is the elastic constant \( C_{12}=C_{44} \). To create this new method they used density functional theory. The final derived equation is

\[
E_{\text{tot}} = \sum_i F_i(\rho_i(R_i)) + \frac{1}{2} \sum_{i,j} \varphi(R_{i,j}),
\]

where \( F \) is the embedding energy, \( \rho \) is the electron density, and \( \varphi \) is the pair potential\textsuperscript{2}. The embedding energy is based on the quasiatom or effective medium theory which states the energy of an embedded impurity is a function of the
electron density of the pure material at the embedding location. Combining the embedding energy with the pair potential, a realistic description of metallic materials can be defined and correct elastic constants can be calculated. Equation 2.1 was later altered by Daw and Baskes to apply to alloys. This altered version of Equation 2.1 is

\[ E_{\text{tot}} = \sum_i F_i(\rho_{h,i}) + \frac{1}{2} \sum_{i,j \neq i} \varphi_{i,j}(R_{i,j}), \]  

(2.2)

where \( \rho_{h,i} \) is the host electron density and \( \varphi_{i,j} \) is the pair interaction between atom \( i \) and \( j \). Atom \( i \) and \( j \) can be the same species (pure element pair potential) or different species (mixed pair potential)\(^1\). During the alteration of Equation 2.1, an equation,

\[ \rho_{h,i} = \sum_j \rho_j(R_{i,j}), \]  

(2.3)

describing the host electron density was formally introduced\(^1\).

2.1.1.2 Purpose for Developing Method for EAM Potential Estimation

If the MEAM already has a standard equation set and methodology for estimating metal potentials\(^20\), why produce a methodology for the EAM? The primary reason is simulation speed and the secondary reason is a less complicated optimization procedure. The MEAM\(^20\) is extremely complicated which means slower simulations. For materials where directional bonding is required to accurately model the material properties, the increased simulation time may be worth the increased accuracy. But for materials which can be modeled as uniformly bonded, such as the face centered cubic (FCC) crystal system, the increased simulation time may be prohibitive. Additionally the MEAM requires a double fitting procedure to develop the potential\(^20\), but the EAM potential is developed with a single fitting procedure. Therefore developing EAM potentials should be easier and quicker and the better choice for uniformly
bonded materials. Currently there are only two methods for estimating EAM potentials and both have flaws which make them non universal for metals.

2.1.2 Metal Fitting Software

Both of these methods for estimating EAM potentials are available in software packages. The first is the General Utility Lattice Program (GULP)\textsuperscript{21-23} and the second is POCO\textsuperscript{24}.

GULP\textsuperscript{21-23} was originally designed for fitting potentials to crystalline ceramic and organic systems. Later the software was updated to allow the development of metal and metallic compound potentials in the form of the EAM. These potentials can be fitted to any bulk solid property. For metals, these properties include elastic constants, phonon frequencies, Poisson’s ratio, Young’s modulus, and bulk and shear moduli. Any non bulk properties, such as surface energies, vacancy properties, stacking faults, interstitials, etc. cannot be included in the fit\textsuperscript{21-23}. These unavailable properties can be problematic for the user depending on the properties required in developing the potential.

POCO uses the force matching method developed by Adam et al\textsuperscript{25}. In this method, two sets of properties are included in the potential. One set of properties is calculated at 0K and the other is calculated with MD. The MD properties are included by matching the forces calculated from MD with a database of forces. The database is created by using quantum mechanics calculations to calculate the forces on atoms in different MD simulated properties. These forces are then used to update the atoms’ velocities and positions. Quantum mechanics calculates new forces which are then used to update the atoms again. The calculations and updates continue until enough forces have been gathered for each property\textsuperscript{25}.

Unfortunately, the force matching method is believed to have issues for cases of partially filled d orbitals. All of the recorded information show the method has been used for metals with no d orbitals or filled d orbitals\textsuperscript{26}. The issue is believed to stem from the quantum mechanical derived forces being more complicated and slightly less reliable when the d orbitals are partially filled\textsuperscript{25}. 
In an attempt to address the flaws of both methods, a framework design will be developed and tested. This design will allow current software tools which are useful for developing EAM potentials to be added to and utilized by the methodology developed in this dissertation.

2.1.3 Software Tools

Software tools useful for fitting EAM potentials are of two types. The first type employs optimization. These are designed to find the equations’ parameters which minimize an objective function. The objective function is the weighted sum of squared errors between the experimental and calculated properties. The second type calculates material properties using Monte Carlo simulations, MD simulations, or energy optimizations. Many of these software tools are located on the internet and are free to use.

One website written by John Burkardt contains many non linear optimization tools written in C, C++, and fortran. Python also has many software packages containing non linear optimization tools. SciPy, NLPy, and pyOpt are a few of these packages. Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) and GULP are capable of calculating many material properties. These software can be added into the framework and used on an as-needed basis.

2.2 Polymer Initialization

Polymer initialization is a necessary step for MD or MC simulations. In this step, a sample polymer configuration is built. The configurations are produced using theoretical models, self-avoiding walk methods, or optimization algorithms. The theoretical models include freely jointed chain and freely rotating chain. Initialization methods are attempts to simulate natural polymer chain configurations.
2.2.1 Freely Jointed Chain

In this method of initialization, the bond length is fixed but the chain is allowed to freely rotate\(^3\). Therefore, the chain follows the random walk model and can be described statistically as a Gaussian distribution\(^3\). The chain on average is centered around the origin due to the chains completely random path\(^3\). On the other hand the chains 2nd moment is
\[
\langle R^2 \rangle = N b^2, \tag{2.4}
\]
where \( \langle R^2 \rangle \) is the average end to end distance of the chain squared, \( N \) is the number of bonds in the chain, and \( b \) is the bond length\(^3\).

The Kuhn length approach allows the chain stiffness to be included in freely jointed chain models\(^3\) by modeling a real chain as a freely jointed chain consisting of Kuhn segments. The Kuhn segments can be viewed as linear sections of a real chain that has a length equal to the Kuhn length. These segments can than orient in any random direction leading to an average end to end distance of the chain squared which is equal to Equation 2.4. But in this case, \( b \) in Equation 2.4 is the Kuhn length and not the bond length\(^6\).

2.2.2 Freely Rotating Chain

The freely rotating chain restricts the bond angles to a specific value\(^3\). The average end to end distance of the chain squared than becomes
\[
\langle R^2 \rangle = N b^2 \frac{1 - \cos \theta}{1 + \cos \theta}, \tag{2.5}
\]
where \( \theta \) is the bonding angle. In the case where the bonding angle is 109.5°, the freely rotating chain’s \( \langle R^2 \rangle \) is 2 times the freely jointed’s \( \langle R^2 \rangle \)\(^3\).

The rotational isomeric state model is more realistic than the freely rotating chain or freely jointed chain because the model restricts both bond angles and dihedral angles\(^3\). Although dihedral angles in a real chain can be any value, they tend to be near values that lead to the lowest energy\(^3\). These restrictions lead to an average end to end distance of the chain squared defined as
\[
\langle R^2 \rangle = Nb^2 \frac{1 - \cos \theta}{1 + \cos \theta} \frac{1 - \langle \cos \phi \rangle}{1 + \langle \cos \phi \rangle},
\]
(2.6)

where \( \langle \cos \phi \rangle \) is the average cosine value of the minimum energy dihedral angles\(^3^4\).

The rotational isomeric state model, freely rotating chain model and freely jointed chain model approximate chain behavior of bulk amorphous polymers. In this state, the non-local interactions between groups of atoms do not affect chain conformation\(^3^4\).

### 2.2.3 Self-Avoiding Walk

When non-local interactions begin to affect the chain conformation, the self-avoiding walk is the most accurate model to initialize the polymer chains\(^3^4\). In this case, chains are placed on a lattice with no overlap allowed. Additionally, each unit is required to be one lattice position over from bonded units\(^3^7\). If while adding units to the chain an overlap occurs, the added unit is discarded and a new unit added to the chain is attempted\(^3^7\). This model forces the chains’ bonds not to cross each other in space. This leads to an average end to end distance of the chain squared relationship of
\[
\langle R^2 \rangle \propto N^\nu,
\]
(2.7)

where \( \nu \) equals 0.6 for the self-avoiding walk in three-dimensional space\(^3^4\).

The conditions that lead to non-local interactions affecting chain conformation often occur when the polymer chains are in a dilute solution. For this case, self-avoiding walk is the most accurate model to describe the chain conformations\(^3^4\).

### 2.2.4 Optimization Algorithms

Another option for polymer initialization is to use optimization algorithms. There are two categories of objective functions these algorithms attempt to minimize, spatial separation and energy. Energy minimization can be done one
time with the bulk material, such as the minimize command in LAMMPS, or each
time a new chain segment or atom is added to the simulation box. Additionally
the spatial separation between chains can be optimized assuring each chain is
far enough apart to start MD or other derivative based modeling approaches.
This approach is used in Packmol\textsuperscript{38}.

2.2.5 Nano Composite

Currently, self-avoiding walk and optimization algorithms can be used to
initialize the polymer in the nano composite because both methods can keep the
polymer from overlapping the nano particle. With the random walk methods, the
random nature of the polymer chain means there is no guarantee an overlap with
the nano particle will be avoided. To solve the issue with random walk methods, a
new random number generator (RNG) algorithm will be developed.

2.3 Molecular Dynamics Bonding

Currently their are four methods that allow bonding to occur during MD
simulations. ReaxxFF allows the researcher to analyze the reaction pathway,
surface rearrangements due to reaction and product distribution\textsuperscript{39} while running a
MD simulation. To accomplish this task, a quantum mechanical training set is
calculated, and the ReaxxFF parameters are fitted to the training set\textsuperscript{39}. The
training set includes charge distributions, bond dissociation energies, angle
distortion energies, rotational barriers, reaction energies and mechanisms, and
condensed-phase (equation of state) data\textsuperscript{39}. Large-scale atomic/molecular
massively parallel simulator (LAMMPS)\textsuperscript{32,33} comes with two bonding algorithms.
These algorithms are “fix_bond_create”\textsuperscript{40}, and “fix_bond_swap”\textsuperscript{41-43}. For
“fix_bond_create”, bonds are created between atoms of the user specified types
during the MD simulation when certain specifications are met\textsuperscript{40}. These
specifications are the atoms must be less than the bonding distance, must not be
bonded already, and must not have reached the maximum number of bonds\textsuperscript{40}.
Furthermore, only one bond can form for each atom per bonding time step, and
the bonds are permanent\textsuperscript{40}. For “fix_bond_swap”, bonds are swapped between
polymer chains. The swapping operation uses Monte Carlo rules and the Boltzmann acceptance criteria. Swaps only occur if the atom’s molecule types are the same and the atom’s separation distance is less than bonding distance. Finally quantum mechanics can be coupled to MD to allow chemical reactions to occur during the MD simulation. None of these methods give the user control over where these bonds will form or how many will form between the nano particle and the polymer matrix.

Researchers have explored bead spring models to bond polymer to the nano particle. This method of bonding allows researchers easy control over both the location and the number of bonds. Unfortunately the bead spring [coarse grain] models are unable to accurately model ceramic and metal nano particles because they can not realistically be represented as bonded beads. Instead, the nano particles need to be defined in terms of the explicit or united atom model. These models would allow the ceramic nano particles to be represented as charged atoms and enable the use of Coulomb potentials. These models would also allow the metal nano particles to be represented as atoms floating in a sea of electrons and enable the use of the embedded atom method or modified embedded atom method.

Changing the nano composite to the explicit or united atom method is simple but methods for controlled bonding of the polymer to the nano particle become more complicated. The complications arise from numerous sources, one of which is the extra information (bonds, angles, dihedrals, etc, etc.) that switching from the bead spring model adds. There are two approaches, minimum energy approach and separation distance approach.

The minimum energy approach searches for positions on the nano particle where monomer units close to the nano particle can bond to give a minimum energy. Although quantum mechanical calculations would give the most accurate answer, these calculations are infeasible due to the large system size of the nano composite system. The only other option is a traditional potential and optimization calculation. For this option, all possible combinations of monomers close to the surface and nano particle atoms are bonded together. These
combinations can than have an energy optimization done. The combination with the lowest energy can than be selected as the correct bonding location. While this approach is very computationally intensive and may work for small polymer chains, for polymers chains above the entanglement length this approach is actually incorrect.

For chains above the entanglement length, the chains’ movements follow the reptation model. This model says the chains are confined in a tube based on the entanglements with other chains\textsuperscript{34}. Additionally the chains only move through their individual “tubes”. This means polymers above their entanglement length will be at kinetic equilibrium rather than thermodynamic equilibrium because kinetic barriers surrounding the polymers tubes possibly prevent the polymer chains from proceeding directly towards thermodynamic equilibrium.

The separation distance approach searches for positions on the nano particle and monomer units that have a distance less than a specified separation distance which is small. The specified separation distance is small enough that monomer units and nano particle atoms that meet this criterion are assumed to have an even probability of bonding because the monomers are close enough to form bonds without requiring the chains to reptate into position. This approach is extremely fast computationally and is the best approach for polymer chains longer than the entanglement length. Since this approach is much faster than the minimum energy approach, even for chains shorter than the entanglement length, this approach may be preferred.

2.4 Nano Composite Theory

In the literature there are three methods of producing nano composites\textsuperscript{47, 48, 49}. One is to mix the nano particles directly without any modifications. This method is good for nano particles that have a strong attractive interaction. Additionally the nano particles can have the surface functionalized to improve interaction with the polymer. These surface coatings can even allow the polymer to react with the coating, thus bonding polymer to nano particle. These coated nano particles are than mixed in with the polymer. The final method is to take the
surface coated nano particles and graft polymer brushes onto the nano particles. Than mix the brush coated nano particles with the polymer matrix.

2.4.1 Bonding

In some cases, strong polymer-nano particle interactions exist when the nano particles are mixed with the polymer. The two materials form bonds without any other steps required. For these materials, the bonding is believed to modify the interface thickness and improve dispersion of the nano particles in the polymer matrix\textsuperscript{47}. PMMA-metal oxide nano composites fall into this category and can be formed by simple mixing at a high enough temperature to melt the material. Two methods found in the literature to produce these composites are shown below in Table 2.1.

Table 2.1: Methods to create polymer metal oxide nano composites.

<table>
<thead>
<tr>
<th>Mixing Method</th>
<th>Mixing Rate</th>
<th>Temp.</th>
<th>Time</th>
<th>Final Forming</th>
<th>Extra Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Melt Compounding</td>
<td>200 rpm</td>
<td>230 °C</td>
<td>4 minutes</td>
<td>Extrusion</td>
<td>NA</td>
</tr>
<tr>
<td>Batch Mixing</td>
<td>50 rpm</td>
<td>225 °C</td>
<td>7 minutes</td>
<td>Compression Moulding</td>
<td>250 °C, 100 bar, 5 minutes</td>
</tr>
</tbody>
</table>

An alternative method of improving the dispersion of the nano particles in the polymer is to modify the surface of the nano particle\textsuperscript{49}. These surface modifications, which can include grafted polymer brushes, are designed to increase the polymer-nano particle interaction strength and reduce the nano particle-nano particle interaction strength\textsuperscript{49}. In the case of grafting poly(methyl methacrylate) (PMMA) brushes onto magnetite, the dispersion of nano particles in the PMMA matrix was controlled by the brush length\textsuperscript{50}. At longer brush lengths, the nano particles disperse uniformly while at shorter lengths they aggregate\textsuperscript{50}. 
Mean-field theory also shows longer brush lengths which are the same homopolymer as the matrix improve dispersion. When the brush and the matrix are the same homopolymer, Self-consistent field theory has found that increasing grafting density (number of bonded chains per surface area) leads to a minimum in the free energy of interaction between the nano particle and the bulk polymer. Though not noted in their paper, this minimum should lead to nano particles forming clusters in the composite in approaching thermodynamic equilibrium.

2.4.2 Young’s Modulus

Computational methods have shown the Young’s modulus of polymer nano composites is related to the interface thickness/density, polymer-nano particle interaction strength and the dispersion/clustering of nano particles. A finite element (FE) computational study using the Mori-Tanaka approach to model the effective interface found the Young’s modulus decreased with a larger interface, increased with increasing particle size and increased with smaller nano particle clusters. Molecular dynamics (MD) tensile simulations show the Young’s Modulus increases with decreasing nano particle size as long as the polymer-nano particle interaction is stronger than the polymer-polymer interaction, which is the opposite of the above FE result. MD also shows the density of the interface layer increases with a stronger polymer-nano particle interaction and decreases with increasing size nano particles.

2.5 PMMA-Metal Oxide Nano Composites

2.5.1 Thermal Stability and Mechanical Properties

Over the last decade, the addition of inorganic fillers has been shown to lead to enhanced mechanical properties and/or reduced flammability/increased thermal stability. The reduced flammability or increased thermal stability is the most important property of these materials. Fires in Europe kill 12 people a day and has been estimated to cost the European economy a 1% gross domestic
product loss. To decrease these losses in both people and dollar value, structural materials used in transportation, housing and buildings are now being engineered to have increased flame retardancy and decreased flammability. Also regulations have been passed requiring materials used to have thermal stability and fire retardant properties. Smoke emissions also are required to be low, non toxic and non opaque\textsuperscript{47}. To meet these requirements and reduce damages to people and property, PMMA, an important structural material which is flammable\textsuperscript{47}, has been engineered over the last decade to decrease flammability and increase thermal stability.

Below is a brief examination of different designs to reduce the fire hazard of PMMA. These examinations will also include a brief explanation on the designs effectiveness on increasing mechanical properties and/or decreasing flammability/increasing thermal stability. Friederich et al examined alumina and titanium dioxide filler at various loading content\textsuperscript{47}. Alumina was shown to have slightly higher storage modulus than the titanium dioxide at 15% loading by mass. The storage modulus decreased until 10% loading by mass. Glass Transition temperature was found to correspond with tan delta and increased with loading except for alumina. The time to ignition was increased by approximately 25% at 15% loading by mass for these two materials. Also the peak heat released was reduced by approximately 50% at 15% loading by mass. The total heat released was effectively reduced (approximately 30%) by the alumina at 15% loading by mass but only slightly reduced (approximately 10%) for the titanium dioxide at the same loading\textsuperscript{47}.

Laachachi examined titanium dioxide and iron oxide fillers\textsuperscript{55}. Both fillers improved the thermal stability by about 70°C and altered the rate of the three stages of degradation. The first two stages’ degradation were slowed by the addition of the filler, while the last stage’s degradation was increased. The alterations to degradation for this data explain the increase in thermal stability. Another important result is limiting oxygen index increases (decreased flammability) with increasing glass transition temperature\textsuperscript{55}.
Another paper examined titanium dioxide and iron oxide fillers. The results showed iron oxide and titanium dioxide both improved the thermal stability by 50ºC and lowered the peak heat release but time to ignition decreased for iron oxide, yet increased for titanium dioxide. The peak heat release rate decreased and total burn time increased with increasing percent of both fillers. Unfortunately, for the iron oxide filler, the total smoke released was always greater than the base polymer but titanium dioxide filler showed at high filler loading by weight (15%-20%) a decrease in the total smoke released. The effect on size of the filler was studied using nano particles and micro particles. The increasing size of the filler increased the time to ignition yet decreased the peak heat release rate and total burn time for both fillers studied. Two papers were written examining how the shape of Fe$_2$O$_3$ nano particles added to PMMA effects PMMA properties. Two different nano shapes were examined nanorods and particles (square shaped). The two papers clearly show the nanorods begin decomposing in air at a higher temperature and decrease the glass transition temperature. The particles on the other hand did not have any effect on the glass transition temperature.

Iron oxide/PMMA core/shell nano particles are not used as a structural material but mainly for their magnetic properties. Interestingly, this material decreases the thermal degradation temperature in air by 60ºC. The explanation for the decreased degradation temperature was the iron oxide in the core was catalyzing random scission of polymer chains in the PMMA core. Ironically, when the iron oxide nano particles are mixed rather than in a core/shell configuration, the iron oxide nano particles do not catalyze any reactions in the polymer and the degradation temperature increases from that of the bulk PMMA. The above results clearly show a large variation even when examining similar material variables.

2.5.2 Nano Particle Reactivity

In the previous paragraph, many metal oxide fillers were examined through traditional lab experiments. But, for molecular dynamics, the most
common filler analyzed was alumina, because the bonding process of PMMA onto alumina is well understood. The bonding process involves many reactions.

Table 2.2: Bonding process between PMMA and alumina$^{47}$.

<table>
<thead>
<tr>
<th>Step</th>
<th>Reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\text{Al}_2\text{O}_3 + \text{H}_2\text{O} \rightleftharpoons 2\text{AlO(OH)}$</td>
</tr>
<tr>
<td>2</td>
<td>$-\text{CH}_2\text{C(CH}_3\text{)(COOCH}_3\text{)} - + \text{H}_2\text{O} \rightarrow -\text{CH}_2\text{C(CH}_3\text{)(COOH)} - + \text{CH}_3\text{OH}$</td>
</tr>
<tr>
<td>3</td>
<td>$-\text{CH}_2\text{C(CH}_3\text{)(COOH)} - + \text{AlO(OH)} \rightleftharpoons \text{AlO}^+ (\text{CH}_2\text{C(CH}_3\text{)(COO)})^- + \text{H}_2\text{O}$</td>
</tr>
</tbody>
</table>

In the first step alumina reacts with water to produce AlO(OH) in a reversible reaction. Then the alumina/AlO(OH) mixture catalyzes the hydrolysis of PMMA ester group to produce methyl alcohol and a poly (methyl acrylate) group. The poly(methyl acrylate) group than reacts with AlO(OH) to form a bond. This reaction is reversible$^{47}$. If these reactions (Table 2.2) were done in solution, the number of bonds formed to the alumina could possibly be estimated. Unfortunately, no such simple estimate exists since these are solids. Also polymer chain mobility and effective volume of the polymer limit the possible number of interactions with the alumina nano particle.

2.5.3 MD Simulations

The literature shows molecular dynamics was used to better understand interactions with alumina nano particles. Cheng et al examined polymers adsorbed on alumina surfaces using molecular dynamic (MD) simulations. Cerius 2.0 was used to do the MD simulations$^{59}$. Polymers in these simulations were between 1400 and 1500 g/mol. The polymer chain was located about 5 to 10 angstroms from the alumina at the beginning of the MD simulation. The simulation used Universal 1.01 force field (UFF) and was ran for 5 to 30 ps at 300 K$^{59}$. The time step was set at 1 fs allowing a large number of conformations to be examined. The ensemble used in these simulations was NVE. In these simulations only the polymer chain was allowed to move. The simulations were done until the energy had stabilized. According to this paper the UFF is very well
suited for simulations between polymers and inorganic materials. The Force Field can be written as a sum of many interactions:

\[ E = E_R + E_\theta + E_\phi + E_w + E_{vdW} + E_{el}, \]  

where \( E_R \) is the bond stretching energy, \( E_\theta \) is the bond angle energy, \( E_\phi \) is the dihedral angle energy, \( E_w \) is the inversion energy, \( E_{vdW} \) is the van der Waals energy, and \( E_{el} \) is the electrostatic energy\(^{59}\).

In a paper by Chung and Leeuw, LiI and alpha/gamma alumina ion conduction was examined by molecular dynamics. The ion conductivity part of this paper is not very important to this background but the potential they used is important. The potential used is called the hybrid model potential function,

\[ U = \sum_{(ij)} U_2(r_{ij}) + \sum_{(ijk)} U_3 \left( \vec{r}_{ij}, \vec{r}_{ik} \right), \]  

which consists of a sum of two body and three body terms. The \( U_2 \) is the two body term, and \( U_3 \) is the three body term. The two body term consisted of many interactions: coulombic, repulsive, Born-Mayer, and many more. The triple body term only described the bonding system Al-O-Al and O-Al-O\(^{60}\).

Prathab et al. studied the adsorption behavior of MMA oligomers on metal oxides including alumina. These studies were designed to find how the MMA oligomers would interact with the metal oxides. To test these interactions, MMA oligomer and Alumina slabs were examined with molecular dynamics (MD) simulations using 300,000 steps at 1 fs time steps with a temperature of 298k. The potentials used for these MD simulations were condensed-phase optimized molecular potentials for atomistic simulation studies (COMPASS)\(^{61}\).

### 2.5.4 Thermal Property Relationship

With regard to addition of inorganic fillers to polymers, two sets of properties are examined. These properties either examine the effects of the filler on the polymer’s mechanical properties, thermal stability, or flammability. The glass transition temperature of the polymer-metal oxide nano composites was
found using differential scanning calorimetry (DSC)\textsuperscript{56}. Glass transition temperature can be affected by the interaction of the polymer with the nano particle’s surface\textsuperscript{57} through bonding and steric hinderance\textsuperscript{55}. This interaction can alter the polymer chain mobility and thus affect the glass transition temperature. Glass transition temperature can also be altered by other factors not associated with any polymer-nano particle interaction. These other factors are polydispersity, tacticity, molar mass, and residual monomer\textsuperscript{57}. The glass transition temperature can also be calculated using dynamic mechanical analysis (DMA) by finding the temperature at which tan delta reaches a maximum value. When adding filler, an increase of glass transition temperature should correspond to an increase in the elastic modulus (storage modulus) but has been shown to not always be the case\textsuperscript{47}. Thermal gravimetric analysis (TGA) was used to examine difference in decomposition mechanism between PMMA and PMMA-Fe$_2$O$_3$ in air and nitrogen. TGA show the effect the different filler have on thermal stability compared to the base material\textsuperscript{56,57}. Limiting oxygen index (LOI) for PMMA is equal to 18\textsuperscript{47}. LOI is the minimum oxygen needed to completely burn a material in at least 30 seconds\textsuperscript{55}. DMA is used to analyze mechanical properties. Specific mechanical properties measured are the storage modulus (E'), loss modulus (E'\textsuperscript{\prime\prime}), and tan delta (\tan [E'\textsuperscript{\prime\prime}/E'])\textsuperscript{47}. Properties measured in a cone calorimeter by Friederich et al that measure flammability are time-to-ignition (TTI), total heat released (THR), heat release rate (HRR), peak heat release rate (pHRR)\textsuperscript{47}. HRR is the amount of heat given off per unit area when a material is exposed to a constant heat flux. This measurement has been shown to predict fire hazard by estimating the spread rate and the maximum attainable temperature of a fire. The pHRR is the peak of the heat released and can be correlated to the maximum fire generating danger of the material. This value is useful in evaluating the potential fire safety of a material\textsuperscript{47}. Other useful fire properties are time of flameout (TOF), total heat release (THR), the time to peak heat release rate (t_pHRR), mass loss rate (MLR), total smoke released (TSR), and total burning period (TBP)\textsuperscript{48}. The properties discussed above describe the mechanical, thermal stability and
flammability properties of a material and are useful for analyzing the effectiveness of metal oxide fillers in PMMA.

Unfortunately predicting flammability properties or even thermal stability of materials through molecular dynamics is normally impossible but exploration of a relationship with thermal diffusivity holds the key to being able to estimate these properties using molecular dynamics. Thermal diffusivity can be calculated by

$$\alpha = \frac{C_p \times \rho}{k(T)}.$$  \hspace{1cm} (2.10)

Here, $k$ is the thermal conductivity, $C_p$ is the heat capacity, and $\rho$ is the material density. All these variables can be calculated using molecular dynamics and than thermal diffusivity can be calculated. Thermal diffusivity was found to be highly related to pHRR and THR and questionably related to TTI. The TTI increased in general as the thermal diffusivity increased but a lot of outliers were in the data set yielding a questionable relationship. The linear correlation coefficients for the relations of pHRR and THR with thermal diffusivity were between 0.8 and 0.92, corresponding to highly correlated properties. These relations showed pHRR and THR decrease with increasing thermal diffusivity. The explanation for these correlations involve the thermal gradient of the material which is affected by the thermal diffusivity. With a low thermal diffusivity material, the thermal gradient is high and the temperature on the outside is much higher than the temperature of the core. The higher surface temperature leads to faster decomposition on the surface causing the TTI to be lower\textsuperscript{47}. Also a low thermal diffusivity leads to less heat being transferred into the core of the burning material and more heat being released on the surface (hotspots) and away from the material (higher pHRR, THR). With a high thermal diffusivity material, the thermal gradient is low and the temperature on the outside is more equivalent to the temperature of the core. This temperature condition leads to a slower decomposition on the surface and a higher TTI\textsuperscript{47}. Also a higher thermal diffusivity increases the heat transfer into the core of a burning material, so less heat is released on the surface and away from
the material (lower pHRR, THR). The relationship and equation above explain how molecular dynamics calculations can be used to estimate TTI and pHRR.

\textbf{2.5.5 Material Variable Effects}

To better understand what affects polymer-metal oxide nano composite properties, researchers have studied the effects of certain material variables on mechanical, thermal stability and flammability properties. The effects of shape are discussed in two papers by Dzunuzovic et al\textsuperscript{56,57}. One paper discusses cubic particles\textsuperscript{56} and the other paper discusses the effects on nanorods\textsuperscript{57}. These two papers clearly showed that shape affects the thermal stability properties\textsuperscript{56,57}.

The size of the metal oxide particle and % filler was found to have a large effect on the flammability properties\textsuperscript{55}. The % filler was also found to affect the mechanical properties and the glass transition temperature\textsuperscript{47}.

Another key material variable which affects mechanical properties is weak/strong bonding. Weak/strong bonding defines how the polymer bonds to the surface of the nano particle. Weak bonding causes large polymer loops to form which extend into the polymer matrix. Strong bonding is the opposite of the weak bonding and has very small polymer loops form. These bonding types create different size interphases which can be seen using transmission electron microscopy (TEM). A large interphase corresponds with weak bonding and a small interphase corresponds to strong bonding. Improved bonding reduces the size of this interphase and theoretically (Composite Theory) should lead to improved mechanical properties.

Unfortunately, research has not shown composite theory to work consistently for polymer-metal oxide nano composites. Nonuniform bonding between polymer and nano composite and uneven nano composite distribution could cause the lack of relationship between bonding strength and mechanical properties that were expected\textsuperscript{47}. Therefore, the relationship between bonding strength and mechanical properties will have to be examined with a technique
that allows the ability to control both uniformity in bonding and distribution. Molecular simulations are perfect for this examination; since, initial bonding and distribution can easily be controlled by setting initial positions of the nano particle, and the polymer’s bonds to the nano particle. The simulations will hopefully be able to prove or disprove that stronger bonding leads to better mechanical properties (Composite Theory).

One material variable, molecular weight, was not discussed in any paper. Though molecular weight was not discussed in any paper, molecular weight is important for defining the base properties of the polymer. The molecular weight could also play an important role in how effective the nano particle is in modifying the properties of the polymer, and therefore molecular weight could be interesting to study.

Above is an examination of the different material variables and their effects on polymer-metal oxide nano composites. Studying some of these variables through computer simulation should give a better understanding of how these variables effect the material and lead to better future designs of polymer-metal oxide nano composites.

2.6 References


CHAPTER 3

METHODOLOGY

3.1 Objectives

3.1.1 Developing Metallic Potentials

- Invent methodology for quickly estimating metallic potentials
- Develop software to implement new methodology
- Compare newly created potentials with past potentials
- Discover efficiencies, deficiencies of methodology

3.1.2 Studying PMMA-Metal Oxide Nano Composites

- Utilize MD simulations to determine effects of design variables on tensile properties
- Manipulate design variables: mass percent of filler and percentage bonds to filler
- Examine composite interface and PMMA molecular changes
- Find relationships between the design variables and the interface, molecular changes, and tensile properties
- Produce theoretical model to explain relationships

3.1.3 Producing Computational Tools

3.1.3.1 Metallic Potentials

- Implement new methodology for estimating metallic potentials
- Allow user to select properties
- Include interface to connect developed software to MD/MC programs

3.1.3.2 PMMA-Metal Oxide Nano Composites

- Create algorithm for controlled bonding of polymers to nano particles
- Test new method of initializing polymer for composite MD simulations
3.2 Metal EAM Potential Fitting Algorithm

3.2.1 Framework Design

The framework design for fitting potentials consists of three main components: the potential fitting software, the controls, and the database. Connectivity of these components are shown in Figure 3.1. The controls affect the specific algorithms chosen by the potential fitting software. The database stores important information needed by the potential fitting software. All three components will be explained in more detail below.

![Diagram of framework components](image)

Figure 3.1: The three components of the framework and how they connect.

The potential fitting software consists of five modules: Main Controller, Optimization, Potential Creation, Fitting Properties, and Testing Properties. These five modules and how they connect to each other are shown in Figure 3.2. Each module’s algorithm is chosen at runtime using the controls. The Main Controller is designed to choose the best fitted potential, and control the selection of input parameters into Optimization. The best potential has the lowest sum of residuals squared from the Testing Properties. The reason behind this selection and the definition of testing properties can be found in Mishin et al. The Optimization module contains optimization algorithms used to minimize the objective function. The objective function is the sum of residuals squared from the Fitting Properties.
The optimization algorithm finds the minimum by changing the parameters defining the interatomic potential. These parameters are passed to the Potential Creation module where the parameters are used to build the potential as a numerical table. Afterwards, the selected properties in the Fitting Properties module are calculated. From the calculated values, the objective function is calculated and passed to the Optimization module. This loop continues until the selected optimization algorithm runs out of iterations or determines the objective function reached a minimum. The Testing Properties module calculates different selected properties then the Fitting Properties module. These calculated properties are used to calculate the sum of residuals squared needed by the Main Controller module.

Figure 3.2: The design of the potential fitting software.

The “controls” are a series of text files which enable run time selection of the different algorithms or calculations in each module. The text files are divided into algorithm selection and calculation selection. The calculations are subdivided into fitting and testing calculations. This gives a total of three files.
The algorithm selection file contains the number of elements, name of the elements, optimization algorithm, and interatomic potential. The number of elements and interatomic potential control which algorithm is used in the Main Controller module. The optimization algorithm controls the selection process in the Optimization module. The interatomic potential controls which potential is selected in the Potential Creation module. The name of the elements is used by the Potential Creation module to name the potential. The name of the elements is also used to access the specific elements database files.

The two calculation files control the Testing Properties module and Fitting Properties module. Each module has a separate file which contains the number of properties to be calculated and which properties the module should calculate.

The database contains many files designed to store important information for each metal. For the framework design the database requires material properties for potential production, and calculated material properties and their weights for fitting. The database may require sets of initial conditions for the Optimization module depending on which algorithm is selected in the Main Controller module. To organize the database by metal, the elemental composition without the numbers is placed in front of the file name. An example of this format is in Appendix L.

### 3.2.2 Test Case

In order to test the framework design, a new potential needed developing and testing. This potential would be the test case for the framework design.

The description of the potential and the number of elements affect how the main controller operates. The potential can be written as either a series of equations or splines. For the case where the potential is constructed with splines for one element, the algorithm in the Main Controller module has been solved by Mishin et al\(^1\). For the other cases, multiple algorithms exist but the best solution is not clear. For these reasons, the framework design utilized the algorithm by Mishin et al\(^1\) in the Main Controller module, and a potential constructed with splines using the same formulation in Mishin et al\(^1\).
The optimization algorithm selected for the Optimization module was the Nelder-Mead simplex\(^2\) with slight modifications from the original algorithm\(^3\). This algorithm has been used with great success in fitting many EAM potentials including Mishin et al\(^1\).

The Fitting and Testing module property calculation selections are shown below in Table 3.1. They are designed to produce a potential which is capable of calculating thermal mechanical properties. To decrease the time required for fitting a potential, all properties calculated in the Fitting module are at 0k and do not require molecular dynamics. These properties can be calculated in seconds. The Testing module properties are calculated at room temperature using molecular dynamics which can take up to an hour to finish.

<table>
<thead>
<tr>
<th>Fitting</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energetics</td>
<td>Thermal</td>
</tr>
<tr>
<td>Elastic Constants</td>
<td>Thermal Expansion</td>
</tr>
<tr>
<td>Vacancy Formation Energy</td>
<td>Lattice Constant (298.15 K)</td>
</tr>
<tr>
<td>Structures</td>
<td>Thermal Conductivity</td>
</tr>
<tr>
<td>HCP Energy</td>
<td></td>
</tr>
<tr>
<td>Faults</td>
<td></td>
</tr>
<tr>
<td>Stable Stacking Fault</td>
<td></td>
</tr>
<tr>
<td>Energy-Volume</td>
<td></td>
</tr>
<tr>
<td>Rose et al. Energy</td>
<td></td>
</tr>
</tbody>
</table>

The Rose et al. energy\(^4\), stable stacking fault, vacancy formation energy, and hexagonal close packed (HCP) energy were calculated using the methods described in Mishin et al\(^1\). The elastic constants were calculating using the method described in Zhu et al\(^5\). The thermal conductivity was calculated using
the algorithm developed by Muller-Plathe and Reith in LAMMPS with a 20x20x20 face center cubic (FCC) super cell. The thermal expansion coefficient

Table 3.2: Property values used for fitting and testing nickel potentials and the calculated values of the nickel potential selected by the Main Controller module. The properties in grey boxes are guaranteed to be the experimental values because of how the potential is constructed, and the properties in white blocks are calculated. The lattice constants and thermal conductivity values in the experimental column are calculated from experimental relationships with other property values.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Experimental</th>
<th>Calculated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lattice Constant 0K (Å)</td>
<td>3.5147</td>
<td>3.5147</td>
</tr>
<tr>
<td>Lattice Constant Room Temperature (Å)</td>
<td>3.524</td>
<td>3.524</td>
</tr>
<tr>
<td>Bulk Modulus (GPa)&lt;sup&gt;9&lt;/sup&gt;</td>
<td>190.3</td>
<td>190.3</td>
</tr>
<tr>
<td>Elastic Constant C11 (GPa)&lt;sup&gt;9&lt;/sup&gt;</td>
<td>261.4</td>
<td>283.4</td>
</tr>
<tr>
<td>Elastic Constant C12 (GPa)&lt;sup&gt;9&lt;/sup&gt;</td>
<td>154.8</td>
<td>143.7</td>
</tr>
<tr>
<td>Elastic Constant C44 (GPa)&lt;sup&gt;9&lt;/sup&gt;</td>
<td>130.9</td>
<td>123.7</td>
</tr>
<tr>
<td>Cohesive Energy (ev/atom)&lt;sup&gt;10,11&lt;/sup&gt;</td>
<td>-4.44</td>
<td>-4.44</td>
</tr>
<tr>
<td>Hexagonal Close Pack Energy (ev/atom)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>-4.42</td>
<td>-4.46</td>
</tr>
<tr>
<td>Vacancy Formation Energy (ev/vacancy)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>1.6</td>
<td>1.66</td>
</tr>
<tr>
<td>Stable Stacking Fault Energy (mJ/m&lt;sup&gt;2&lt;/sup&gt;)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>125</td>
<td>124</td>
</tr>
<tr>
<td>Linear Thermal Expansion Coefficient (10&lt;sup&gt;-6&lt;/sup&gt;/°C)&lt;sup&gt;12&lt;/sup&gt;</td>
<td>13.4</td>
<td>12.4</td>
</tr>
<tr>
<td>Thermal Conductivity(W/m/K)&lt;sup&gt;13&lt;/sup&gt;</td>
<td>4.177</td>
<td>3.817</td>
</tr>
</tbody>
</table>

and lattice constant at room temperature were calculated using LAMMPS and a 20x20x20 FCC super cell. The super cell was equilibrated at two temperatures using NVE integration combined with a Berendson temperature and pressure
control for 5,000 time steps with a delta time value of 0.001 psec. The two
temperatures were 293 K and 300 K. After the initial equilibration, the super cells
at both temperatures were simulated for 10,000 time steps with a NPT controller
and a delta time value of .001 psec. The average volume of both NPT
simulations was calculated. The two average volumes and temperatures were
used to calculate the linear thermal expansion using Equation 3.1, i.e.
\[ \alpha = \frac{V_{300} - V_{293}}{T_{300} - T_{293}} \times \frac{1}{V_{293} \times 3 \times 10^{-6}}, \] (3.1)
which is the volume thermal expansion divided by \(3^8\). The average volume at 293
K was used to calculate the lattice constant at room temperature i.e.,
\[ a_{293} = \frac{\sqrt{V_{293}}}{20}. \] (3.2)

Table 3.2 shows property values used for fitting and testing nickel
potentials and the calculated values of the potential selected by the Main
Controller module. Nickel was the element selected for this study due to the large
number of EAM potentials developed which allow for easy comparison. The
experimental values of the elastic constants and bulk modulus are measured
near 0 K. The cohesive energy, vacancy formation energy, stable stacking fault
energy are assumed to be temperature invariant. The linear thermal expansion
coefficient was measured experimentally at room temperature (293 K). The 0 K
lattice constant is calculated using a theoretical relationship between constant
pressure heat capacity and lattice constants explained in the next section. The
room temperature lattice constant is calculated from the experimental density
(8.908 g/cm\(^3\)) at room temperature (293 K) using the density data from Bower et
al.\(^9\). The thermal conductivity-size dependency relationship was calculated using
molecular dynamics\(^{13}\). The value in Table 3.2 is for a box length of 20 lattice
units. A molecular dynamics derived value was required because traditional atom
and interatomic potential MD simulations are incapable of calculating the correct
thermal conductivity of metals due to the fact that the thermal conductivity
derives from electron migration rather than phonons\(^{14}\).
In Table 3.2, the near 0 K elastic constants, bulk modulus and lattice constant are fit into the potential when usually the room temperature values are used. A graphical reason for this can be shown in Figure 3.3. Figure 3.3 shows four Al potential lattice constant temperature relationships. From the figure, most of the potentials 0K lattice constants actually are the lattice constants at room temperature and these potentials do not correspond to the lattice constant temperature relationship. The “adamsAl.eam.alloy” potential does correspond to the lattice constant temperature relationship. The differences between the potentials are intentional. In the first set of potentials with no correspondence to lattice constant temperature relationship, the potentials are fit to the room temperature properties to allow researchers to do room temperature calculations without having to do molecular dynamics.
In the case of “adamsAl.eam.alloy”, the force-matching method was used which allows researchers to include both 0K properties and molecular dynamics into the fitted potential\textsuperscript{15}. Unfortunately, the force-matching method has yet to be tested with a partially filled d orbital. According to the POCO (software package containing the force-matching method) website, the method has been used to fit aluminum, magnesium, copper, lead, and silicon\textsuperscript{18}. These metals are all in A columns (have no d orbitals or the d orbitals have been completely filled) of the periodic table except copper\textsuperscript{19}. Even though Copper is in column 1B, the d orbitals are completely filled\textsuperscript{19}. Whether the lack of fitted partially filled d orbital metals for this method is due to accuracy issues in the required quantum calculations or to lack of exploration is not known. What is known is the lack of partially filled d orbitals greatly increases the accuracy of quantum mechanics calculations\textsuperscript{15}. The property values in Table 3.2 combined with the fitting-testing design in Table 3.1 are designed to expand the forced fitting method capabilities to metals with unfilled d orbitals (transition metals).

### 3.2.3 Testing Potential

To assure the accuracy of the interatomic potential selected by the Main Controller module, more properties were calculated and compared to their experimental values. The properties examined were the lattice constant and the Young’s modulus. These properties were calculated from 0K to 293K.

The lattice constants from the potentials were calculated using LAMMPS. The software was used to find the equilibrium volume of a 20x20x20 FCC super cell at different temperatures. The temperatures examined were 0, 25, 50, 75, 100, 150, 200, 250, and 293 K. The equilibrium volume was converted to the lattice constants using Equation 3.2, where $V_{293}$ is the equilibrium volume at the examined temperature. Above 0 K, the equilibrium volume was found using the same molecular dynamics methods for finding the average volume in the thermal expansion calculations. At 0 K, the equilibrium volume was found using conjugate gradient minimization to relax the pressure in the box to 0 bars.
The lattice relationship on the graphs was derived from a theoretical lattice-energy relationship. This relationship is

\[ a_T = a_0 e^{\int_0^T C_v \, dt}, \]  

(3.3)

where \( a_T \) is the lattice constant at temperature \( T \), \( C_v \) is the constant volume heat capacity integrated over temperature, \( a_0 \) is the lattice constant at 0 K and \( p \) is a fitted parameter\(^{20} \). The derivation for Equation 3.3 was later shown in Mitra and Giri\(^ {21} \). In this derivation, \( p \) included the Grüneisen parameter\(^ {22} \) which allows Equation 3.3 to be modified to use the constant pressure heat capacity, \( C_p \), instead. This modification,

\[ a_T = a_0 e^{\int_0^T C_p \, dt}, \]  

(3.4)

is used in the lattice constant temperature relationships throughout this article. Since \( C_p \) is not defined continuously between 0K and \( T \); an interpolation method is required to form a continuous equation or equations between the known temperature values of \( C_p \) taken from Hultgren et al\(^ {23} \). Natural cubic splines were chosen to form the set of continuous equations between the known \( C_p \) data. The new integration becomes

\[ \int_0^T C_p \, dt = \sum_{i=1}^{n_T-1} \int_{T_i}^{T_{i+1}} C_{pi} + b_i(t-t_i) + c_i(t-t_i)^2 + d_i(t-t_i)^3 \, dt, \]  

(3.5)

where \( n_T \) is the number of known temperature values needed to calculate the integral, \( T_i \) is the lower interval known temperature value, \( T_{i+1} \) is either the upper interval known temperature or \( T \), \( C_{pi} \) is the constant pressure heat capacity at the known value \( T_i \), and \( b_i, c_i, d_i \) are the natural cubic spline variables at the known value \( T_i \). To calculate \( a_T, a_0 \) and \( p \) need to be fitted using three lattice constants. One of these lattice constants is calculated using the room temperature density. The other two lattice constants are calculated slightly above and below that temperature using the linear thermal expansion coefficient. Using Equation 3.5, the energy at these temperatures were calculated. Than, \( a_0 \) and \( p \) were fitted by minimizing the difference between the three actual lattice constants and the three calculated using Equation 3.4. Table 3.3 shows the \( a_0 \) calculated using Equation
3.4, experimental $a_0$, and $a_0$ calculated using Equation 3.3. The lattice constants in Table 3.3 are the same to at least 3 digits which shows Equations 3.3 and 3.4 are viable methods for calculating the 0 K lattice constants.

Table 3.3: Different values of 0K lattice constants for copper and aluminum.

<table>
<thead>
<tr>
<th>Element</th>
<th>Equation 3.4 (Å)</th>
<th>Experimental (Å)</th>
<th>Equation 3.3 (Å)</th>
</tr>
</thead>
<tbody>
<tr>
<td>copper</td>
<td>3.6029</td>
<td>3.6029</td>
<td>3.6024</td>
</tr>
<tr>
<td>aluminum</td>
<td>4.0314</td>
<td>4.0318</td>
<td>4.0317</td>
</tr>
</tbody>
</table>

The other lattice constant relationship, for nickel only, was fitted to experimental data. The form of the relationship found by Bandyopadhyay and Gupta is

$$a_T = \sqrt{\alpha_0^2 \left( 1 + \frac{T^2}{\beta_0^2} \right) + k_0^2}.$$  \hspace{1cm} (3.6)

The parameters fitted to Equation 4.6 are shown in Table 4.4. Table 4.5 shows the lattice constant at room temperature and 0 K for Equation 4.6 and Equation 4.4.

Table 3.4: Parameters for Equation 3.6.

<table>
<thead>
<tr>
<th>alpha (Å)</th>
<th>beta (Å)</th>
<th>k (Å)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.02518</td>
<td>344.61</td>
<td>3.4903</td>
</tr>
</tbody>
</table>

Table 3.5 shows both relationships produce similar values at room temperature and 0K further verifying Equation 3.4’s validity. Additionally the room temperature values in Table 3.5 are close to the calculated value of 3.524 Å in Table 3.2 and 3.5238 Å in Eberhart and Horner

The Young’s modulus were calculated using molecular dynamics in LAMMPS at different temperatures for each potential. The temperatures were 0.5, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, and 293 K. A 10x10x10 FCC
Table 3.5: Ni lattice constants for two different relationships at room temperature and 0K.

<table>
<thead>
<tr>
<th>Temperature (K)</th>
<th>Equation 3.4 (Å)</th>
<th>Equation 3.6 (Å)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.5147</td>
<td>3.5155</td>
</tr>
<tr>
<td>293</td>
<td>3.5241</td>
<td>3.5236</td>
</tr>
</tbody>
</table>

supercell was equilibrated using a NPT controller for 30,000 steps with a delta time of 0.001 ps. Than a tensile test in molecular dynamics was performed using a strain rate in the x direction of 1.5e9 s⁻¹ and a NPT controller in the y and z direction to ensure the pressure was kept at zero. This tensile test was ran for 1,000 steps with a delta time of 0.001 ps. Afterwards, the strained sample was equilibrated by keeping the strain from the previous step constant in the x direction and varying the size of the box in the y and z direction using an NPT controller to ensure the pressure was zero. The equilibration procedure took 10,000 steps with a delta time of 0.001 ps. The stress perpendicular to the x face was averaged for the steps in the final equilibration. This stress was than divided by the fixed strain during that procedure to get the Young’s modulus M defined by

\[
M = \frac{\sigma_{xx}}{\varepsilon_{xx}},
\]

where \(\sigma_{xx}\) is the stress perpendicular to the x face, \(\varepsilon_{xx}\) is the strain perpendicular to the x face.

Alers et al. contains experimental values of nickel elastic constants for a range of temperatures\textsuperscript{26}. These elastic constants can be used with

\[
M = C_{11} - \frac{2C_{12}^2}{C_{11} + C_{12}}
\]

(3.8)

to calculate the Young’s modulus assuming the material is cubic and strained in the \(<1 0 0>\) family of directions. Equation 3.8 was derived but matches the equation used by Zhu et al\textsuperscript{27}.
3.3 Bonding Algorithm

To further the capabilities in preliminary material design, work was also done on nano composites. For the composite material, new methods for bonding the nano particle to the polymer and initializing the polymer are explained in the following sections.

In order to bond polymer chains to ceramic nano particles, a ten step approach was utilized. The steps in order were:
1. reading a data file containing polymer chains and nano particles
2. splitting the polymer chains’ molecules from the data
3. splitting the nano particles' molecules from the data
4. separating the nano particle surfaces from the nano particles
5. finding bond locations between polymer chains and nano particle surfaces
6. bonding the polymer chains to the nano particle surfaces at those locations
7. deleting atoms on the nano particle surfaces at those locations
8. adding or deleting atoms on the nano particle surface to obtain charge neutrality
9. forming a bonded nano composite from the modified polymer chains and nano particles
10. writing a new data file containing the bonded nano composite.

The best way to implement these steps was by using an object oriented approach where the objects represent the composite, the molecules and the nano particle surfaces. Each of these objects are explained below. Additional explanation of these steps can be found in Appendix J.

3.3.1 Composite

The composite exists in modeling as a set of data containing atom information for both the nano particle and the polymer matrix. This data can either be read from a data file or written to a data file. After the bonding process,
the atom information is written to a data file for further MD simulation of the bonded composite.

### 3.3.2 Particle Surface

The particle surface is the surface of the nano particle. For the bonding algorithm, this surface is vitally important because the bonds between the polymer matrix and the nano particle form here. Additionally, any atoms added or subtracted to obtain charge neutrality also occurs at this surface.

### 3.3.3 Molecules

The molecules in this algorithm are groups of atoms which either form a polymer chain or a single nano particle. The particle surface is separated from nano particles. The two bonding methods, steps 5 and 6, occur with the polymer chains.

The two bonding methods are the key steps to form a bonded nano composite. The first method, step 5, is explained by Figure 3.4. Figure 3.4 shows there is a region around the nano particle surface where atoms in the polymer chain can form bonds. To find these locations, the algorithm requires the following inputs: which type of atoms on the polymer chain can form bonds, how large is the bonding region, and what number of bonds is formed. Figure 3.5 explains the bond formation and deletion on the polymer chain for the second method, step 6. Figure 3.5 shows a portion of a PMMA chain in which the central carbon atom is bonded to the nano particle surface (shown as a blue circle). Because this carbon now has five bonds, one of
the bonds needs to be deleted to stabilize the system. The red triangle with square inset pattern represents the one bond which will be maintained because its attached to the nano particle surface. The green triangles with triangular stripes represents the bonds which may be deleted because they are part of the polymer chain. In this case one of the green triangles will be deleted, thus stabilizing the system. Each triangle contains a single bond sticking off the central atom (the central carbon atom) and the neighboring atom attached to that bond. Additionally the triangle includes the other bonds and their atoms attached to the neighboring atom. The algorithm requires the bonding location on the polymer chain and the nano particle surface, and the correct number of bonds sticking off the central atom to be deleted. The method will then define the bonding location of the polymer chain as the central atom and delete the triangle(s) defined by the given bonds.
3.3.4 Testing

To test the ten step approach, a python module was developed which contained the three objects discussed above. The module was designed to interface with LAAMPS data files. During initial code testing certain portions of the code were found to run extremely slow even after optimization and so parallel computing was implemented. Unfortunately, due to the global interpreter lock, true parallelism can only be accomplished through multiprocessing\(^\text{28}\). Since each process has its own memory, for a large composite system or a large number of paralleled processes, the computer may actually run out of memory\(^\text{28}\). For these cases, multiprocessing should be turned off in the module. The module was tested using PMMA-alumina nano composite.

![Figure 3.5: Representation of PMMA bonded to the nano particle surface (blue circle). The red triangle with square stripes is kept during bonding. The green triangles with triangular inset pattern may be deleted during bonding.](image)
3.3.5 Chemistry

In order to run the test case on the PMMA-alumina nano composite, the chemistry needs to be well defined. For PMMA-alumina the bonding reaction is

\[
\text{Al}_2\text{O}_3 + \text{H}_2\text{O} + 2\text{CH}_3\text{C(\text{CH}_3)(\text{COOCH}_3)}^- \rightleftharpoons 2\text{AlO}^-(\text{CH}_3\text{C(\text{CH}_3)(\text{COO})})^- + 2\text{CH}_3\text{OH}^{29} \tag{3.9}
\]

If any simplifications to Equation 3.9 can be made due to assumptions, these simplifications should be used because they will greatly simplify the process for forming the bonded polymer nano composite in the MD simulation box.

The methanol produced in the MD simulation box can be assumed to have a negligible effect on MD calculated properties compared to PMMA units bonded to alumina. The methanol does not interact strongly with the alumina and PMMA unlike the PMMA units bonded to alumina, which have strong ionic interactions with the alumina and PMMA, and apply strong forces to the PMMA chain. Also the methanol is produced in such a small quantity relative to the alumina and PMMA. For these reasons, the methanol can be removed from the simulation box as the reaction occurs.

Since the methanol will be removed from the simulation box, the methanol can be broken into separate ions. This version of the reaction is shown in Table 3.6 under the explicit atom simulation type. This version simplifies the steps needed for computational bonding because now the OH- and CH3+ do not need to be bonded together to form methanol. Instead, the ions can be removed directly from the simulation box.

Table 3.6 also has the same reaction written in the united atom method. This method removes the hydrogen from the system. For testing the Python module, the united atom method reaction will be used.
Table 3.6: Bonding process in the MD simulation box.

<table>
<thead>
<tr>
<th>Simulation Type</th>
<th>Reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit Atom</td>
<td>[ \text{Al}_2\text{O}_3 + \text{H}_2\text{O} + 2[-\text{CH}_2\text{C}(\text{CH}_3)(\text{COOCH}_3)-] \rightarrow 2\text{AlO}^+(\text{CH}_2\text{C}(\text{CH}_3)(\text{COO}))^- + 2\text{OH}^- + 2\text{CH}_3 ]</td>
</tr>
<tr>
<td>United Atom</td>
<td>[ \text{Al}_2\text{O}_3 + \text{O} + 2[-\text{CC}(\text{C})(\text{COOC})-] \rightarrow 2\text{AlO}^+(\text{CC}(\text{C})(\text{COO}))^- + 2\text{O}^- + 2\text{C}^+ ]</td>
</tr>
</tbody>
</table>

**3.4 Polymer Initialization Algorithm**

In the ideal case, a novel polymer initialization algorithm for nano composites would be tested using nano composites. Unfortunately at this time, there is not enough data to know explicitly what a nano composite’s properties will be. Because of this a well known polymer material will be used as the test case for the new algorithm. The polymeric material chosen for the test case is PMMA because the properties are well known and have been extensively tested. Additionally potentials for this material have been developed and well tested in the literature.

**3.4.1 Potentials**

The potential used for simulating PMMA is a modified version of that used by Okada et al\textsuperscript{30}. This potential describes PMMA using the united atom method. A united atom representation of PMMA is shown in Figure 3.6. The notation used in Figure 3.6 will be used to describe the modifications made to the Okada et al\textsuperscript{30} potential. This notation is similar to the one used in Okada et al\textsuperscript{30} to make comparison between their work easier. The modifications were required because the equation formats to describe some of the intramolecular forces were not available in LAMMPS. An equivalent but different form of the equation was selected if possible. For intramolecular forces not discussed below, this dissertation uses the Okada et al\textsuperscript{30} equations and parameters. The intermolecular forces were described using Leonard Jones and Coulombs potentials using the parameters as described in Okada et al\textsuperscript{30}.
For the dihedral angles, the original form of the equation,
\[ U = V_1 \cos \phi + V_2 \cos 2\phi + V_3 \cos 3\phi + V_6 \cos 6\phi, \]  
(3.10)
is replaced with
\[ U = \sum_{n=1,5} A_n \cos^{n-1} \phi \]  
(3.11)
an equivalent form of Equation 3.10, in this work. In Equation 3.10 and 3.11, the parameter \( U \) is the potential energy; \( \phi \) is the dihedral angle; \( V_i \) are parameters, and \( A_n \) are parameters. The parameters for both equations are shown in Table 3.7.
Table 3.7: Parameters for dihedral Equation 3.10 and 3.11. The upper row corresponds to the notation used in Figure 3.6. The “?” represents c1 and c2. All parameter values are listed in kilo calories per mole.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>c?-c4s1-</th>
<th>c4s1-c2-</th>
<th>c?-c4s1-</th>
<th>c4s1-cds-ods</th>
<th>c4s1-cds-o2s</th>
<th>ods-cds-o2s-c1s2</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>c2-c4s1</td>
<td>c4s1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V1</td>
<td>3.5</td>
<td>3.5</td>
<td>2.4</td>
<td>0.0</td>
<td>4.07</td>
<td>0.0</td>
</tr>
<tr>
<td>V2</td>
<td>0.0</td>
<td>0.7</td>
<td>0.0</td>
<td>-1.9</td>
<td>-2.23</td>
<td>-2.20</td>
</tr>
<tr>
<td>V3</td>
<td>0.87</td>
<td>0.87</td>
<td>0.44</td>
<td>0.26</td>
<td>-0.16</td>
<td>-0.35</td>
</tr>
<tr>
<td>V6</td>
<td>0.05</td>
<td>0.05</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>A1</td>
<td>0.00043</td>
<td>-0.69938</td>
<td>-0.00012</td>
<td>1.90050</td>
<td>2.22989</td>
<td>2.20044</td>
</tr>
<tr>
<td>A2</td>
<td>0.88728</td>
<td>0.88749</td>
<td>1.08001</td>
<td>-0.78234</td>
<td>4.54992</td>
<td>1.04739</td>
</tr>
<tr>
<td>A3</td>
<td>-0.0059</td>
<td>1.39251</td>
<td>0.00043</td>
<td>-3.80681</td>
<td>-4.45939</td>
<td>-4.40604</td>
</tr>
<tr>
<td>A4</td>
<td>3.48522</td>
<td>3.48489</td>
<td>1.76002</td>
<td>1.04477</td>
<td>-0.63999</td>
<td>-1.39492</td>
</tr>
<tr>
<td>A5</td>
<td>0.00840</td>
<td>0.01003</td>
<td>-0.00036</td>
<td>0.00941</td>
<td>-0.00051</td>
<td>0.00856</td>
</tr>
</tbody>
</table>

The inversion potential,

\[ U = K_1 \left( \theta - \theta_0 \right) + K_2 \left( \theta - \theta_0 \right)^2, \]  

(3.12)

was converted to an improper potential,

\[ U = K \left( \chi - \chi_0 \right)^2, \]  

(3.13)

in LAMMPS. \( K_1, K_2, \) and \( K \) are fitted parameters for Equations 3.12 and 3.13. \( \theta \) is the sum of the three bond angles formed around cds in Figure 3.6. \( \theta_0 \) is the equilibrium value of the sum of these angles. \( \chi \) in Equation 3.13 is the improper angle which measures how far out of plane cds is with respect to os2, ods, and c4s1. \( \chi_0 \) is the equilibrium value of the improper angle. To convert between the system used in Equation 3.12 and Equation 3.13 a code was written in Fortran 77 (Appendix A) which calculates \( \theta \) and \( \chi \) of 4096 atom position combinations around the central atom cds. The energies for Equations 3.12 and 3.13 were
than calculated. The correct parameter $K$ in Equation 3.13 was found by minimizing the least squared error between the two energies.

Table 3.8: Parameters for inversion Equation 3.12 and improper Equation 3.13.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_1$ (Kcal/mol/rad)</td>
<td>-60.0</td>
</tr>
<tr>
<td>$K_2$ (Kcal/mol/rad$^2$)</td>
<td>30.0</td>
</tr>
<tr>
<td>$\theta_0$ (degrees)</td>
<td>360.0</td>
</tr>
<tr>
<td>$K$ (Kcal/mol/rad$^2$)</td>
<td>0.73501</td>
</tr>
<tr>
<td>$\chi_0$ (degrees)</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

The 1,5 nonbonding terms in Okada et al.$^{30}$ were removed for two reasons. First, the 1,5 nonbonding terms would require extensive recoding in LAMMPS and adding the capability for the software to recognize 1,5 neighbors would be extremely difficult, if not impossible. Additionally, the modified PMMA potential will later be used for modeling PMMA interactions with an alumina nano particle. According to quantum mechanics calculations of the PMMA-alumina interface the PMMA carbonyl oxygen reacts strongly to the aluminum$^{31}$. This strong interaction with aluminum causes the structure of PMMA to alter from its normal state$^{32}$. The 1,5 nonbonding terms act to both stiffen the polymer and restrict the confirmations. When PMMA is close enough to strongly interact with the alumina, the 1,5 nonbonding terms are not needed; but, when PMMA is far enough away to weaken the alumina interaction, the 1,5 nonbonding term may be necessary. Unfortunately, applying a 1,5 nonbonding term a certain distance away from the nano particle is impossible due to the derivative at the boundary being infinite and uneven forces between atoms interacting across the boundary, which violates Newton’s third law. Hence, the 1,5 nonbonding terms need to be removed, and all modifications to Okada et al.$^{30}$ require confirmation of their accuracy in reproducing PMMA properties in order for the modified potential to be later used in modeling PMMA-alumina nano composite.
3.4.2 Creating Polymer Simulation Box

To produce the nano composites for MD research a random walk algorithm which avoids the nano particle(s) needs to be developed. As part of this development process a novel RNG algorithm is invented. Both algorithms require testing to show they reproduce properties for a known material, in this case PMMA. The methodology for both algorithms is described below along with any modifications used specifically for testing PMMA.

The algorithms developed below specifically apply to a single nano particle centered at the origin of the simulation box. The nano particle in the algorithm is spherically shaped.

With random walk algorithms there are two possible methods. One method is to use a RNG for every unit placed. The second method is to use the RNG every nth step where n is equal to the number of units in a Kuhn length. The second method is used for the random walk algorithm throughout this dissertation.

In the random walk algorithm both real space and periodic space are used. Real space is normal three dimensional space while periodic space is the simulation box which uses periodic boundary conditions (PBC). The nano particle is stored in periodic space. So every time a polymer unit is tested if its inside the space allocated to the nano particle, the test is done on the polymer unit in periodic space. Additionally, since LAMMPS will be run using PBC, the polymer chain is written to a LAMMPS data file in its periodic space coordinates. All other calculations are done using real space.

The shift vector is the vector used to translate polymer units out of the space allocated for the nano particle. This translation is done after the chain has been placed as an error correction step for any units or atoms which may have accidentally been placed in the nano particle’s space by the RNG algorithm. For a single nano particle, this step may be an unnecessary safety precaution. But for multiple nano particles, this step is required. The final translated location can be described mathematically as,
\( \vec{x}' = \vec{x} + (r_{\text{sphere}} + d_{\text{safety}})(\vec{p} - \vec{p}), \)  \hspace{1cm} (3.14)

where \( x \) is the position of the atom or unit in real space, \( r_{\text{sphere}} \) is the radius of the nano particle, \( d_{\text{safety}} \) is the safety distance to translate the atom or unit off the nano particle surface, \( p \) is the position of the atom or unit in periodic space. In this work, \( d_{\text{safety}} \) is set at 10%.

### 3.4.2.1 Random Walk Algorithm

1. Place first unit of the new chain randomly inside the simulation box.
2. If the first unit is located inside the space taken up by the nano particle redo step 1.
3. Shift over by the addition vector and place the next unit of the chain in that location.
4. If the current unit of the chain is located inside the space taken up by the nano particle than run the RNG algorithm and reset the number of units placed to 0. Skip to 7.
5. If the number of units placed in the chain is equal or greater than the kuhn count, run the RNG algorithm and reset the number of units placed to 0. The kuhn count can be set to zero for a simple random walk or the number of units in a kuhn length for a kuhn length random walk. Skip to 7.
6. Add the number of monomers in a unit to the Kuhn count.
7. Go to 3 unless the end of the chain has been reached.
8. If there is no nano particle skip to step 11 otherwise go to step 9.
9. If any of the units are located inside the space taken up by the nano particle, calculate the shift vector needed to move the unit outside of the space.
10. Use the shift vector to translate the unit outside the space taken by the nano particle.
11. Repeat step 1 through 10 until all chains have been placed in the simulation box.

### 3.4.2.2 RNG Algorithm

Under certain assumptions, the RNG algorithm can be done using real space even though the nano particle is in periodic space. In order to make these assumptions, the system requires a single nano particle centered on the origin in the center of periodic space. These requirements allow the RNG algorithm to be simplified; the original version written in periodic space is shown in appendix C.
The assumptions which allow this simplified version to be used are the Kuhn length is much shorter than the length between the boundary of the simulation box and the nano particle surface, and most of the polymer chain is in the periodic box centered around the origin.

Figure 3.7: Diagram of mirror based RNG algorithm before new unit and addition vector have been reflected across the mirror plane.
In LAMMPS this box has the image flags (0,0,0) which corresponds to the real coordinates and periodic coordinates being equivalent. The assumption about the Kuhn length insures any polymer which is not in the periodic box centered around the origin will not come close to contacting the nano particle in the other periodic boxes. The explanation for the simplified RNG algorithm in real space is shown below.

Figure 3.7 shows a diagram of the RNG algorithm. In this diagram there are two units an old unit and a new unit. The new unit is the old unit translated by the addition vector. The addition vector is attached to the mirror point which is the last sub unit or atom in the old unit that is connected to the new unit. At the mirror point a mirror plane exists with the equation

\[ ax + by + cz + d = 0 , \]  

(3.15)

and a vector normal to the plane equal to

\[ \vec{n} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \vec{x}_{\text{mirror point}}, \]  

(3.16)

where \( \vec{x}_{\text{mirror point}} \) is the coordinate of the mirror point in real space. The value of \( d \) in the mirror plane can be calculated using the equation

\[ d = -\vec{n} \cdot \vec{n}. \]  

(3.17)

The new unit is than reflected in the mirror plane. The new positions of atoms/sub units from this reflection can be described using this equation,

\[ \vec{x}'_{\text{atom}} = \vec{x}_{\text{atom}} - 2 \frac{\vec{x}_{\text{atom}} \cdot \vec{n} + d}{\vec{n} \cdot \vec{n}} \vec{n}, \]  

(3.18)

where \( \vec{x}_{\text{atom}} \) is the positions of the atoms/sub units in real space before the reflection. After the reflection is done the addition vector is recalculated using the new positions of the atoms in the new unit. This new addition vector is used for future translations of units.

This algorithm does not guarantee that the new unit after the reflection will be completely out of the nano particle space. But the change in the addition
vector, under most circumstances guarantees future unit translations will avoid the nano particle space. Additionally, the algorithm under most circumstances guarantees the new unit will avoid the old unit. In other words this algorithm is a pseudo RNG algorithm which allows the random walk algorithm to be a nano particle avoiding walk near the nano particle.

3.4.3 MD Simulations and Property Calculations

In order to prove the modified potential and novel random walk algorithm are correct, MD was used to calculate the glass transition temperature, Young’s modulus, and radius of gyration. MD was also used to calculate the molecular energy and density. The molecular weight was chosen to be 23,000 g/mol to allow direct comparison with existing experimental calculations of the radius of gyration. Three RNG seeds, 3, 7, and 14, were used by the random walk algorithm to build three separate simulation boxes for each box size. Five different box sizes were used. They are, in angstroms, 63.74, 74.94, 86.51, 96.77, 132.58. The methods for the MD simulations and property calculations are explained below.

3.4.3.1 Initial Equilibrium

This step allows the distance between PMMA atoms to be increased. After, the PMMA has been built in the box, the distance between atoms is too small to use normal NVE, NVT, or NPT integration methods. In this step, the NVE/limit algorithm in LAMMPS was used. This algorithm modifies the NVE integration so atoms can only move a specified maximum distance which allows atoms to slowly spread apart and eventually reach an equilibrium separation distance. The NVE/limit algorithm was ran for 10,000 steps with a time step of one fs.

3.4.3.2 Final Equilibrium

In this step, the atoms from the previous step are equilibrated using thermal annealing. The thermal annealing process used rRESPA multi-timescale
integrator\textsuperscript{34} with two levels and the pair potential calculations in the second level. The time step for the second level is four femto seconds. The first step of the thermal annealing process was a NVT simulation for 12,500 steps increasing the temperature from 300 K to 600 K. The next step was a NVT simulation for 25,000 steps with a constant temperature of 600 K. The final step was a NVT simulation for 12,500 steps decreasing the temperature from 600 K to 298 K.

3.4.3.3 Bulk Property NVT Calculations

In this step, properties are calculated using NVT MD. The rRESPA conditions and time step from the previous step are used. All simulations are ran at 298 K. The first simulation is ran for 25,000 steps and insures the material has been equilibrated to the simulation temperature. The reported molecular energy is the average over this simulation. Next is a short series of 20 simulations ran for 1,250 steps. This series of simulations was used to calculate the density. Finally, the radius of gyration for each PMMA chain is calculated using LAMMPS during a 12,500 step simulation. The radius of gyration is than averaged over the simulation and each PMMA chain.

3.4.3.4 Young’s Modulus

In this step, the Young’s modulus is calculated. The temperature equilibrated material is the starting point. All simulations used velocity-Verlet integration and a time step of two fs. The first simulation is a NPT simulation for 50,000 steps at a pressure of 0 atm and temperature of 298 K. This simulation ensures the material has been equilibrated to both the temperature and pressure of the system. The next step is to simulate uniaxial strain on the sample by applying 0 atm of pressure on two axis with an NPT simulation and applying a strain rate of $1\times10^8$ on the last axis. This simulation is performed at 298 K for 50,000 steps. The final step continues to apply the 0 atm pressure on two axis but stops straining the sample on the other axis. This step is ran for 50,000 time steps at 298K. These steps lead to a total of 1% strain in the sample. The
Young’s modulus is the average of stress divided by strain in the last simulation step.

3.4.3.5 Glass Transition Temperature

In this step, the glass transition temperature is calculated. The pressure and temperature equilibrated material is the starting point. All simulations used velocity-Verlet integration and a time step of two femto seconds. The first step was running a NPT simulation with a starting temperature of 298 K and an ending temperature of 350 K. The temperature of the simulation steadily increased over 50,000 steps. The next step was running a series of NPT simulations with 5 K increments from 350 K to 400 K. Each of these simulations was 25,000 steps. The average volume from each of these simulations was calculated. Two lines, an upper temperature and lower temperature, were fitted to the average volume data. The temperature at which the lines intersected is the glass transition temperature.

3.4.3.6 Density

The density is calculated using spherical shells starting at the origin of the box and ending at the box boundary. The mass in each shell is summed up and divided by the volume of the shell. The density of each shell and its initial distance are written to a data file and thus a relationship between density and length in the material can be examined. The density in this work is reported in amu/Å³.

3.5 PMMA-Alumina Nano Composite

3.5.1 Potentials

3.5.1.1 Alumina

The potential describing the alumina nano particle came from Matsui35. The potential equation is
\[ U(r_{ij}) = \frac{q_i q_j}{r_{ij}} - \frac{C_i C_j}{r_{ij}^6} + 4.184 \text{KJ/mo}l \left( B_i + B_j \right) e^{\frac{A_i + A_j - r_{ij}}{B_i + B_j}}, \tag{3.19} \]

where \( U \) is the potential energy, \( r \) is the separation distance, \( C, B \) and \( A \) are fitted parameters, and \( q \) is the effective charge which is also fitted\(^\text{35}\). For alumina these parameters are shown in Table 3.9.

Table 3.9: Potential parameters for alumina\(^\text{35}\).

<table>
<thead>
<tr>
<th>Element</th>
<th>( q )</th>
<th>( A(\text{Å}) )</th>
<th>( B(\text{Å}) )</th>
<th>( C(\text{Å}^6 \text{kJ/mol}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al</td>
<td>1.4175</td>
<td>0.7852</td>
<td>0.034</td>
<td>36.82</td>
</tr>
<tr>
<td>O</td>
<td>-0.945</td>
<td>1.8215</td>
<td>0.138</td>
<td>90.61</td>
</tr>
</tbody>
</table>

In LAMMPS, this potential\(^\text{35}\) can be represented using a Born-Mayer-Huggins potential in the form

\[ U = A e^{\frac{\sigma - r}{\rho}} - \frac{C}{r^6} + \frac{D}{r^8} \tag{3.20} \]

and a Coulomb’s potential in the form

\[ U = \frac{q_i q_j}{r}. \tag{3.21} \]

The parameters for the Born Mayer Huggins potentials for the alumina are listed in Table 3.10.

Table 3.10: Parameters for the Born-Mayer-Huggins potential for alumina in LAMMPS.

<table>
<thead>
<tr>
<th>Elements</th>
<th>( A(\text{kcal/mol}) )</th>
<th>( \sigma(\text{Å}) )</th>
<th>( \rho(\text{Å}) )</th>
<th>( C(\text{kcal/mol}^*\text{Å}^6) )</th>
<th>( D(\text{kcal/mol}^*\text{Å}^8) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al-Al</td>
<td>0.068</td>
<td>1.57</td>
<td>0.068</td>
<td>324.0230</td>
<td>0.0</td>
</tr>
<tr>
<td>O-O</td>
<td>0.276</td>
<td>3.643</td>
<td>0.276</td>
<td>1962.2782</td>
<td>0.0</td>
</tr>
<tr>
<td>Al-O</td>
<td>0.172</td>
<td>2.607</td>
<td>0.172</td>
<td>797.3853</td>
<td>0.0</td>
</tr>
</tbody>
</table>
3.5.1.2 PMMA-Alumina

The potential describing PMMA-alumina is a slightly modified version of Shaffer and Chakraborty\(^{36}\). This paper was designed for PMMA-aluminum and consisted of a binding and non binding potentials\(^{36}\). The binding potential,

\[
U = A_i e^{-2\lambda(z_i-z_b)} - B_i e^{-\lambda(z_i-z_b)} + C_i \lambda^2 (z_i-z_b)^2 e^{-\lambda(z_i-z_b)},
\]  

(3.22)

describes the interaction between the carbonyl and ester oxygen of PMMA and aluminum. The B and C parameters for the binding potential of the carbonyl interaction above are equal to

\[
B = b_0 + b_1 \cos \theta + b_2 \cos^2 \theta
\]

(3.23)

\[
C = c_0 + c_1 \cos \theta + c_2 \cos^2 \theta\]

(3.24)

To simplify Equation 3.23 and 3.24, B and C were averaged over all possible values of \(\theta\). The non binding potential,

\[
U = \frac{2}{3} \pi \sigma_i \varepsilon_i \rho \left[ \frac{2}{15} \left( \frac{\sigma_i}{z_i} \right)^9 - \left( \frac{\sigma_i}{z_i} \right)^3 + \frac{2}{3} \sqrt[3]{\frac{5}{2}} \right],
\]  

(3.25)

describes the interaction between the carbons in PMMA and aluminum. There were no potential interactions with hydrogen in PMMA, since Shaffer and Chakraborty\(^{36}\) were written for united atom PMMA. Shaffer and Chakraborty\(^{36}\) was expanded to alumina by assuming the interaction of the oxygen atoms in the alumina with PMMA could be described as the average non binding potential.

Table 3.11: Binding potential parameters describing the interaction between aluminum and the oxygen in PMMA.

<table>
<thead>
<tr>
<th>Elements</th>
<th>A (kcal/mol)</th>
<th>B (kcal/mol)</th>
<th>C (kcal/mol)</th>
<th>(\lambda) (1/Å)</th>
<th>(z_b) (Å)</th>
</tr>
</thead>
<tbody>
<tr>
<td>O (carbonyl)</td>
<td>11.95</td>
<td>34.95</td>
<td>21.36</td>
<td>1.1</td>
<td>2.0</td>
</tr>
<tr>
<td>O (ester)</td>
<td>11.95</td>
<td>37.52</td>
<td>20.79</td>
<td>1.1</td>
<td>2.0</td>
</tr>
</tbody>
</table>

58
Table 3.12: Non binding potential parameters describing the interaction between aluminum and carbon in PMMA and the interaction between oxygen in alumina and the atoms in PMMA.

<table>
<thead>
<tr>
<th>Elements</th>
<th>ρ(Å⁻³)</th>
<th>σ(Å)</th>
<th>ε(kcal/mol)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al-C</td>
<td>0.0603</td>
<td>2.88</td>
<td>1.74</td>
</tr>
<tr>
<td>Al-C(carbonyl)</td>
<td>0.0603</td>
<td>2.88</td>
<td>2.18</td>
</tr>
<tr>
<td>Al-C(CH₂ or CH₃)</td>
<td>0.0603</td>
<td>3.24</td>
<td>1.28</td>
</tr>
<tr>
<td>O-PMMA</td>
<td>0.0603</td>
<td>3.00</td>
<td>1.74</td>
</tr>
</tbody>
</table>

The work of Drabold et al. shows the oxygen atoms in alumina have very weak interactions to PMMA which are similar to the aluminum atoms interactions with the carbon and hydrogen in PMMA. Tables 3.11 and 3.12 show the binding and non binding parameters used to describe the interactions between PMMA and alumina.

### 3.5.2 Compositions and Bonding

In the computational simulations of PMMA alumina nano composite, there are two variables that will be controlled. The two variables are composition and bonding. The composition is controlled by the filler weight percent. The filler is a 3 nm spherical nano particle. The weight of the polymer rather than the nano particle controls the filler weight percent. The filler percentages used are zero, five, ten, and fifteen weight percent. For the zero weight percent composition no nano particle is present. There are four bonding conditions experimented with. They are no bonding, cross linking, weak bonding, and strong bonding. The experimental combinations of bonding and composition are shown below in Table 3.13.
Table 3.13: Computational experimental design of PMMA-alumina nano composite where the light grey squares are planned computational simulations and the white squares are inaccessible.

<table>
<thead>
<tr>
<th>Bonding</th>
<th>Filler %</th>
<th>0 wt%</th>
<th>5 wt%</th>
<th>10 wt%</th>
<th>15 wt%</th>
</tr>
</thead>
<tbody>
<tr>
<td>no bonding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cross linking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>weak bonding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>strong bonding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The four bonding conditions—no bonding, cross linking, weak bonding, and strong bonding—relate to how much the PMMA chains react with the alumina nano particle. The no bonding condition is when the PMMA chains do not bond at all with the nano particle. For the cross linking condition, the PMMA chains are allowed to form two bonds per chain maximum. The weak bonding condition allows four bonds per chain maximum to form. For the strong bonding condition, the PMMA chains are allowed to form eight bonds per chain maximum. These bonding conditions were accomplished using the methodology discussed in section 3.3.

3.5.3 MD Simulations and Property Calculations

The molecular weight for these calculations is 10,000 g/mol. Therefore, the properties of the pure material needed to be recalculated using the methods described in 3.4. These properties were molecular energy, density and Young’s modulus. Three RNG seeds, 3, 7, and 14, were used by the random walk algorithm to build three separate simulation boxes for each box size. Five different box sizes were used. They are, in angstroms, 63.74, 74.94, 86.51, 96.77, 132.58. These properties of the pure PMMA were used to compare with the same properties of the PMMA alumina nano composite.
3.5.3.1 Initial Equilibration of PMMA Around Nano Particle Space

In this step, the random walk algorithm discussed in 3.4 was used to initialize polymer chains around the space where a nano particle will later be inserted. This initialization process was done for three RNG seeds, 3, 7, and 14. The initialized chains separation distance was increased using the NVE/limit algorithm in LAMMPS. This algorithm modifies the NVE integration so atoms can only move a specified maximum distance. The NVE/limit algorithm was ran for 10,000 steps with a time step of one fs. To keep the chains from traveling into the empty space where the nano particle will be inserted, a wall potential was placed around the space. The wall potential was a Leonard Jones potential with parameters derived from the average Leonard Jones parameters of Okada et al.’s PMMA potential. Those Parameters are listed below in Table 3.14.

Table 3.14: Wall Potential Parameters.

<table>
<thead>
<tr>
<th>$\varepsilon$ (kcal/mol)</th>
<th>$\sigma$ (Å)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.11749</td>
<td>4.00078</td>
</tr>
</tbody>
</table>

3.5.3.2 Final Equilibration of PMMA Around Nano Particle Space

In this step the atoms from the previous step are equilibrated using thermal annealing. The thermal annealing process used rRESPA multi-timescale integrator with two levels and the pair potential calculations in the second level. The time step for the second level is four femto seconds. The first step of the thermal annealing process was a NVT simulation for 12,500 steps increasing the temperature from 300 K to 600 K. The next step was a NVT simulation for 25,000 steps with a constant temperature of 600 K. The final step was a NVT simulation for 12,500 steps decreasing the temperature from 600 K to 298 K. During this thermal annealing process, the wall potential in Table 3.14 was used to keep the polymer from overlapping the nano particle space.
3.5.3.3 Preparing PMMA for Computational Bonding

The alumina nano particle was inserted into the empty space in the simulation box where the wall potential was used to keep the PMMA polymer chains out. The wall potential moved the polymer chains away from the nano particle. Removal of the wall potential along with the NVE/limit algorithm in LAMMPS allowed the PMMA polymer chains to slowly move back towards the nano particle. This process was simulated for 5000 steps with a time step of one femtosecond. The restart file from the finished simulation was then converted to a data file which is compatible with the bonding algorithm.

3.5.3.4 Bonding PMMA to the Alumina Nano Particle

In this step, the four bonding conditions were applied to the nano composite. Each bonding condition produced a separate material for later simulation. To produce the bonding conditions, the reaction and bonding process in 3.3 were followed. The final reacted composites were then placed in new data files for later simulations.

In Appendix F, the python scripts used to drive the LAMMPS simulations after the bonding process have the descriptor bonded and non-bonded. In most cases, the only difference between the two is the presence of an additional atom type to show where the ester oxygens have bonded to the alumina nano particle.

3.5.3.5 Initial Equilibration of the Nano Composite

After the bonding process, the polymer chains need to be equilibrated again. In order to get rid of any instabilities due to polymer chains being too close, the NVE/limit algorithm in LAMMPS was run for 5,000 steps with a time step of one fs.

3.5.3.6 Final Equilibration of the Nano Composite

In this step, the atoms from the previous step are equilibrated using thermal annealing. The thermal annealing process used rRESPA multi-timescale integrator\textsuperscript{34} with two levels and the pair potential calculations in the second level.
The time step for the second level is four femto seconds. The first step of the thermal annealing process was a NVT simulation for 12,500 steps, increasing the temperature from 300 K to 600 K. The next step was a NVT simulation for 25,000 steps with a constant temperature of 600 K. The final step was a NVT simulation for 12,500 steps decreasing the temperature from 600 K to 298 K.

3.5.3.7 Bulk Property NVT Calculations

In this step, properties are calculated using NVT MD. The rRESPA conditions and time step from the previous step are used. All simulations are ran at 298 K. The first simulation is ran for 25,000 steps and insures the material has been equilibrated to the simulation temperature. The reported molecular energy is the average over this simulation. Next is a short series of 20 simulations ran for 1,250 steps. This series of simulations was used to calculate the density.

3.5.3.8 Non-Bonded Young’s Modulus

In this step, the Young’s modulus for the non bonded composite is calculated. The temperature equilibrated material is the starting point. All simulations used velocity-Verlet integration and a time step of two femto seconds. The first simulation is a NVT simulation for 50,000 steps at a temperature of 298 K. The second simulation is a NPT simulation for 50,000 steps at a pressure of 0 atm and temperature of 298 K. This simulations insures the material has been equilibrated to both the temperature and pressure of the system. The next step is to simulate uniaxial strain on the sample by applying 0 atm of pressure on two axis with an NPT simulation and applying a strain rate of $1 \times 10^8$ on the last axis. This simulation is done at 298 K for 50,000 steps. The final step continues to apply the 0 atm pressure on two axis but stops straining the sample on the other axis. This step is ran for 50,000 time steps at 298K. These steps lead to a total of 1% strain in the sample. The Young’s modulus was determined in the last simulation step.
3.5.3.9 Bonded Young's Modulus

In this step, the Young’s modulus for the bonded composite is calculated. The temperature equilibrated material is the starting point. All simulations used velocity-Verlet integration and a time step of half a fs. The first step uses a NPT simulation for the polymer at a temperature of 298 K and a pressure of 0 atm. The alumina nano particle is treated as a rigid particle and kept at a constant temperature of 298 K. This first step is ran for 50,000 steps. The next step is to simulate uniaxial strain on the sample by applying 0 atm of pressure on two axis with an NPT simulation of the polymer and applying a strain rate of $1 \times 10^8$ on the last axis. The alumina nano particle continues to be treated as a rigid particle. Both materials are kept at 298 K during the 200,000 steps of the simulation. The final step continues to apply the 0 atm pressure on two axis but stops straining the sample on the other axis. This step is ran for 50,000 time steps at 298K. These steps lead to a total of 1% strain in the sample. The Young’s modulus was determined in the last simulation step.

3.5.3.10 Density

The density is calculated using spherical shells starting at the origin of the box and ending at the box boundary. The mass in each shell is summed up and divided by the volume of the shell. The density of each shell and its initial distance are written to a data file and thus a relationship between density and length in the material can be examined. The density in this work is reported in amu/Å³.

3.6 References


4.1 Lattice Relationship

In this section the lattice relationship, which is the lattice constant’s behavior with temperature, is examined. This relationship is important for examining the accuracy of the fitted potential.

Figure 4.1: Lattice constant temperature relationships of different nickel potentials and experimental/theoretical derived equations. The experimental/theoretical derived equations are “lattice relationship 1” (Equation 3.4) and “cryogenics June 1977” (Equation 3.6). The nickel potentials are “Ni_u3.eam”, “Ni_smf7.eam”, “Ni99.eam.alloy”, “NiAl.eam.alloy”, “yuNi.eam.alloy”, and “Kraus_ni” (potential developed in this work).
Figure 4.1 shows the lattice constant temperature relationship of different nickel potentials compared to experimental/theoretical derived equations. The solid line is Equation 3.4 for Ni, and the dashed line is Equation 3.6 for Ni. Kraus_ni (diamonds) approximately equals the derived equations. The slope of NiAl.eam.alloy (hexagons) is approximately equivalent to Kraus_ni. Mishin (NiAl.eam.alloy) corroborates this evidence by showing the nickel potential at least for temperatures under 300 K follow the linear expansion curve. Although, the 0 K lattice constant is different for yuNi.eam.alloy (squares), the potential produces similar linear expansion to Ni_u3.eam (circles), Ni_smf7.eam (triangles), and Ni99.eam.alloy (pentagons). Figure 4.1 proves using the fitting-testing approach in the methodologies section can yield similar results to that of the force-matching method with the added benefit the metal’s electronic structure will not effect the efficacy of potential fitting.

4.2 Young’s Modulus

In this section the Young’s modulus is examined. This property is also important for testing the accuracy of the fitted potential.

Figure 4.2 shows the effect of temperature on the Young’s Modulus of different potentials and the modulus calculated from experimental elastic constants. The solid line is the calculated modulus of which NiAl.eam.alloy (hexagons) is the closest approximation. Kraus_ni (diamonds) at first glance seems to have the largest error. But with further examination, the error for the first half of the data is roughly equivalent to the error in Ni_u3.eam (circles) and Ni_smf7.eam (triangles). The error for the second half of the data is approximately equivalent to yuNi.eam.alloy (squares). This means from the perspective of absolute error Kraus_ni performs as well as Ni_u3.eam, Ni_smf7.eam and yuNi.eam.alloy.
Figure 4.2: The Young’s modulus of different nickel potentials and those calculated from elastic constants for various temperatures. The calculated modulus is “Young’s Modulus Ni” (Equation 3.8). The nickel potentials are “Ni_u3.eam”, “Ni_smf7.eam”, “Ni99.eam.alloy”, “NiAl.eam.alloy”, “yuNi.eam.alloy”, and “Kraus_ni” (potential developed in this work).

The correct slope of the potential is most important because even if the potential is not producing the accurate values, the potential produces the physical phenomena correctly. From this perspective, the potential with the worst performance, even though the error is low, is Ni99.eam.alloy because the data are not even close to matching the slope of the experimentally calculated curve. The data shows a sudden increase and then decrease in modulus at low temperature (0-100 K). Afterward, the modulus continues to increase with increasing temperature. These slope issues can also be seen in Kraus_ni at higher temperatures (150-300 K) where sudden increases and decreases in the modulus can be observed. Interestingly the potential design of
Kraus_ni is based on Ni99.eam.alloy and the slope problems with Ni99.eam.alloy seem to have carried over as well; though, the magnitude of the problems have been diminished. The other potentials in Figure 4.2 do not appear to have any issues with slope; only the spline based potentials (Ni99.eam.alloy and Kraus_ni) are having this issue.

The slope problems could be due to the uniaxial tensile test losing accuracy due to the interactions with the spline based potentials. During the uniaxial test, the stress is calculated in the direction of strain and controlled in the two perpendicular directions to be zero. The stress is controlled by varying the length in those directions. Stress in LAMMPS is defined as

\[ p_{ij} = \frac{1}{V} \sum_k m_k v_{ki} v_{kj} + \sum_k r_{ki} f_{kj}, \]  

where \( p_{ij} \) is the stress tensor, \( m_k \) is the mass, \( v_k \) is the velocity, \( V \) is the volume, \( r_{ki} \) is the distance between atoms on the i index, and \( f_{kj} \) is the force between two atoms on the j index. The force for the EAM is calculated using

\[ f = -\frac{dE}{dr} = -0.5 \sum_j \frac{d\Phi(r_j)}{dr_j} - \frac{dF}{dr} \sum_j \frac{d\rho(r_j)}{dr_j}, \]

where \( f \) is the force between two atoms, \( E \) is the energy, \( r \) is the distance between atoms, \( F \) is the embedding energy, \( \rho \) is the electron density, and \( \Phi \) is the pair potential. If the derivatives of the embedding energy, electron density, or pair potential have changes from positive to negative, such as can happen between two troughs, the stress can suddenly change when atoms go from the positive derivative to the negative derivative. During a tensile test, this transition can occur due to the strain and or thermal velocity. These transitions would effect the calculated stress and the ability to control the stress in the perpendicular directions to the strain.

If derivative transitions are causing the shape problems to occur, the embedding energy, electron density or pair potential functions of Kraus_ni and Ni99.eam.alloy should have transitions while the other examined potentials will not. These functions are shown in Figures 4.3, 4.4, and 4.5. The embedding energy functions in Figure 4.5 do not show any changes in derivative for the
spline based potentials. Figure 4.3 shows both Kraus_ni and Ni99.eam.alloy have multiple troughs (note the Ni99.eam.alloy).

![Graph showing EAM pair potentials for nickel](image)

Figure 4.3: EAM pair potentials for nickel. The nickel potentials are “Ni_u3.eam”\(^2\), “Ni_smf7.eam”\(^3\), “Ni99.eam.alloy”\(^4\), “NiAl.eam.alloy”\(^5\), “yuNi.eam.alloy”\(^6\), and “Kraus_ni” (potential developed in this work).

extra troughs are not visible on the graph but are in their paper). The local minimums in the troughs attempt to keep the atoms in those troughs, but the local maximum between troughs can allow the atoms through thermal variations and/or strain to go into the nearby trough. This behavior could lead to the observation in the Young’s modulus, but in this case probably does not, because NiAl.eam.alloy also has multiple troughs and was the closest approximation to experiment. Figure 4.4 also shows both Kraus_ni and Ni99.eam.alloy have multiple troughs. Additionally, though not shown on the figure due to scaling issues Ni_u3.eam and Ni_smf7.eam also have multiple troughs with a
maximum for both at approximately (.5,2.5). Though in the case of Ni_u3.eam and

Figure 4.4: EAM electron density potentials for nickel. The nickel potentials are “Ni_u3.eam”\textsuperscript{2}, “Ni_smf7.eam”\textsuperscript{3}, “Ni99.eam.alloy”\textsuperscript{4}, “NiAl.eam.alloy”\textsuperscript{5}, “yuNi.eam.alloy”\textsuperscript{6}, and “Kraus_ni” (potential developed in this work).

Ni_smf7.eam, the atomic structure during a tensile test should never reach such a small separation distance since the equilibrium nearest neighbor distance at room temperature is 2.4917 Å\textsuperscript{8}. The local maximum for both Kraus_ni and Ni99.eam.alloy is located approximately at 2.5 Å which is close enough to the room temperature nearest neighbor value to cause atoms to transition from one trough to another during uniaxial tensile testing.
Figure 4.5: EAM embedding energy potentials for nickel. The nickel potentials are “Ni_u3.eam”², “Ni_smf7.eam”³, “Ni99.eam.alloy”⁴, “NiAl.eam.alloy”⁵, “yuNi.eam.alloy”⁶, and “Kraus_ni” (potential developed in this work).

Table 4.1 shows the elastic constants at 0 K for the potentials shown in Figure 4.2 and their calculated Young’s modulus using Equation 3.8. The table shows the Young’s modulus values in Figure 4.2 at least near 0 K have been accurately calculated.

Table 4.1: Elastic constants at 0K for different potentials and their calculated Young's modulus using Equation 3.8.

<table>
<thead>
<tr>
<th>Potential</th>
<th>C11 (GPa)</th>
<th>C12 (GPa)</th>
<th>Young's Modulus (GPa)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kraus_ni</td>
<td>283.4</td>
<td>143.7</td>
<td>186.6</td>
</tr>
<tr>
<td>yuNi.eam.alloy⁶</td>
<td>244.3</td>
<td>178.3</td>
<td>93.8</td>
</tr>
<tr>
<td>NiAl.eam.alloy⁵</td>
<td>241.3</td>
<td>150.8</td>
<td>125.3</td>
</tr>
<tr>
<td>Potential</td>
<td>C11 (GPa)</td>
<td>C12 (GPa)</td>
<td>Young’s Modulus (GPa)</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------</td>
<td>-----------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Ni99.eam.alloy(^4)</td>
<td>247.0</td>
<td>148.0</td>
<td>136.0</td>
</tr>
<tr>
<td>Ni_smf7.eam(^3)</td>
<td>not listed</td>
<td>not listed</td>
<td>not listed</td>
</tr>
<tr>
<td>Ni_u3.eam(^2)</td>
<td>233.0</td>
<td>154.0</td>
<td>110.4</td>
</tr>
</tbody>
</table>

### 4.3 Spline Fitting

In this section the accuracy of the spline fitting method will be analyzed. The analysis will be done by using the sum of residuals squared for both the fitting properties and the testing properties.

Figure 4.6 shows the normalized sum of residuals squared for the testing properties and how they relate to the number of knots. The relationship matches the explanation given in Mishin et al.\(^4\), who showed the sum of residuals squared for the testing properties decreases with increasing number of knots until the optimal number of knots is reached at which point the sum of residuals squared begins to increase again. The portion of the graph where sum of residuals squared is decreasing with increasing knots corresponds to a potential that is under fit. The other portion of the graph, where the sum of residuals is increasing, corresponds with a potential that is over fit, and the optimum lies between the two portions.

The results shown in Figure 4.6 relate to the lattice relationship due to the properties used for testing. The fact, Figure 4.6 is producing the expected results explains why the lattice constant calculations for this potential are fairly accurate within the temperature range shown. This result also suggests testing different properties within the fitting-testing framework would yield a potential with different calculated lattice constants.
Figure 4.6: Sum of residuals squared for the testing properties relative to the number of knots used for fitting.

Figure 4.7 shows the normalized sum of residuals squared for the fitting properties and how they relate to the number of knots. This relationship does not match the explanation given in Mishin et al.\textsuperscript{4}, who showed the sum of residuals squared for the fitting properties decreases with increasing number of knots. Instead, the figure shows absolutely no relationship between sum of residuals squared and number of knots.
The results in Figure 4.7 relates to both the Young’s modulus and the elastic constants due to the properties used for fitting. These results explain why both the Young’s modulus and the elastic constants are not very close to their objective property values. More importantly, the values of the properties according to the potential fit by Mishin et al.\textsuperscript{4} are fairly close to their objective property values.

Based on the previous results (Figures 4.2, 4.4, and 4.7), there are three possibilities as to why this issue is occurring. The first is that the optimization algorithm is incapable of fitting the properties when the initial potential is too far away. This is justified based on the lack of fit in most of the data points shown in Figure 4.7. The second is that the method to move the knots horizontally allowing the cutoff radius to be optimized causes the optimization algorithm to get stuck in local minima. These horizontal shifts can cause troughs in the potential to shift or

Figure 4.7: Sum of residuals squared for the fitting properties relative to the number of knots used for fitting.
new ones to form which can cause properties to change quite drastically as was shown from Figure 4.2 with the Young's modulus. The third is that the extra throughs in the potential from using splines causes the optimization algorithm to fail to produce potentials which calculate the elastic constants near their objective values in Table 3.2. Figures 4.2 and 4.4 combined with the data in Table 3.2 show enough evidence to back up this possible reason. Unfortunately, the exact reason for the issue is not known and Mishin et al. did not discuss the efficacy of his fitting algorithm from far away or the method used to move the knots horizontally with shifting cutoff radius.

4.4 References


5.1 Splitting Process

The bonding algorithm was used to implement the reaction shown in Table 3.6 by first splitting the PMMA-Alumina nano composite into its separate components. The splitting process, steps 2, 3, and 4 of the methodology, is shown in Figure 5.1. The images in the figure were made using Visual Molecular Dynamics (VMD)\(^1\).

The composite in Figure 5.1 (topmost image) consists of 4 chains of 100 monomer units and a spherical alumina nano particle with 510 atoms. The alumina nano particle’s [0 0 1] and [0 1 0] direction matches the [0 0 1] and [0 1 0] direction of the simulation box. The structure of the alumina used comes from Mincryst\(^2\), a crystallographic database from the Institute of Experimental Mineralogy of the Russian Academy of Sciences. By weight, alumina is approximately twenty percent of the composite.

The splitting process allows the composite to be split into its separate components. The first five images in Figure 5.1 below the composite are the four PMMA chains and the alumina nano particle. The alumina nano particle can be further split into the nano particle surface, bottom-most image in Figure 5.1. This surface is composed only of oxygen atoms from the nano particle because only those atoms react with the polymer chains.
Figure 5.1: Splitting PMMA-alumina nano composite into separate components.
5.2 Bonding Process

The nano particle surface is shown interacting with the four polymer chains in Figure 5.2. Chemically, these interactions allow bonds to form between the two components. Computationally, the possible bonds between the components require locating before they can be bonded. This step requires the atom(s) on the polymer chain where bonds will form and the size of the bonding region around the nano particle surface atom. For the PMMA-Alumina system, the bond to the nano particle surface comes from the ester oxygen on the PMMA, and ideally, the maximum distance for the bonding region should be set to 2.8621 Å corresponding to the maximum oxygen-oxygen distance in the alumina. This distance may need increasing by up to an angstrom to account for ester oxygen atoms which are the ideal maximum distance away from the nano particle but are not radially aligned to the oxygen surface atoms of the nano particle. These two values (ester oxygen and 2.8621) along with the number of bonds are used by the bonding algorithm to give the ester locations on the pmma chain and oxygen atoms on the particle surface where bonds will be formed computationally.

With the bonding locations for each chain known, the PMMA chains can now be bonded to the nano particle surface. Figure 5.3 shows a graphical representation of this bonding process between PMMA, alumina, and water. This figure is graphically equivalent to the chemical reaction in Table 3.6 for the united atom system. The bonding process requires the number of bonding locations for each chain to be a factor of two. Half the bonding process in Figure 5.3 is accomplished on the polymer chain, where part of the replace step and the full delete step will take place. The other half of the bonding process occurs on the nano particle surface where the rest of the replace step and the add to step occurs.
Figure 5.2: The different PMMA polymer chains interacting with the oxygen atoms on the nano particle surface.

The bonding process has important requirements in order for the bonded composite to form correctly. The only requirement is that the replaced and added oxygens have the correct charge. The replaced oxygen has a charge of -0.7825. This is the sum of the charge from the PMMA ester oxygen atom and half the charge from an oxygen atom in alumina. The PMMA ester oxygen atom has a value of -0.31\(^3\). The oxygen atom in alumina has a charge of -0.945\(^4\) so the portion contributed to the replaced oxygen is -0.4725. The added oxygen has a charge of -0.945, which is the charge of an oxygen atom in alumina\(^4\).
With the bonding finished, the modified polymer chains and nano particles are combined to form a bonded composite. The information stored in the bonded composite is written to a data file for future MD simulations.

5.3 References


CHAPTER 6
POLYMER INITIALIZATION ALGORITHM

The polymer initialization algorithm developed in this dissertation is examined for its accuracy in calculating the radius of gyration, Young’s modulus and glass transition temperature using MD simulations. Additionally, the modifications to the PMMA potential are also examined from these calculations. These calculated values and their statistical analysis are listed in Table 6.1. The experimental values in Table 6.1 are within the 95% confidence interval of the calculated values. Therefore, from the examination of Table 6.1, the polymer initialization algorithm successfully models the PMMA properties.

Table 6.1: Experimental values, calculated values and statistical analysis of the radius of gyration, Young’s Modulus, and glass transition temperature.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Calculated</th>
<th>Standard Deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius of Gyration (Å)</td>
<td>35-43¹ at 23,000 g/mol</td>
<td>44.21</td>
<td>9.992</td>
<td>5.056</td>
</tr>
<tr>
<td>Young’s Modulus (Mpa)</td>
<td>2743², 2553-3174³, 1800-3100⁴</td>
<td>2005.4</td>
<td>635.00</td>
<td>321.6</td>
</tr>
<tr>
<td>Glass Transition Temperature (ºC)</td>
<td>85-105³, 105⁴</td>
<td>104.1</td>
<td>8.76</td>
<td>4.46</td>
</tr>
</tbody>
</table>

But, Table 6.1 does not give the full information because size effects are not included. Size effects cause material properties to change over a range of box lengths. These changes are not due to statistical variations but due to the box size/length effecting the property calculation. In the following sections, the
radius of gyration, Young’s modulus and glass transition temperature are examined for size effects occurring in the MD simulations. Additionally, the molecular energy and density will also be examined. If any size effects are present, they will be explained in the following sections.

6.1 Radius of Gyration

In this section the size effects on radius of gyration are examined.

![Figure 6.1: Radius of gyration with respect to simulation box length. The error bars are the standard deviation.](image)

Figure 6.1 shows the relationship between the radius of gyration and the simulation box length. In this figure, the standard deviation of each point is shown with an error bar. The error bars show the data points are significantly different. Additionally the figure shows that the radius of gyration for the smallest
simulation box is less than the experimental value and for the two largest simulation boxes is greater than the experimental value.

For the smallest simulation box, the radius of gyration is believed to be limited by the box size because the polymer chains on one side of the periodic boundary can not expand through the boundary because of polymer chains on the other side of the boundary blocking the expansion. Hence the polymer chains are limited in their expansion in the simulation box. For an ideal chain, the radius of gyration can be calculated using

$$
\left\langle R_s^2 \right\rangle = \frac{1}{6} \left\langle R^2 \right\rangle,
$$

(6.1)

where $R$ is the end to end distance and $R_s$ is the radius of gyration. If the box is limiting the radius of the gyration, there are three possible end to end distances that can occur. The first is an end to end distance equal to the box length. The second is an end to end distance equal to the length of the diagonal on the side of the box. The third is an end to end distance equal to the length of the diagonal running through the box. The radius of gyration for the first and second end to end distance is listed in Table 6.2 along with values calculated from MD simulation. Table 6.2 clearly shows the box size is limiting the radius of gyration for the smallest box length.

Table 6.2: Radius of gyration calculated using Equation 6.1 for the end to end distance equal to the simulation box length and the length of the diagonal on the side of the box. and the radius of gyration calculated through MD simulations.

<table>
<thead>
<tr>
<th>Simulation Box Length (Å)</th>
<th>Radius of Gyration for Simulation Box Length (Å)</th>
<th>Radius of Gyration Calculated from MD Simulations (Å)</th>
<th>Radius of Gyration for Length of the Diagonal on the Side of the Box (Å)</th>
</tr>
</thead>
<tbody>
<tr>
<td>63.7</td>
<td>26.0</td>
<td>33.6</td>
<td>36.8</td>
</tr>
<tr>
<td>74.9</td>
<td>30.6</td>
<td>35.7</td>
<td>43.2</td>
</tr>
<tr>
<td>86.5</td>
<td>35.3</td>
<td>42.4</td>
<td>49.9</td>
</tr>
<tr>
<td>Simulation Box Length (Å)</td>
<td>Radius of Gyration for Simulation Box Length (Å)</td>
<td>Radius of Gyration Calculated from MD Simulations (Å)</td>
<td>Radius of Gyration for Length of the Diagonal on the Side of the Box (Å)</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>96.8</td>
<td>39.5</td>
<td>48.1</td>
<td>55.9</td>
</tr>
<tr>
<td>132.6</td>
<td>54.1</td>
<td>61.3</td>
<td>76.6</td>
</tr>
</tbody>
</table>

Additionally, Table 6.2 shows the simulation box length is limiting the radius of gyration for the rest of the data; but, in fact the radius of gyration is not limited at all. The second and third box lengths yield a radius of gyration within the experimental range of the radius of gyration. For the two largest simulation boxes, the radius of gyration is believed to be above the experimental radius of gyration due to relaxation times. According to Lewis et al., polymers exhibit two ranges of relaxation times slow and fast\textsuperscript{6}. MD simulations can only capture the fast relaxation times due to the simulation times usually being limited to the nano second time scale which explains why the polymer in the two largest boxes are unable to change their conformations to fit with the experimental data in Table 6.1.

### 6.2 Molecular Energy

The molecular energy as shown in Figure 6.2 decreases with the simulation box length. The molecular energy is the energy of the bonds, angles, dihedrals and impropers of the molecule. The molecular energy once the box became big enough was expected to be invariant with box length. Based on the previous discussion of the radius of gyration, the molecular energy should become invariant past the 86.5 Å simulation, instead the energy continues to decrease. This decrease could be caused by the inability for the polymer to relax into its bulk confirmation as discussed in the radius of gyration section above. The other possibility for the continued decrease is that the box size at which energy invariance occurs has not been reached yet. In this case, the decreasing
molecular energy could be caused by a nano scale effect due to lack of sufficient confirmations visited.

![Figure 6.2: Molecular energy with respect to simulation box length. The error bars are the standard deviation.](image)

**6.3 Young’s Modulus**

The Young’s modulus in Figure 6.3 shows no size effects in the values but does in the standard deviation. The standard deviation shows a large decrease for the first three box lengths, and then, the standard deviation changes very little. The change in the first two box lengths are most likely from the simulation box being too small and therefore restricting confirmations. This effect is also seen in the radius of gyration but only for the first box length. The third box length is big enough that conformations are no longer restricted, and therefore, all box lengths after show very little change in the standard deviation.
The density in Figure 6.4 also shows no size effects in the values but does in the standard deviation. The standard deviation for the first three box lengths is much larger than the last two box lengths. Examining the standard deviations in Figures 6.4 and 6.3, there seems to be a relationship between the density and the Young’s modulus. The first two box lengths in Figures 6.4 and 6.3 have higher standard deviations than the other box lengths. Even though the third box length in Figure 6.4 has a higher standard deviation than other box lengths, the standard deviation in Figure 6.3 is the second smallest. For the two largest box lengths the standard deviations in Figures 6.4 and 6.3 are smaller than other box lengths.
Figure 6.4: Density with respect to simulation box length. The error bars are the standard deviation.

6.5 Glass Transition Temperature

The glass transition temperature data in Figure 6.5 show no size effects in the values or standard deviations, but the standard deviations are approximately multiples of ten. The factor of ten for the standard deviations occurs because the simulations for glass transition temperature are done with 5 °C temperature increments. This temperature increment leads to the error approximately being in increments of 5 °C in either direction which translates to approximately increments of 10 °C for the standard deviation.
Figure 6.5: Glass transition temperature with respect to simulation box length. The error bars are the standard deviation.

6.6 References


CHAPTER 7
PMMA-ALUMINA NANO COMPOSITE

7.1 PMMA Properties

The results for chapter 6 showed that the standard deviation for the Young’s Modulus and the density of PMMA were related, but in order to examine if these properties are related in the nano composite, any relationship with simulation box length needs to be removed from PMMA.

Figure 7.1: Young’s modulus of 10,000 g/mol PMMA with respect to box length. The error bars are the standard deviation.
The changes in standard deviation for the Young’s modulus in chapter 6 were thought to have been from the box size being too small for a 22,000 g/mol PMMA chain. So, in this chapter the PMMA chain was reduced to 10,000 g/mol, and the standard deviation for the Young’s modulus as shown in Figure 7.1 became fairly consistent. Additionally, from examining Figures 7.1 and 7.2 the relationship between the standard deviations in the Young’s modulus and the density has disappeared. These figures also show no relationship with box length for either of these properties.

![Figure 7.2](image)

**Figure 7.2**: Density of 10,000 g/mol PMMA with respect to box length. The error bars are the standard deviation.

Because all box length relationships for the Young’s modulus and the density have been removed, the average value of these properties can be used as a benchmark of PMMA with a 10,000 g/mol chain. These property values are listed in Table 7.1 along with the standard deviation and 95% confidence interval.
of these properties. The average value and statistical analysis for the Young’s modulus are similar to those found in chapter 6. But table 7.1 only shows the box length independent properties, the box length dependent properties of PMMA are discussed next.

Table 7.1: The calculated values and statistical analysis of the box length independent properties of pure PMMA.

<table>
<thead>
<tr>
<th>Property</th>
<th>Calculated</th>
<th>Standard Deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Young’s Modulus (Mpa)</td>
<td>1907.0</td>
<td>598.61</td>
<td>302.94</td>
</tr>
<tr>
<td>Density (amu/Å³)</td>
<td>0.656</td>
<td>0.00212</td>
<td>0.00107</td>
</tr>
</tbody>
</table>

7.2 Molecular Energy Difference Between PMMA and Composite

The only box length dependent property examined is the molecular energy. This property’s relationship with box length for PMMA is shown in Figure 7.3. Figure 7.3 also shows the unbonded nano composite’s relationship with box length. The figure shows the molecular energy increases for all box lengths except the biggest during the formation of the nano composite. This increase in energy most likely means the polymer chains are modified from their base conformation in order to form the unbonded nano composite.
7.3 Effects of Filler on Composite Properties

In this section the effects of composition on the Young’s modulus, molecular energy and density are examined. The composition is in percent mass of PMMA rather than the alumina filler.
Figure 7.4: Molecular energies of the unbonded and bonded PMMA-alumina nano composite as they change with composition.

Figure 7.4 shows the bonding of the PMMA to the alumina nano composite further deforms the polymer chains adding even more molecular energy to the system. Interestingly, once the bonding process has occurred further bonding does not add a substantial amount of molecular energy to the polymer in the composite. The reason for the initial change in molecular energy is probably due to the first bonds formed causing many changes in the polymer’s conformation, but additional bonds only lead to subtle changes in the molecular energy, because the energy has either reached a maximum or the conformations change very little.
Figure 7.5: Young’s modulus of unbonded, bonded, experimental and theoretical data with respect to composition. The experimental 13 nm data comes from Friederich et al\textsuperscript{1}. The experimental 39 nm data comes from Ash et al\textsuperscript{2}. The equation describing the Guth-Smallwood relationship comes from Joni et al\textsuperscript{3}.

The Young’s modulus calculated from molecular dynamics for the nano composite was expected to have some similarities with experimental data. Instead, Figure 7.5 shows there are no similarities at all with the experimental data. The difference between the two systems is thought to be due from the MD system having a single nano particle but the experimental system having numerous nano particles. Additionally, the MD system’s nano particle has a single bonding state, but the experimental system has numerous bonding states. From the discrepancy, a hypothesis was formed that the experimental system is a superposition of all the bonding states in the material, and the MD calculations show individual states. This hypothesis would lead to the experimental Young’s
modulus being an average of the statistical distribution of bonding states. This hypothesis needs further testing.

Table 7.2: The Young’s modulus (Mpa) of the un-bonded composite by composition and RNG seed values. The values listed by composition are the average of the three seed values. The three seed values are 3, 7, and 14.

<table>
<thead>
<tr>
<th>Composition</th>
<th>85 % PMMA</th>
<th>90 % PMMA</th>
<th>95 % PMMA</th>
<th>98 % PMMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1852.1</td>
<td>3515.3</td>
<td>1701.5</td>
<td>2125.7</td>
</tr>
<tr>
<td>7</td>
<td>3674.1</td>
<td>3462.7</td>
<td>666.7</td>
<td>2582.8</td>
</tr>
<tr>
<td>14</td>
<td>1204.4</td>
<td>1945.0</td>
<td>2716.4</td>
<td>2160.4</td>
</tr>
<tr>
<td>Composition</td>
<td>2243.5</td>
<td>2974.3</td>
<td>1694.9</td>
<td>2289.6</td>
</tr>
</tbody>
</table>

The Guth-Smallwood relationship in Figure 7.5 was calculated using volume fraction data of the composite and the Young’s Modulus of the PMMA. The line on the graph is equal to

\[
\frac{Y_{\text{composite}}}{Y_{\text{polymer}}} = 1 + 2.5V_f + 14.1V_f^2,
\]

(7.1)

where \(Y_{\text{composite}}\) is the Young’s Modulus of the composite and \(Y_{\text{polymer}}\) is the Young’s modulus of the polymer and \(V_f\) is the volume fraction of the nanoparticle. The Young’s modulus of the unbonded composite is believed to follow this trend line since three out of four of the data points are within reasonable range of the line. The composition at 90 % PMMA is not within reasonable range because two of its three RNG seed values give very high Young’s Modulus. The Young’s modulus for each composition along with the average of the seeds can be seen in Table 7.2. The table shows the other three compositions do not have two seed values which give very high (over 3,000 Mpa) or very low (under 1,500 Mpa). Therefore the value of seed 14 for the 90% composition is probably closer to the true average than the other two seeds which give very high values. Seed 14 at 1,945 Mpa is fairly close to the Guth-Smallwood relationship at the 90 % PMMA composition which means all the data points for the unbonded composite probably follow the Guth-Smallwood relationship.
In Ciprari et al. the modulus of PMMA-alumina and other polymer metal oxide nano composites were related to bonding\(^4\). The modulus of the composite relative to the pure polymer was found to relate to the anchors/nm\(^2\) with the following relationship,

\[
Y_{\text{relative}} = \frac{Y_{\text{composite}}}{Y_{\text{polymer}}} = \delta^{1/3},
\]

where \(\delta\) is the anchors/nm\(^2\) and \(Y_{\text{relative}}\) is the relative modulus\(^4\). The relationship starts at approximately 40% relative modulus and increases with the number of anchor points following Equation 7.2\(^4\). These measurements were done with a 5% by volume filler loading\(^4\); this loading corresponds approximately to 98 % PMMA by mass. Because no bonding experiments were done at 98 % PMMA, direct comparison to the data is impossible. But, the data of Ciprari et al\(^4\) does at
least show that large decreases from the modulus of the pure material is possible, which suggests the Young’s modulus of the bonded nano composites at 95 and 90 % PMMA in Figure 7.5 are at least reasonable.

Figure 7.7: The average density of peak two for the unbonded and bonded PMMA-alumina nano composite for different compositions. The density of the nano composite formed three distinct peaks and a bulk region where the density changed very little. The first peak was closest to the nano particle, and the bulk region was farthest from the nano particle. The data shown in Figures 7.6, 7.7, 7.8 and 7.9 are the average values of each peak/region. Of key interest are the first peak and the bulk region because most theories discuss these two distinct areas. The results for peak 1 (Figure 7.6) show the density increases when forming the unbonded nano composite. The density further increases from bonding. Additionally, the density of the unbonded nano composite is fairly similar with composition. This trend is also found with
bonded nano composite. The bulk region (Figure 7.9) shows that the density decreases with decreasing percent of PMMA in the nano composite. Consequently, all density values of the nano composite are below the pure PMMA values. Additionally, the bonded nano composites have a lower density than the unbonded nano composites.

Figure 7.8: The average density of peak three for the unbonded and bonded PMMA-alumina nano composite for different compositions.

These two figures suggest that as the unbonded nano composite forms; material from the bulk region is pulled into peak 1. Bonding causes more material to be transported out of the bulk region and placed in peak 1. Peak 2 and peak 3 (Figures 7.7 and 7.8) are very random in how the density relates to both bonding and composition. These figures suggest these two peaks are by products of material being moved from the bulk region to peak 1. Why these two peaks form instead of having only one peak and a bulk region is not exactly clear.
A molecular dynamics study by Cho and Sun showed that the density around the nano particle increases as the interaction strength between nano particle and polymer matrix increases. This result is similar to what occurs in Figure 7.6 assuming bonding increases the interaction strength. Unfortunately, additional bonding either does not increase the interaction strength or the density directly around the nano particle has reached a maximum and cannot be increased further.

In Friederich et al. and Ciprari et al., the change in density at the interphase is discussed. The density of the interphase depends on the strength of the interaction between the polymer and the nano particle surface. For a weak interaction, the density at the interphase decreases, but for a strong interaction, the density at the interphase increases. Therefore the interphase
calculated from MD shows a strong interaction between the PMMA and alumina nano particle.

Experimentally, the PMMA-alumina interphase was examined using a 200 kV transmission electron microscopy (TEM)\(^1\). The results showed a single interphase region\(^1\) which differs from the three peak interphase region calculated from MD. Each peak in the MD simulations is between 1.5 Å and 3 Å in thickness. These peaks are thin enough that depending on the microscopes minimum resolution, the peaks would either show up as a single peak (as in the experiment) or as multiple peaks (as in the MD simulations).

### 7.4 Effects of Density on Young’s Modulus

The relationship between Young’s modulus and the average density of the three peaks and the bulk region are shown in Figures 7.10, 7.11, 7.12 and 7.13. In each of these figures, the star like point is the pure PMMA, and a nearly horizontal line can be drawn between the points of the pure PMMA and the unbonded composite. Figures 7.10 and 7.13 show a nearly vertical line can be drawn between the points of the bonded composite.
Figure 7.10: The relationship between the peak 1 density and the Young's modulus of the bonded and unbonded composite.

Figures 7.11 and 7.12, on the other hand, do not show any clear relationship between density and Young's modulus for the points of the bonded composite. These figures show for the unbonded material there is little to no relationship between the density and the Young's modulus; but for the bonded material, there is definitely a relationship between these properties even if the relationship is not always well defined.
Cho and Sun examined the stress distribution in their single nano particle composite\(^5\). They found the stress around the nano particle in most cases was larger than the stress in the bulk material. Additionally they related the increase in stress to the increase in density around the nano particle. Their results show that a relationship exists between density and the Young’s modulus\(^5\).

Figure 7.11: The relationship between the peak 2 density and the Young’s modulus of the bonded and unbonded composite.
Figure 7.12: The relationship between the peak 3 density and the Young’s modulus of the bonded and unbonded composite.
7.5 Effects of Molecular Energy on Young’s Modulus

In this section the effects of the molecular energy on Young’s modulus is examined.

Figure 7.14 examines the relationship between the molecular energy and the Young’s modulus. When looking at all the data, there does not seem to be any relationship at all. But realizing, the left most data point of each composition is the unbonded composite, and the other points are the bonded composite; two relationships emerge. The first is the Young’s modulus of the unbonded composite changes very little with increasing molecular energy. In fact, the molecular energy does not seem to effect the unbonded composite at all. This result is very similar to that found between the density and the Young’s modulus
for the unbonded composite. The second relationship is for the bonded composite data points. For these data points, each composition shows a different linear relationship between the points of the bonded composite. This second relationship suggests a strong relationship between the molecular energy and the Young’s modulus. These two relationships are very similar to those found for the density and the Young’s modulus which suggests the molecular energy, density and Young’s modulus are all related.

Figure 7.14: The relationship between Young’s modulus and the molecular energy.

7.6 Generalized Model

The suggested relationship between the molecular energy, the density and the Young’s modulus could be explained if the bonded composite is considered an infinite series of non linear parallel springs. The compression/extension of each spring would than directly be related to the difference in density between
the pure PMMA and the bonded composite in a infinitesimal section of the composite. The Young’s modulus which relates the stress to strain relationship of the spring system can than be defined using Equation 7.3,

\[
Y = \frac{\int_{P_{end}}^{L_{box}} Y_0 + S e^{\alpha(x) - \rho_0} dx}{\int_{P_{end}}^{L_{box}} dx},
\]  

(7.3)

where \( Y_0 \) is the Young’s modulus of the pure PMMA, \( Y \) is the Young’s modulus of the bonded composite, \( x \) is the distance from the center of the material, \( \rho \) is the density of bonded composite at \( x \), \( \rho_0 \) is the density of the pure PMMA, \( S \) and \( \alpha \) are constants defining the stiffness of the spring, \( L_{box} \) is the length of the simulation box and \( P_{end} \) is the end of the nano particle. The molecular energy which relates the work done on the spring can than be defined using Equation 7.4,

\[
E = \frac{\int_{P_{end}}^{L_{box}} E_0(c) + R \beta e^{\beta(x) - \rho_0} dx}{\int_{P_{end}}^{L_{box}} dx},
\]  

(7.4)

where \( E \) is the molecular energy of the bonded composite, \( E_0(c) \) is the molecular energy of the unbonded composite which is composition dependent, \( x \) is the distance from the center of the material, \( \rho \) is the density of bonded composite at \( x \), \( \rho_0 \) is the density of the pure PMMA, \( R \) and \( \beta \) are spring constants, \( L_{box} \) is the length of the simulation box and \( P_{end} \) is the end of the nano particle.

The proposed equations explain the relationship between the Young’s modulus, molecular energy and density. These equations also describe the effects of the changes in the chain configuration that occur during the addition of the nano particle and bonding process in the simulation box. These changes were discussed earlier and are accounted for with the \( S \) and \( R \) parameter. The other parameters, \( \alpha \) and \( \beta \), affect how the newly configured chain expands and contracts with the internal forces of the system. These equations, allow both the
reconfiguration of the polymer chains and the internal forces of the system to be related to molecular energy and Young’s modulus through use of density.

Figure 7.15: Relationship between S and % bonding

Since $\alpha$ and $\beta$ are small, the majority effect on the molecular energy and Young’s modulus is from changes in S and R. Therefore these parameters are examined in more detail. These parameters with respect to percent bonding are shown in Figures 7.15 and 7.16. The S and R values match the changes in the Young’s modulus and molecular energy from the pure material to the composite. These figures shows that chain reconfiguration is the majority factor in how the Young’s modulus and molecular energy behave. Additionally Figure 7.17 shows that S and R are related. This relationship is required because the two equations are describing the same springs. Additionally, the Young’s modulus (Equation 7.3) and molecular energy (Equation 7.4) were also found to be related; so, if no
relationship was found between S and R, then these equations would not be correct.

The first issue with the proposed model is with the signs of the parameters. For $s$ and $\alpha$, the change in modulus with increasing density depends on the sign of the parameters which is problematic because the equation can either match reality or not. The effects of the four possible sign combinations are shown below in Table 7.3. When fitting the $S$ and $\alpha$ parameters, the parameters were required to have the same sign in order to insure the model shows the Young’s modulus increases with increasing density which matches previous discussion in this chapter. $R$ and $\beta$ were required to be positive to insure the molecular energy increased with increasing density.

Table 7.3: How the sign of the parameters in Equation 7.1 affects the change of modulus with increasing density.

<table>
<thead>
<tr>
<th>Sign of S</th>
<th>Sign of $\alpha$</th>
<th>Change in Modulus with Increasing Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>increases</td>
</tr>
<tr>
<td>+</td>
<td>-</td>
<td>decreases</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>increases</td>
</tr>
<tr>
<td>-</td>
<td>+</td>
<td>decreases</td>
</tr>
</tbody>
</table>
The second issue with the proposed model is if a viscous or plastic region forms from the increase or decrease in density from the bulk value. There is evidence in Ash et al. that plastic or viscous forces may affect the PMMA-alumina nano composite’s tensile properties. Ash et al. found that the nano composites strain at failure was nearly 6 times greater than the pure PMMA or the PMMA-alumina micro composite\(^2\). Additionally, the nano composite had a yield point while pure PMMA and micro composite fail before reaching the yield point. These results suggest the nano composite had ductile failure while the micro composite and pure PMMA had brittle failure. Magnification of the broken surface of the nano composite revealed nano particles in small voids with no polymer attached to the particle\(^2\) which suggests the failure was due to polymer pull out. In fact, according to their explanation the nano composite experienced brittle-type fast failure initiating from defects in the material\(^1\). The authors explanation for why the
PMMA-alumina nano composite was able to yield and reach a failure strain six times larger than pure PMMA was from shear yielding. Though the authors suggest such a phenomenon can also occur from stress relief in the material such as might occur if a viscous or plastic region formed in the nano composite.

Table 7.4: Experimental material failure information from Ash et al. The PMMA composites listed below are PMMA-alumina.

<table>
<thead>
<tr>
<th>Material</th>
<th>Strain at Failure (% Strain)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMMA</td>
<td>4.5</td>
</tr>
<tr>
<td>95 wt % PMMA (Micro)</td>
<td>5</td>
</tr>
<tr>
<td>95 wt % PMMA (nano)</td>
<td>29</td>
</tr>
</tbody>
</table>

Figure 7.17: Relationship between S and R.
The model does have one limitation. At a low enough density, the Young’s modulus does not follow Equation 7.3 anymore. This limitation occurs because as the density of the composite nears zero the Young’s modulus also should near zero. But, Equation 7.3 does not show the Young’s modulus nearing zero at zero density. The exact density at which the Young’s modulus deviates from Equation 7.3 is unknown at this time.

Based on the relationship between the strength of the polymer-nano particle interaction and the interphase thickness discussed in Friederich et al. and Ciprari et al., the interphase thickness should decrease with increased bonding. Instead the interphase thickness listed in Table 7.5 has no relationship with the amount of bonding. There are three possible explanations to the lack of relationship. The first is that the near constant interphase thickness is an artifact of the multiple cutoff lengths used in the molecular dynamic simulations. The second is entanglement effects due to polymer migrating from the bulk region to the first peak cause the interphase to consistently form a certain distance away from the nano particle. The third is the relationship itself describes accurately the large difference in interaction effects between different materials but does not capture correctly the small interaction effects caused by bonding.

Table 7.5: Interphase thickness of different compositions and bonding levels for the PMMA-alumina nano composite.

<table>
<thead>
<tr>
<th></th>
<th>85%</th>
<th>90%</th>
<th>95%</th>
<th>98%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unbonded</td>
<td>7.5 Å</td>
<td>7.5 Å</td>
<td>7.5 Å</td>
<td>7.3 Å</td>
</tr>
<tr>
<td>Cross Link</td>
<td>7.3 Å</td>
<td>7.0 Å</td>
<td>7.0 Å</td>
<td></td>
</tr>
<tr>
<td>Weak Bond</td>
<td>7.5 Å</td>
<td>7.0 Å</td>
<td>7.5 Å</td>
<td></td>
</tr>
<tr>
<td>Strong Bond</td>
<td>8.3 Å</td>
<td>7.5 Å</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7.7 References


CHAPTER 8

CONCLUSIONS AND FUTURE WORK

The fitting-testing framework design yields good results for the lattice constant relationship due to the properties calculated in the Testing Properties module. Additionally, the framework is a good replacement for the force-matching method with the added benefit the results are independent of the electronic structure. The splines used in the Potential Creation module were ineffective in being fit to the properties calculated in the Fitting Properties module. This ineffectiveness can be seen in the elastic constants, Young’s modulus, and finally in the sum of residuals squared for the fitting properties. Additionally, the multiple troughs in the electron density function are suspected of causing issues with the Young’s modulus calculation and possibly the calculated elastic constants as well. Due to the ineffectiveness of the fit and suspected issues from the multiple troughs in the electron density function, the use of splines in the Potential Creation module is not useful for a generalized approach such as the fitting-testing framework.

Future exploration of the fitting-testing framework design should focus on utilizing equation based potentials in the Potential Creation Module. These potentials do not create multiple troughs in the electron density function and calculate the correct slope of the curve for the Young’s modulus. Special attention should be given to the equations of Mishin\textsuperscript{16} because they produced the closest approximation to the modulus and the correct thermal expansion. Additionally, a main controller algorithm needs to be produced which allows the fitting-testing to work with an equation based potential.

This new controlled bonding method expands MD simulation’s capabilities and accuracy with polymer-ceramic nano composites. Researchers can now represent the ceramic nano particle as actual atoms rather than a bead and spring model. Thus, the physical interactions can be more accurately
represented during MD simulations and the chemical reactions can be implemented more realistically on the molecular dynamic length scale. To allow bonding reactions between polymer and nano particle to occur computationally, an algorithm was developed. The algorithm allows the bonding chemistry to be translated into a simple step by step procedure which can be executed quickly and easily. The result is a bonded nano composite.

Currently the algorithm only supports ceramic nano particles. With further development and modifications, the algorithm should be capable of supporting polymer and metal nano particles as well.

The polymer initialization algorithm developed for nano composites was found to reproduce the radius of gyration, Young’s modulus and glass transition temperature with in statistical error of their experimental values. Size effects were found for the values of the radius of gyration, and molecular energy. Size effects were also found for the standard deviation of the Young’s modulus and density. A relationship was found between the standard deviations of the Young’s modulus and density.

The size effects of the radius of gyration and molecular energy could use further examination. The radius of gyration for the larger box sizes was believed to be greater than the bulk radius of gyration due to MD only capturing the fast relaxation times. This hypothesis can be tested by running MD simulations for a longer total time such as a few nano seconds. The hypothesis that the size effect of the molecular energy is due to a nano scale effect could be tested further. In this test, very large simulation boxes with a length greater than 100 nm (outside nano region) would have there molecular energy calculated using the same methodology in this dissertation. If these tests show the energy is invariant under these conditions than the hypothesis about nano scale effects is probably correct.

As part of researching PMMA-alumina, the properties of pure PMMA were simulated for use as a baseline comparison to the composite. These new experiments on pure PMMA used a molecular weight of 10,000 g/mol which is much lower than the 23,000 g/mol used in the previous experiment on PMMA.
The lower molecular weight PMMA caused the size effect found for the standard deviation of the Young’s modulus and density to disappear. The relationship between the standard deviation of the Young’s modulus and the density also disappeared. This result was needed in order for the pure PMMA to be used as a baseline comparison with the composite.

The modeling of PMMA-alumina using molecular dynamics found a number of interesting relationships in the density, molecular energy and the Young’s modulus including a model relating all three properties. The Molecular energy of the unbonded composite was found to increase slightly in most cases from pure PMMA. Bonding of the composite caused a much large increase in the molecular energy but further bonding caused little to no change. The Young’s modulus of the unbonded composite was found to follow the Guth-Smallwood relationship, but the bonded composite did not follow either the experimental data or the Guth-Smallwood relationship. From this data a hypothesis was formed that the experimental data was an average of the statistical distribution of bonded states in the composite. The density was found to form three peaks and a bulk region. When forming an unbonded composite, the density in the first peak was found to consistently increase and consistently decrease in the bulk region. Bonding of the composite caused further increase in the first peak and further decrease in the bulk region. The second and third peak did not show any consistent trend. The explanation for the density data is that forming the composite causes polymeric material from the bulk region to migrate to peak 1. Peak 2 and 3 are by products of this migration and therefore no consistent trend would be found. Density and molecular energy were found to have little effect on the Young’s modulus for the unbonded composite. But for the bonded composite, both density and molecular energy had a strong effect on the Young’s modulus. A model was formed for the bonded composite using an infinite series of non linear parallel springs. This model related the compression/extension of the springs to the change in density from the composite to the pure PMMA. The change in compression/extension was than related to the Young’s modulus and molecular energy through a linear integral model. The model describes two steps.
that occur during the formation of a nano composite. The first is the change in chain configuration and the second is the expansion and contraction of the chain to balance internal forces. From fitting parameters to the simulation results, the change in chain configuration was found to have the majority effect on both molecular energy and Young’s modulus. Additionally the parameters describing the change in chain configuration for both Young’s modulus and molecular energy were inter related. Finally, no relationship was found between bonding levels and interphase thickness. This final result does not match current theory.

The results from the PMMA-alumina simulations suggests both future experiments and possible material improvements. The Young’s modulus of the bonded composite suggests the bonding is a major factor in controlling the Young’s modulus. If the bonding could be controlled, nano composites could be designed with very specific tensile properties. Additionally, the hypothesis that the nano composite’s Young’s modulus is the results of the statistical distribution of bonding states should be tested using MD simulations with many nano particles at different levels of bonding. The result relating the change in the density of the bulk region to the change in the first peak needs experimental confirmation; additionally, the existence of the two other peaks also needs confirmation. The model formed from an infinite series of non linear springs in parallel needs further confirmation. Also, the two issues with the model need resolving which will probably lead to an improved model.
**APPENDIX A**

**CALCULATING DIFFERENT ANGULAR SYSTEMS FOR INVERSIONS**

```fortran
program angletophi
  implicit none
  double precision angle, phi

  C angle is the sum of the angles between the three atoms
  C phi is the improper dihedral angle between the 4 atom group
  double precision atoms(3,4), atomsinit(3,4), atom(3),
  tempatom(3)
  integer roll1, roll2, pitch1, pitch2
  double precision rx1(3,3), rx2(3,3), ry1(3,3), ry2(3,3),
  T1(3,3), T2(3,3)
  double precision a(3),b(3),c(3),d(3),e(3),f(3),na(3),nb(3)
  double precision sa,sb,sc,se,sf,sna,snb
  double precision ctheta,stheta

  C roll 1 and roll 2 are the gimbel rx rotation control
  C pitch 1 and pitch 2 are the gimbel ry rotation control
  real pi
  parameter (pi=3.14159)
  integer i,k,j

  C 1st distance is the carbon carbon distance
  C only the oxygen carbon bonds will be rotated
  data atomsinit/0.0, 0.0, 0.0,
    & 1.517, 0.0, 0.0,
    & -0.703786670920162, -0.983038820105857, 0.0,
    & -0.496232426699163, 1.26623590957304, 0.0/

  do 16 i=1,3
    do 17 j=1,2
      atoms(i,j)=atomsinit(i,j)
    17 continue
  16 continue

  open(25,file='angle-phi.txt')
  write (25,*) 'angle phi'

  do 1 roll1=0,315,45
    do 5 i=1,3
      do 6 j=1,3
        C producing rx1
```

122
if (i .eq. j) then
  if (i .eq. 1) then
    rx1(i,j)=1.0
  else
    rx1(i,j)=cos(real(roll1)*pi/180.0)
  endif
elseif (i .eq. 1 .or. j .eq. 1) then
  rx1(i,j)=0.0
elseif (i .eq. 3 .and. j .eq. 2) then
  rx1(i,j)=sin(real(roll1)*pi/180.0)
else
  rx1(i,j)=-sin(real(roll1)*pi/180.0)
endif

C finished producing rx1

C producing rx2

if (i .eq. j) then
  if (i .eq. 1) then
    rx2(i,j)=1.0
  else
    rx2(i,j)=cos(real(roll2)*pi/180.0)
  endif
elseif (i .eq. 1 .or. j .eq. 1) then
  rx2(i,j)=0.0
elseif (i .eq. 3 .and. j .eq. 2) then
  rx2(i,j)=sin(real(roll2)*pi/180.0)
else
  rx2(i,j)=-sin(real(roll2)*pi/180.0)
endif

C finished producing rx2

C producing ry1

if (i .eq. j) then
  if (i .eq. 2) then

C finished producing ry1
ry1(i,j)=1.0
else
  ry1(i,j)=cos(real(pitch1)*pi/180.0)
endif
elseif (i .eq. 3 .and. j .eq. 1)
  ry1(i,j)=-sin(real(pitch1)*pi/180.0)
else (i .eq. 1 .and. j .eq. 3)
  ry1(i,j)=sin(real(pitch1)*pi/180.0)
else
  ry1(i,j)=0.0
endif

C finished producing ry1

C producing ry2
  if (i .eq. j) then
    if (i .eq. 2) then
      ry2(i,j)=1.0
    else
      ry2(i,j)=cos(real(pitch2)*pi/180.0)
    endif
  elseif (i .eq. 3 .and. j .eq. 1)
    ry2(i,j)=-sin(real(pitch2)*pi/180.0)
  elseif (i .eq. 1 .and. j .eq. 3)
    ry2(i,j)=sin(real(pitch2)*pi/180.0)
  else
    ry2(i,j)=0.0
  endif

C finished producing ry2

C calculate T1 (transformation matrix 1) and T2 (transformation matrix 2)
call dmm3calc(rx1,ry1,T1)
call dmm3calc(rx2,ry2,T2)
C write (25,*) 't1 is', T1
C write (25,*) 't2 is', T2

C calculate the rotated atoms (atom 3 and 4 [atom1 and 2 do not change ever])
    do 13 j=3,4
        do 14 i=1,3
            atom(i)=atomsinit(i,j)
        14
        continue
        if (j .eq. 3) then
            call dmvcalc(3,3,T1,atom,tempatom)
        else
            call dmvcalc(3,3,T2,atom,tempatom)
        endif
        do 15 i=1,3
            atoms(i,j)=tempatom(i)
        15
        continue
    13

C calculate the needed vectors for the angle calculations and theta calculation
    do 18 i=1,3
        a(i)=atoms(i,2)-atoms(i,1)
        b(i)=atoms(i,3)-atoms(i,2)
        c(i)=atoms(i,3)-atoms(i,4)
        d(i)=a(i)
        e(i)=atoms(i,3)-atoms(i,1)
        f(i)=atoms(i,4)-atoms(i,1)
    18
    continue

C calculate the sizes for the above vectors
    call vsize(a,3,sa)
    call vsize(b,3,sb)
    call vsize(c,3,sc)
    sd=sa
    call vsize(e,3,se)
    call vsize(f,3,sf)

C calculate the three angles ([2,1,3], [3,1,4], [2,1,4]) and sum them to calculate angle
    angle=0
    do 19 i=1,3
        if (i .eq. 1) then
            call fcos(d,e,sd,se,ctheta)
        angle=angle +acos(ctheta)*180.0/pi
elseif (i .eq. 2) then
    call fcos(e,f,se,sf,ctheta)
    angle=angle
+acos(ctheta)*180.0/pi
else
    call fcos(f,d,sf,sd,ctheta)
    angle=angle
+acos(ctheta)*180.0/pi
endif

C calculate na and nb and their size (na=bx a and nb=cxa)
call xprod(b,a,na)
call xprod(c,a,nb)
call vsize(na,3,sna)
call vsize(nb,3,snb)

C Calculate cos phi and sin phi
    call fcos(na,nb,sna,snb,ctheta)
call fsin(nb,b,snb,sna,sa,stheta)

C calculate phi using -atan(stheta/ctheta)
    phi=-atan(stheta/ctheta)*180.0/pi
write (25,*) angle, phi
4
3  continue
2  continue
1  continue

end

subroutine dmm3calc(rotomat1,rotomat2,rotomat)
implicit none
C global variables
    double precision rotomat1(3,3), rotomat2(3,3), rotomat(3,3)
C local variables
    integer I, J, K
    DO 2 I = 1, 3
        DO 2 J = 1, 3
            rotomat(I,J) = 0D0
        DO 1 K = 1, 3
            rotomat(I,J) = rotomat(I,J) + rotomat1(I,K) *
rotomat2(K,J)
1  CONTINUE
2  CONTINUE
return
end

subroutine dmvcalc(mrow, mcol, A, b, c)
implicit none
C global variables
  integer mrow, mcol
  double precision A(mrow,mcol), b(mcol), c(mrow)
c local variables
  integer i, j
c this routine solves the problem c=A*b where
  C A is a matrix of mrow and mcol
  c and c is a vector with # of rows = to mcol
  do 65 j=1, mrow
    c(j)=0
    do 66 i=1, mcol
      c(j)=c(j)+A(j,i)*b(i)
66    continue
65  continue
return
end

subroutine xprod(a,b,c)
  implicit none
C global variables
  double precision a(3),b(3),c(3)
c(1)=a(2)*b(3)-a(3)*b(2)
c(2)=a(3)*b(1)-a(1)*b(3)
c(3)=a(1)*b(2)-a(2)*b(1)
return
end

subroutine vsize(v,l,s)
  implicit none
C global variables
  integer l
double precision v(3), s
c local variables
  integer i
  s=0
  do 74 i=1,l
    s=s+v(i)**2
74  continue
s=sqrt(s)
return
end

subroutine fcos(v1,v2,s1,s2,ctheta)
  implicit none
C global variables
  double precision v1(3),v2(3), s1, s2, ctheta
c local variables
double precision ans
call dot(v1,v2,ans)
cttheta=ans/s1/s2
return
dot

subroutine fsin(v1,v2,s1,s3,s4,stheta)
implicit none
C global variables
double precision v1(3),v2(3),s1,s3,s4,stheta
C local variables
double precision ans
C this is of the form stheta=(v1)dot(v2)*s4/s3/s1
call dot(v1,v2,ans)
sttheta=ans*s4/s3/s1
return
dot

subroutine dot(v1,v2,ans)
C global variables
double precision v1(3),v2(3),ans
C local variables
integer i
ans=0
do 69 i=1,3
    ans=ans+v1(i)*v2(i)
69 continue
return
end
APPENDIX B

PRODUCING INITIAL PMMA STRUCTURE FOR MD

program pmma_uam
    implicit none
    C Ability to set up PMMA chains around a hollow sphere in a
    C simulation box of size lx,ly,lz
    C lx,ly,lz depend on the %alumina mixed into the pmma. The size
    C always be 1 angstrom diameter. But in this version the hollow
    C sphere will be skipped.
    integer chainnum, atomnum, unitnum, kuhncount
    parameter (chainnum=19, unitnum=100, atomnum=7, kuhncount=7)
    c Chainnum is the number of chains, unitnum is the number of pmma
    c units in the chain
    c atomnum is the number of particles in a unit
    integer bondnum, anglenum, dihedralnum, impropernum
    integer atomtypenum, bondtypenum, angletypenum, dihedraltypenum, impropertypenum
    parameter (atomtypenum=6, bondtypenum=6, angletypenum=6, dihedraltypenum=6, impropertypenum=1)
    double precision atoms(5,atomnum), adition(3),
    masses(atomtypenum)
    double precision chainatoms(3,unitnum*atomnum),
    pbcchain(3,unitnum*atomnum)
    integer imageflag(3,unitnum*atomnum)
    double precision bondcoef(2,bondtypenum),
    anglecoef(2,angletypenum)
    double precision dihedralcoef(5,dihedraltypenum),impropercoef(2,impropertypenum)
    integer bondinit(3,6), bonds(3,7), angleinit(4,7),
    angles(4,11)
    integer dihedrallimit(5,6), dihedrals(5,13), impropers(5)
    double precision hspherer, latticesize
    parameter (hspherer=0.0, latticesize=64.422891)
    C hspherer, latticesize and minshift are in angstrom.
    C hspherer is the radius of the hollow sphere in the center of the
    simulation box
    double precision shiftdist, shiftunit(3), shift(3), dist2
    integer i, j, k, l,n, imageflag, count, unitcount
    data atoms/3, .11, 0, 0, 0,
    & 2, 0, -1.52429914864554, 0,.2193447182826,
    & 1, 0, .333005967244697, -1.23820341059349, -.8529708903439,
    & 4, .31, .75, 0, 1.29903810567666,
& 5, -.37, .26258675537238, .528275496308508, 2.28481360198517,
& 6, -.31, 2.03631224555881, -.619208163705874, 1.38199626938253,
& 1, .26, 2.43869486125905, -.42813272480992, 1.38199626938253,
& edition/.333005967244697, 1.23820341059349, -.852970890343899/
& masses/12.0107, 12.0107, 12.0107, 12.0107, 15.9994, 15.9994/,
& bondinit/1, 1, 3,
& 2, 1, 2,
& 3, 1, 4,
& 4, 4, 5,
& 5, 4, 6,
& 6, 6, 7/,
& bonds/2, 1, -5,
& 1, 1, 3,
& 2, 1, 2,
& 3, 1, 4,
& 4, 4, 5,
& 5, 4, 6,
& 6, 6, 7/,
& bondcoef/368, 1.539,
& 300, 1.549,
& 326, 1.517,
& 968, 1.209,
& 471, 1.360,
& 342, 1.446/,
& angleinit/2, 2, 1, 3,
& 2, 3, 1, 4,
& 2, 2, 1, 4,
& 4, 1, 4, 5,
& 3, 1, 4, 6,
& 5, 6, 4, 5,
& 6, 7, 6, 4/,
& angles/1, -6, -5, 1,
& 2, -5, 1, 4,
& 2, -5, 1, 3,
& 2, 2, 1, 3,
& 2, 2, 1, 4,
& 2, 2, 1, -5,
& 2, 3, 1, 4,
& 3, 6, 4, 1,
& 4, 5, 4, 1,
& 5, 6, 4, 5,
& 6, 7, 6, 4/,
& anglecoef/89.5, 113.3,
& 87.9, 109.47,
& 74.5, 111.4,
& 63.3, 125.6,
& 126.5, 123.0,
& 84.8, 116.4/,
& dihedralinit/4, 3, 1, 4, 5,
& 4, 2, 1, 4, 5,
& 3, 3, 1, 4, 6,
& 3, 2, 1, 4, 6,
& 5, 1, 4, 6, 7,
& 6, 7, 6, 4, 5/,
& dihedrals/1, 1, -5, -6, -4,
& 2, 1, -5, -6, -3,
& 1, 2, 1, -5, -6,
& 1, 3, 1, -5, -6,
& 2, 4, 1, -5, -6,
& 4, 5, 4, 1, -5,
& 3, 6, 4, 1, -5,
& 4, 3, 1, 4, 5,
& 4, 2, 1, 4, 5,
& 3, 3, 1, 4, 6,
& 3, 2, 1, 4, 6,
& 6, 7, 6, 4, 5,
& 5, 7, 6, 4, 1/,
& dihedralcoef/.00043047, .88728, -.0058908, 3.48522, .0083951,
& -.69938, .88749, 1.39251, 3.48489, .010033,
& -.00011517, 1.08001, .00042681, 1.76002, -.00036242,
& 1.90050, -.78234, -3.80681, 1.04477, .0094108,
& 2.22989, 4.54992, -4.45939, -.63999, -.0051598,
& 2.20044, 1.04739, -4.40604, -1.39492, .0085622/,
& impropers/1, 4, 1, 5, 6/,
& impropercoef/0.735006480784105, 0/
dihedralnum=chainnum*(atomnum-1)+chainnum*(unitnum-1)*(2*atomnum-1)

impropernum=chainnum*unitnum
write (25,*) bondnum, ' bonds'
write (25,*) anglenum, ' angles'
write (25,*) dihedralnum, ' dihedrals'
write (25,*) impropernum, ' impropers'

C set up the the number of different types of bonds, angles, dihedrals and impropers
write (25,*) atomtypenum, ' atom types'
write (25,*) bondtypenum, ' bond types'
write (25,*) angletypenum, ' angle types'
write (25,*) dihedraltypenum, ' dihedral types'
write (25,*) impropertypenum, ' improper types'

do 2 i=1,3
  write (25,*)
2  continue

call srand(14)
c  call srand(7)
c  call srand(3)
write (25,*) 'Atoms'
write (25,*)
open (30,file='pmma100.xyz')

C write the xyz file header
write (30,*) chainnum*unitnum*atomnum
write (30,*) 'atoms'

C atom counter for polymer data file
count=1

C build the atoms in the box
do 15 i=1,chainnum
  c  place in first unit of the new chain randomly
1   do 16 j=1,3
      chainatoms(j,1)=latticesize*rand(0)-latticesize/2.0
      call pbc(chainatoms(j,1),latticesize,pbcchain(j,1)), imageflag(j,1))
16   continue
   if (hspherer .ne. 0) then
      dist2=pbcchain(1,1)**2+pbcchain(2,1)**2+pbcchain(3,1)**2
      if (dist2 .le. hspherer**2) goto 1
   endif

do 20 j=2,atomnum
  C just do the addition of atoms to chainatoms and pbcchain
  do 22 k=1,3
chainatoms(k,j) = chainatoms(k,1) + atoms(k + 2, j)

    call pbc(chainatoms(k,j),latticesize, pbccchain(k,j), imageflag(k,j))

C checking to make sure atoms are not in the hollow sphere
    if (hspherer .ne. 0) then
        dist2 = pbccchain(1,j)**2 + pbccchain(2,j)**2 + pbccchain(3,j)**2
        if (dist2 .le. hspherer**2) goto 1
    endif

C above if statement skips code below
    if (hspherer .ne. 0) then
        if (dist2 .le. hspherer**2) goto 1
    endif

    continue
    unitcount = 1

    do 10 j = 2, unitnum
        sphereflag = 0
        do 5 k = 1, atomnum
            C just do the addition of atoms to chainatoms and pbccchain
            dist2 = 0
            do 7 l = 1, 3
                if (k == 1) then
                    chainatoms(l, (j-1)*atomnum + k) = chainatoms(l, (j-1)*atomnum - 5) + adition(l)
                else
                    chainatoms(l, (j-1)*atomnum + k) = chainatoms(l, (j-1)*atomnum + 1) + atoms(l + 2, k)
                endif
            call pbc(chainatoms(l, (j-1)*atomnum + k), latticesize, pbccchain(l, (j-1)*atomnum + k), imageflag(l, (j-1)*atomnum + k))
            if (sphereflag .eq. 1) goto 7
        endif
        C above if statement skips code below
        if (hspherer .ne. 0) then
            dist2 = pbccchain(l, (j-1)*atomnum + k)**2 + dist2
        endif
        continue
        if (sphereflag .eq. 1) goto 7
    endif
    sphereflag = 1
    C check to see if atoms are in the sphere
c if so trigger
sphereflag
5
    continue
    if (sphereflag .eq. 1) then
      call mirror(unitnum,atomnum,j-1,chainatoms,
pbcchain, atoms, adition, latticesize, imageflag)
      unitcount=0
    elseif (unitcount .eq. kuhncount) then
      call mirror(unitnum,atomnum,j-1,chainatoms,
pbcchain, atoms, adition, latticesize, imageflag)
      unitcount=0
    else
      unitcount= unitcount + 1
    endif

10  continue

if (hspherer .ne. 0) then
  c checks to make sure no atoms of the current chain are in the sphere
  do 18  j=2,unitnum
    c Because of how the chain is built the 1st unit cannot be in the sphere so start with unit 2
    do 78 k=1,atomnum
      c Check the atoms in each unit
      dist2=0
      do 12 l=1,3
        dist2=dist2+pbcchain(l,
        (j-1)*atomnum+k)**2
      12     continue
      if (dist2 .le. hspherer**2) then
        c calculate the shift unit vector
        dist2=sqrt(dist2)
        do 9 l=1,3
          shiftunit(l)=pbcchain(l,
          (j-1)*atomnum+k)/dist2
        9       continue
        shiftdist=hspherer+.1
        C calculate the chainatoms, imageflag and pbcchain
        do 13 l=1,3
          chainatoms(l,(j-1)*atomnum
          +k)=chainatoms(l,(j-1)*atomnum+k)+shiftdist*shiftunit(l)\-pbcchain(l,(j-1)*atomnum+k)
        13     call pbc(chainatoms(l,
        (j-1)*atomnum+k), latticesize, pbcchain(l,(j-1)*atomnum+k),
        imageflag(l,(j-1)*atomnum+k))
      13  continue
C write the final chain structure in the data file and in a .xyz file

    do 6 j=1,unitnum
        do 3 k=1,atomnum
            C data file
            write (25,*) count, i, int(atoms(1,k)),
            atoms(2,k), pbcchain1,(j-1)*atomnum+k),
            & pbcchain2,(j-1)*atomnum+k),
            pbcchain3,(j-1)*atomnum+k), imageflag1,(j-1)*atomnum+k),
            & imageflag2,(j-1)*atomnum+k),
            imageflag3,(j-1)*atomnum+k)
            count=count+1
    C xyz file
    if (atoms(1,k) .lt. 5) then
        write (30,*) 6, pbcchain1,(j-1)*atomnum+k),
        & pbcchain2,(j-1)*atomnum+k),
        pbcchain3,(j-1)*atomnum+k)
    else
        write (30,*) 8, pbcchain1,(j-1)*atomnum+k),
        & pbcchain2,(j-1)*atomnum+k),
        pbcchain3,(j-1)*atomnum+k)
    endif
    3 continue
    6 continue
C Reset atoms and adition
C    call reset(atoms, adition, atomnum)
    15 continue
    close (30)
    write (25,*)

C Writing the masses
    write (25,*) 'Masses'
    write (25,*)
    do 25 i=1,atomtypenum
        write (25,*) i, masses(i)
    25 continue
    write (25,*)

C write the bonds secton
    write (25,*) 'Bonds'
    write (25,*)
    count=1
    do 35 i=1,chainnum
        do 36 j=1,atomnum-1
write (25,*) count, bondinit(1,j), bondinit(2,j) + (i-1)*unitnum*atomnum,
& bondinit(3,j)+(i-1)*unitnum*atomnum
count=count+1
36 continue
37 do j=2,unitnum
39 do k=1,atomnum
write (25,*) count, bonds(1,k), bonds(2,k)+(i-1)*unitnum*atomnum+(j-1)*atomnum,
& bonds(3,k)+(i-1)*unitnum*atomnum+(j-1)*atomnum
count=count+1
39 continue
37 continue
35 continue
write (25,*)

C write the bond coefficients section
write (25,*) 'Bond Coeffs'
write (25,*)
doi=1,bondtypenum
30 write (25,*) i, bondcoef(1,i), bondcoef(2,i)
30 continue
write (25,*)

C write the angles section
write (25,*) 'Angles'
write (25,*)
count=1
doi=1,chainnum
40 do j=1,atomnum
41 write (25,*) count, angleinit(1,j), angleinit(2,j)+(i-1)*unitnum*atomnum,
& angleinit(3,j)+(i-1)*unitnum*atomnum,
angleinit(4,j)+(i-1)*unitnum*atomnum
count=count+1
41 continue
40 do j=2,unitnum
42 do k=1,atomnum+4
44 write (25,*) count, angles(1,k), angles(2,k)+(i-1)*unitnum*atomnum+(j-1)*atomnum,
& angles(3,k)+(i-1)*unitnum*atomnum+(j-1)*atomnum,
& angles(4,k)+(i-1)*unitnum*atomnum+(j-1)*atomnum
count=count+1
44 continue
42 continue
40 continue
write (25,*)

C write the angle coefficients section
write (25,*) 'Angle Coeffs'
write (25,*)
do 43 i=1,angletypenum
   write (25,*) i, anglecoef(1,i), anglecoef(2,i)
43 continue
write (25,*)

C write the dihedral section
write (25,*) 'Dihedrals'
write (25,*) count=1
do 45 i=1,chainnum
do 46 j=1,atomnum-1
   write (25,*) count, dihedralinit(1,j),
   dihedralinit(2,j)+(i-1)*unitnum*atomnum,
   & dihedralinit(3,j)+(i-1)*unitnum*atomnum,
   dihedralinit(4,j)+(i-1)*unitnum*atomnum,
   & dihedralinit(5,j)+(i-1)*unitnum*atomnum
   count=count+1
46 continue
do 47 j=2,unitnum
   do 48 k=1,2*atomnum-1
      write (25,*) count, dihedrals(1,k),
      & dihedrals(2,k)+(i-1)*unitnum*atomnum+
      (j-1)*atomnum,
      & dihedrals(3,k)+(i-1)*unitnum*atomnum+
      (j-1)*atomnum,
      & dihedrals(4,k)+(i-1)*unitnum*atomnum+
      (j-1)*atomnum,
      & dihedrals(5,k)+(i-1)*unitnum*atomnum+
      (j-1)*atomnum
   count=count+1
48 continue
47 continue
45 continue
write (25,*)

C write the dihedral coefficient section
write (25,*) 'Dihedral Coeffs'
write (25,*)
do 49 i=1,dihedraltypenum
   write (25,*) i, dihedralcoef(1,i), dihedralcoef(2,i),
   dihedralcoef(3,i), dihedralcoef(4,i),
   & dihedralcoef(5,i)
49 continue
write (25,*)

C write impropers section
write (25,*) 'Impropers'
write (25,*)
count=1
do 53 i=1,chainnum
   do 52 j=1,unitnum
      write (25,*) count, impropers(1),
      &\ impropers(2)+(i-1)*unitnum*atomnum+
      &\ (j-1)*atomnum,
      &\ impropers(3)+(i-1)*unitnum*atomnum+
      &\ (j-1)*atomnum,
      &\ impropers(4)+(i-1)*unitnum*atomnum+
      &\ (j-1)*atomnum,
      &\ impropers(5)+(i-1)*unitnum*atomnum+
      &\ (j-1)*atomnum
      count=count+1
   52 continue
53 continue
write (25,*)

C write the improper coefficient section
write (25,*) 'Improper Coeffs'
write (25,*)
do 51 i=1,impropertypenum
   write (25,*) i, impropercoef(1,i), impropercoef(2,i)
51 continue
write (25,*)
close (25)
end

subroutine mirror(unitnum,atomnum,j,chainatoms, pbcchain, atoms, adition,latticesize, imageflag)
implicit none
C global variables
integer unitnum, atomnum, j
C j is the current unitnumber
double precision chainatoms(3,unitnum*atomnum),
pbcchain(3,unitnum*atomnum)
   double precision atoms(5,atomnum), adition(3), latticesize
   integer imageflag(3,unitnum*atomnum)
C local variables
   double precision oldunit(3,atomnum), newunit(3,atomnum)
C oldunit and newunit are in terms of the pbc positions
C old is the current unit in the pbc and new is the transformed one.
double precision mirrorpoint(3), dist2, dot
double precision atomtemp(3), tempatom(3)
integer i,k
C this algorithm calculates a reflection when the plane of the
mirror is parallel to a sphere defined
C by the vector mirrorpoint. mirrorpoint is also the normal
vector of the mirror.

C Calculate dist2
dist2=0
do 55 i=1,3
   mirrorpoint(i)=chainatoms(i,(j*atomnum)-5)
   write (*,*) 'the mirrorpoint at index ', i, ' is ',
mirrorpoint(i)
   dist2=dist2+mirrorpoint(i)**2
55 continue

C Extract oldunit from pbcchain
C atomtemp is in oldunit coordinate system
C tempatom is in newunit coordinate system
C calculate newunit using mirror formula
C tempatom=atomtemp - 2 * (atomtemp (dot) mirrorpoint - dist2) /
dist2 * mirrorpoint
C Use the newunit and oldunit to calculate the new position of
chainatoms
C Then calculate the pbcchain using the new chainatoms
   do 59 i=1,atomnum
      do 60 k=1,3
         oldunit(k,i)=chainatoms(k,i+j*atomnum)
         atomtemp(k)=oldunit(k,i)
      C      write (*,*) 'the atom at position ', i, ' index
50      , k, ' is ', atomtemp(k)
     60   continue
   C Using mirror formula
     dot=0.0
     do 68 k=1,3
        dot=dot+atomtemp(k)*mirrorpoint(k)
     68   continue
     do 65 k=1,3
        tempatom(k)=atomtemp(k)-2*(dot-dist2)/
dist2*mirrorpoint(k)
     C      write (*,*) 'the mirrored atom at position ', i, ' index
85      , k, ' is ',tempatom(k)
    65   continue
     do 70 k=1,3
        newunit(k,i)=tempatom(k)
    C     chainatoms(k,i+j*atomnum)=chainatoms(k,i
80     +j*atomnum)+newunit(k,i)-oldunit(k,i)
chainatoms(k,i+j*atomnum)=newunit(k,i)
call pbc(chainatoms(k,i+j*atomnum),latticesize,pbcchain(k,i+j*atomnum), imageflag(k,i+j*atomnum))
70 continue
59 continue

C Calculate adition
   do 80 k=1,3
      adition(k)=newunit(k,1)-mirrorpoint(k)
   C remember mirrorpoint is where the pmma unit is attached to the old chain
   80 continue

C Calculate atoms (The 1st column x,y,z values are all 0,0,0)
   do 90 i=2,atomnum
      do 100 k=1,3
         atoms(k+2,i)=newunit(k,i)-newunit(k,1)
      100 continue
   90 continue
end return

subroutine pbc(atompos,latticesize, pbcpos, imageflag)
   implicit none
   C global variables
   double precision atompos, latticesize, pbcpos
   integer imageflag
   C local variables
   double precision min, max, delta
   C add image flag into this code
   max=latticesize/2.0
   min=-latticesize/2.0
   delta=max-min
   if (atompos .lt. min) then
      pbcpos=atompos+delta+delta*int(abs((atompos-min)/delta))
      imageflag=-1-int(abs((atompos-min)/delta))
   elseif (atompos .ge. max) then
      pbcpos=atompos-delta-delta*int(abs((atompos-max)/delta))
      imageflag=1+int(abs((atompos-max)/delta))
   else
      pbcpos=atompos
      imageflag=0
   endif
   return
end

subroutine reset(atoms, adition, atomnum)
C global variables
integer atomnum
double precision atoms(5,atomnum), adition(3)
C only need to reset the position data of atoms so only need to reset rows 3-5

atoms(3,1)=0
atoms(4,1)=0
atoms(5,1)=0

atoms(3,2)=-1.52429914864554
atoms(4,2)=0
atoms(5,2)=.2193447182826

atoms(3,3)=.333005967244697
atoms(4,3)=-1.23820341059349
atoms(5,3)=-.8529708903439

atoms(3,4)=.75
atoms(4,4)=0
atoms(5,4)=1.2990381056766

atoms(3,5)=.262586755537238
atoms(4,5)=.528275496308508
atoms(5,5)=2.28481360198517

atoms(3,6)=2.03631224555881
atoms(4,6)=-.619208163705874
atoms(5,6)=1.38199626938253

atoms(3,7)=2.43869486125905
atoms(4,7)=-.42813272480992
atoms(5,7)=2.74084814035043

adition(1)=.333005967244697
adition(2)=1.23820341059349
adition(3)=-.852970890343899
return
end
APPENDIX C

RNG ALGORITHM IN PERIODIC SPACE

\[ ax + by + cz + d = 0 , \quad (C.1) \]

\[ \tilde{n} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \tilde{p}_{\text{mirror}\text{po int}} - \tilde{p}_{\text{nanoparticle}} , \quad (C.2) \]

\[ d = -\tilde{n} \cdot \tilde{p}_{\text{mirror}\text{po int}} . \quad (C.3) \]

\[ \bar{x}'_{\text{atom}} = \bar{x}_{\text{atom}} - 2 \frac{\tilde{p}_{\text{atom}} \cdot \tilde{n} + d}{\tilde{n} \cdot \tilde{n}} \tilde{n} , \quad (C.4) \]
APPENDIX D

BULK PMMA MD SIMULATIONS AND PROPERTIES

D.1 Initial Equilibrium

#!/usr/bin/env python
#
#
#
#
#
f=open('in.pmma','w')
f.write("#
units               real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic

read_data data.pmma100

pair_style lj/cut/coul/cut 10 15
pair_modify shift yes
pair_coeff 1 1 .2028 3.7888
pair_coeff 1 2 .1394 3.9090
pair_coeff 2 2 .09586 4.0330
pair_coeff 1 3 .00338 5.3295
pair_coeff 2 3 .002324 5.4985
pair_coeff 3 3 .000056338 7.4966
pair_coeff 1 4 .08889 3.8570
pair_coeff 2 4 .06111 3.9793
pair_coeff 3 4 .001481 5.4254
pair_coeff 4 4 .0401 3.9067
pair_coeff 1 5 .2257 3.2312
pair_coeff 2 5 .1551 3.3337
pair_coeff 3 5 .0037624 4.5451
pair_coeff 4 5 .1086 3.2385
pair_coeff 5 5 .3361 2.6251
pair_coeff 1 6 .2005 3.2632
pair_coeff 2 6 .1379 3.3666
pair_coeff 3 6 .003343 4.5900
pair_coeff 4 6 .09652 3.2705

143
pair_coeff 5 6 .2987 2.6511
pair_coeff 6 6 .2654 2.6773

group all id <= 6
#region wall sphere 0.0 0.0 0.0 10 side out units box

velocity all create 300 376847 loop geom

neighbor 2 bin
neigh_modify every 1 delay 5 check yes one 5000

thermo 50
dump 2 all custom 50000 dump.pmma* type x y z
fix 1 all nve/limit .1
#fix 3 all wall/region wall lj126 0.11749 4.00078 10
log log.pmma100initeq
run 10000
write_restart pmma100.initeq

##
f.close()

import os
os.system('mpirun -np 4 ../../lammps/src/lmp_openmpi<in.pmma')

D.2 Final Equilibrium

#! /usr/bin/env python
#
#
#
#
#
f=open('in.pmma','w')
f.write("""# pmma chain

units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic
pair_style lj/cut/coul/cut 10 15
pair_modify shift yes
read_restart pmma100.initeq

reset_timestep 0

144
group all id <= 6
#region wall sphere 0.0 0.0 0.0 10 side out units box

velocity all create 300 376847 loop geom

neighbor 2 bin
neigh_modify every 1 delay 5 check yes
timestep 4
run_style respa 2 2
 thermo 50
dump 2 all custom 50000 dump.pmma* type x y z
fix 1 all nvt temp 300 600 100
#fix 3 all wall/region wall lj126 0.11749
4.00078 10
log log.pmma100finaleq1

run 12500
write_restart pmma100.finaleq1
unfix 1
fix 1 all nvt temp 600 600 100
log log.pmma100finaleq2

run 25000
write_restart pmma100.finaleq2
unfix 1
fix 1 all nvt temp 600 298 100
log log.pmma100finaleq3

run 12500
write_restart pmma100.finaleq3

import os
os.system('mpirun -np 4 ../../../lammps/src/lmp_openmpi<in.pmma')

D.3 Bulk Property NVT

#!/usr/bin/env python
#
#  

145
f=open('in.pmma','w')
f.write(""
units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic
pair_style lj/cut/coul/cut 10 15
pair_modify shift yes
read_restart pmma100.finaleq3
reset_timestep 0
read_restart pmma100.finaleq3

#region
wall sphere 0.0 0.0 0.0 10 side out units box

#velocity
all create 300 376847 loop geom

neighbor 2 bin
neigh_modify every 1 delay 5 check yes

timestep 4
run_style respa 2 2

thermo 50
fix 1 all nvt temp 298 298 100
log log.pmma100finaleq

run 25000
write_restart pmma100.finaleq4

log log.pmma100densitycalc

run 1250
write_restart pmma100.finaleq5

run 1250
write_restart pmma100.finaleq6

run 1250
write_restart pmma100.finaleq7

run 1250
write_restart pmma100.finaleq8
run 1250
write_restart pmma100.finaleq9

run 1250
write_restart pmma100.finaleq10

run 1250
write_restart pmma100.finaleq11

run 1250
write_restart pmma100.finaleq12

run 1250
write_restart pmma100.finaleq13

run 1250
write_restart pmma100.finaleq14

run 1250
write_restart pmma100.finaleq15

run 1250
write_restart pmma100.finaleq16

run 1250
write_restart pmma100.finaleq17

run 1250
write_restart pmma100.finaleq18

run 1250
write_restart pmma100.finaleq19

run 1250
write_restart pmma100.finaleq20

run 1250
write_restart pmma100.finaleq21

run 1250
write_restart pmma100.finaleq22

run 1250
write_restart pmma100.finaleq23

run 1250
write_restart pmma100.finaleq24

compute 3 all gyration/molecule
log log.pmma100gyration

run 12500
write_restart pmma100.gyration
""

f.close()

import os
os.system('mpirun -np 4 ../../lammps/src/lmp_openmpi<in.pmma')

D.4 Young’s Modulus

#!/usr/bin/env python
#
# currently strained at 1% or .01 which is probably too high
# changing strain rate from 1e10 to 1e8 to better match what is
# in the literature
# will still strain sample through 1k timesteps giving strain
# of .0001 or .01%
# now testing a strain sample through 10k timesteps giving strain
# of .001 or .1%

def lammpsstresscalc():
    import os
    f = open('in.tensile', 'w')  # file for running tensile test
    f.write(  
        'units
        atom_style full  
        bond_style harmonic  
        angle_style harmonic  
        dihedral_style multi/harmonic  
        improper_style harmonic  
        pair_style lj/cut/coul/cut 10 15  
        pair_modify shift yes  
        read_restart pmma100.finaleq4

    reset_timestep 0
    group  
        all id <= 6
    #region  
        wall sphere 0.0 0.0 0.0 10 side out units box

    #velocity  
        all create 300 376847 loop geom

    neighbor  
        2 bin
    neigh_modify  
        every 1 delay 5 check yes

timestep 2
    run_style verlet
```
thermo 50
#dump 2 all custom 100000 dump.pmma* type x y z
fix 1 all npt temp 298 298 100 iso 0 0 100 drag 1
log log.pmma100npt_init_eq
run 50000
write_restart pmma100.npt_init_eq

# store final cell length for strain calculations
variable tmp equal "lx"
variable LO equal ${tmp}
print "Initial Length, LO: ${LO}"

unfix 1
fix 1 all npt temp 298 298 100 y 0 0 100 z 0 0 100
variable srate equal 1e8
variable srate1 equal "v_srate / 1.0e15"
fix 2 all deform 1 x erate ${srate1} units box remap x
variable strain equal "(lx-v_LO)/v_LO"
variable p1 equal "-pxx/9.8692316931"
thermo_style custom step temp v_strain v_p1 press
log log.pmma100npt_strain
run 50000
write_restart pmma100.npt_strain

unfix 2
log log.pmma100npt_final_eq
run 50000
write_restart pmma100.npt_final_eq

```

```python
f . close()
o s . s y s t e m ( 'mpirun -np 4 ../../lammps/src/ lmp_openmpi<in.tensile')

def stresscalc(logfile):
youngmod=[]
f=open('{}'.format(logfile),'r')
strain=[] #initialize temperature list
stress=[] # initialize volume List
for line in f:
    row=line.split()
    try:
        int(row[0]) # Checks to make sure line in file
        is a data line and not a text line
    except ValueError:
        if row[0]=='Loop': break #ends data aquisition from file
```
else: continue  # otherwise
continues reading the file
    else: #acquires important data
        strain.append(float(row[2]))
        stress.append(float(row[3]))

f.close()
    # try calculating youngmodulus using 1000 step data.
    stepsize=50
    guessedmod=0.0
    for steps in range(50000,0,-stepsize):
        i=steps/stepsize
        guessedmod+=stress[i-1]/strain[i-1]  # guessed youngmod
    num=float(50000/stepsize)
    youngmod.append(guessedmod/num)
return youngmod

def stressexam(logfile):
    f = open('{0}'.format(logfile), 'r')
    strain=[]  #initialize temperature list
    stress=[]  # initialize stress list
    for line in f:
        row = line.split()
        try:
            int(row[0])  # Checks to make sure line in file
            is a data line and not a text line
        except ValueError:
            if row[0]=='Loop': break  #ends data aquisition
        from file
        else: continue  # otherwise
        continues reading the file
    else: #acquires important data
        strain.append(float(row[2]))
        stress.append(float(row[3]))

f.close()
return strain,stress

def datawrite(header,youngmod):
    f = open('modulusdata.txt','a')
    f.write('{0} {1}\n'.format(header, youngmod))
    f.close()
    #from pylab import *
lammpsstresscalc()
youngmod=stresscalc('log.pmma100npt_final_eq')
strain,stress,stressexam('log.pmma100npt_strain')
datawrite('pmma100bulk',youngmod)
    #plot(strain,stress,'k-')

# graphing variables
D.5 Glass Transition

#!/usr/bin/env python
#  read in important information from ./controls/control.txt
#  reads in important property information from ./
database/\[element names]\materialconst.txt
#  writes a file named in.texp
# Run Lammmps using in.texp
# Use two different log files to calculate thermal expansion
coefficient
# write the thermal expansion coefficient to ans.txt in this
folder

def increasetemp(init_temp,temperatures):
    import os
    f = open('in.pmma', 'w')
    f.write("# pmma chain
    units real
    atom_style full
    bond_style harmonic
    angle_style harmonic
dihedral_style multi/harmonic
    improper_style harmonic
    pair_style lj/cut/coul/cut 10
    pair_modify shift yes
    read_restart pmma100.npt_init_eq

reset_timestep 0

timestep 2
run_style verlet

neighbor 2 bin
neigh_modify every 1 delay 5 check yes

thermo 50
    f.write("\nfix 1 all npt temp \{0\} \{1\} 100 iso 0 0 100
drag 1\n".format(init_temp,temperatures[0]))
    f.write("log                  log.pmma100tempincrease

run 50000
write_restart pmma100.temp
def lammppsvolcalc(temperatures):
    import os
    for temperature in temperatures:
        # loop over all temperatures
        f = open('in.pmma', 'w')
        f.write("****"  # pmma chain
            
            units   real
            atom_style full
            bond_style harmonic
            angle_style harmonic
            dihedral_style multi/harmonic
            improper_style harmonic
            pair_style lj/cut/coul/cut 10
            pair_modify shift yes
            read_restart pmma100.temp

            reset_timestep 0

            timestep 2
            run_style verlet

            neighbor          2 bin
            neigh_modify      every 1 delay 5 check yes

            thermo          50
            thermo_style custom step temp epair emol etotal enthalpy vol press "***")

        f.write("\nfix          1 all npt temp {0} {0} 100 iso 0
          0 100 drag 1
".format(temperature))
        f.write("log{0}".format(temperature))
        f.write("***")
        run 25000
        write_restart pmma100.temp

        ****")
    f.close()
    os.system('mpirun -np 4 ..../lammmps/src/lmp_openmpi in.pmma')
def volcalc(temperatures):
    tempav=[]
    volav=[]
    for temperature in temperatures:
        f=open('log.pmma100.{0}'.format(temperature), 'r')
        Temp=[] #initialize temperature list
        Lat=[]  # initialize volume List
        for line in f:
            row=line.split()
            try:
                int(row[0])  # Checks to make sure line in file is a data line and not a text line
            except ValueError:
                if row[0]=='Loop': break  #ends data acquisition from file
                else: continue            # otherwise continues reading the file
            except IndexError:
                continue
            else:  #acquires important data
                Temp.append(float(row[1]))
                Lat.append(float(row[6]))
                tempav.append(sum(Temp)/len(Temp))
                volav.append(sum(Lat)/len(Lat))
        f.close()
    return tempav, volav

def datawrite(potential,x,y):
    f=open('latdata{0}.txt'.format(potential), 'w')
    for i in range(len(x)):
        f.write('{0} {1}\n'.format(x[i],y[i]))
    f.close()

from pylab import *
# temperatures for md potentials
inittemp=298
temperatures=[350,355,360,365,370,375,380,385,390,395,400]
#PMMA
increasetemp(inittemp, temperatures)
lammpsvolcalc(temperatures)
tempav,volav=volcalc(temperatures)
datawrite('pmma100',tempav,volav)
plot(tempav,volav,'bp-')

# graphing variables
xlabel('temperature (k)')
ylabel('volume (angstrom cubed)')
import lmpsdata
from pylab import *
def converttodata(init, final, restartfile, datafile):
    for i in range(init, final + 1):
        os.system('../../lammps/tools/restart2data ' + restartfile + str(i) + ' ' + datafile + str(i))

def datacreate(init, final, basefile, atomtype):
    data = []
    for i in range(init, final + 1):
        data.append(lmpsdata.Lmpsfdata(basefile + str(i), atomtype))
    return data
def densitycreate(thickness, startdist, enddist, data):
    density = []
    for slot in data:
        density.append(slot.density(thickness, startdist, enddist))
    return density
def densityplot(density, thickness, file): #need to get rid of thickness information for final code
    # the first index corresponds to which data class the density data belongs to
    # the second index corresponds to whether the data is r(distance)=0 or den(density)=1
    # the third index corresponds to the position of the r or den number in the list
    
    initialise r and den list
    r = range(len(density[0][0]))
    den = range(len(density[0][1]))
Create the r list from the r data.
for i in range(len(r)):
r[i]=density[0][0][i]

Zero den
for i in range(len(den)):
den[i]=0.0

Go through den data from each data class and sum the den data for each r value.
Store the sums in the list den
for i in range(len(density)):
j=1 #for accessing den data
for k in range(len(density[i][j])):
den[k]+=density[i][j][k]

Average the sums in the list den
for i in range(len(den)):
den[i]=den[i]/len(density)#len(density) gives the number of data points

Assumes r and den are the same length which they should be
plot(r,den,label=str(thickness))

Assumes r and den are the same length
f=open(file,'w')
for i in range(len(den)):
    f.write('{0} {1}\n'.format(r[i],den[i]))
f.close()

converttodata(4,24,'pmma100.finaleq','pmma100data.finaleq')
data=datacreate(4,24,'pmma100data.finaleq','full')
density=densitycreate(.5,0.0,31.5,data)
densityplot(density,.5,'density.txt')
xlabel('distance (angstrom)')
ylabel('density (amu/angstrom^3)')
#legend(loc=1)
show()

D.7 Molecular Energy

def calculate_epm(file):
f=open(file,'r')
count=0
molecular_energy=0.0
for line in f:
    row=line.split()
    try:
        int(row[0]) # Checks to make sure line in file is a data line and not a text line
    except ValueError:
        if row[0]=='Loop': break #ends data aquisition from file
    else: continue # otherwise continues reading the file
except IndexError:
    continue
else: #acquires important data
    molecular_energy+=float(row[3])
    count+=1

f.close()
molecular_energy=molecular_energy/count #average molecular energy
epm=molecular_energy #converts average molecular energy to #the molecular energy per molecule
return epm

def datawrite(header, epm):
    f=open('epmdata.txt','a')
    f.write('{0} {1}
'.format(header, epm))
    f.close()

epm=calculate_epm('log.pmma100finaleq')
datawrite('pmma100bulk',epm)

D.8 Radius of Gyration

def rgcalc(file,molecules):
    f=open(file,'r')
    count=0
    rg=0.0
    for line in f:
        row=line.split()
        try:
            int(row[0]) # Checks to make sure line in file is a data line and not a text line
        except ValueError:
            if row[0]=='Loop': break #ends data aquisition from file
        else: continue # otherwise continues reading the file
    except IndexError:
        continue
    else: continue
    epm=calculate_epm('log.pmma100finaleq')
datawrite('pmma100bulk',epm)
else: #acquires important data
    for i in range(1,molecules+1):
        rg+=float(row[i])
        count+=1

    f.close()
    rg=rg/count
    return rg

def datawrite(header,rg):
    f=open('rgdata.txt','a')
    f.write('{0} {1}\n'.format(header, rg))
    f.close()

rg=rgcalc('log.pmma100gyration',8)
datawrite('pmma100bulk',rg)
APPENDIX E

PMMA-ALUMINA POTENTIAL TABLE CODE

#include <fstream>
#include <math.h>
using namespace std;

// global parameters
const double A=11.95, lambda=1.1, zb=2.0; // binding potential (morse) [changed to Kcal/mol]
const double rho=.0603, pi=3.14159; // nonbinding potential (lj) [changed to Kcal/mol]

double a_equation(double x)
{
    return (A*exp(-2.0*lambda*(x-zb)));
}

double b_equation(double x, double B)
{
    return (B*exp(-lambda*(x-zb)));
}

double c_equation(double x, double C)
{
    return (C*pow(lambda,2)*pow(x-zb,2)*exp(-lambda*(x-zb)));
}

double da_equation(double x)
{
    return (-2.0*lambda*A*exp(-2.0*lambda*(x-zb)));
}

double db_equation(double x, double B)
{
    return (-lambda*B*exp(-lambda*(x-zb)));
}

double dc_equation(double x, double C)
{
    return (2.0*C*pow(lambda,2)*pow(x-zb,1)*exp(-lambda*(x-zb)) +
            -lambda*C*pow(lambda,2)*pow(x-zb,2)*exp(-lambda*(x-zb)));
}
double ljfront(double x, double sigma, double epsilon)
{
    return (2.0/3.0*pi*pow(sigma,3)*epsilon*rho);
}

double lj9(double x, double sigma, double epsilon)
{
    return (2.0/15.0*pow(sigma/x,9));
}

double lj3(double x, double sigma, double epsilon)
{
    return (pow(sigma/x,3));
}

double dlj9(double x, double sigma, double epsilon)
{
    return (2.0/15.0*pow(sigma,9)*pow(x,-10)*-9);
}

double dlj3(double x, double sigma, double epsilon)
{
    return (pow(sigma,3)*pow(x,-4)*-3);
}

const double ljshift()
{
    return (2.0/3.0*sqrt(5.0/2.0));
}

double ljcut(double sigma)
{
    double base=2.0/5.0;
    double exponent=1.0/6.0;
    return (pow(base,exponent)*sigma);
}

/*table begins with a keyword
than a line with keyword/parameters
than index,r,energy,force
until the end of that keyword
than a new keyword can be started*/

int main()
{
    //completely converted to Kcal/mol
    const double shift=ljshift();
    const int n=3000;
    ofstream table;
table.open("pmma_alumina.table");
table << "ch2/ch3\n"; //keyword
table << "N " << n << "\n\n"; //keyword-param
//potential parameters for ch2/ch3
double sigma=3.24, epsilon=1.28; //changed to Kcal/mol
double cut=ljcut(sigma);
//initial r value and the change in r
double r=.01;
double delta=(cut-r)/double(n-1);
//calculate and store table for ch2/ch3 energy and force values
for (int index=1; index<=n; index++){
    double energy;
    double force;
    if (r<=cut){
        energy=ljfront(r,sigma,epsilon)*
            (lj9(r,sigma,epsilon)-lj3(r,sigma,epsilon)
+shift);
        force=-1*ljfront(r,sigma,epsilon)*
            (dlj9(r,sigma,epsilon)-dlj3(r,sigma,epsilon));
    }
    else{
        energy=0.0;
        force=0.0;
    }
    table<< index<<" " << r << " " << energy <<" " " <<
    force<< " " << endl;
    r+=delta;
}
//begining new keyword [completely converted to kcal/mol]
table<< "\n";
table<<"carbon\n"; //keyword
table<<"N " << n << "\n\n"; //keyword-param
//potential parameters for carbon
sigma=2.88;
epsilon=1.74; //changed to Kcal/mol
cut=ljcut(sigma);
//initial r value and the change in r
r=.01;
delta=(cut-r)/double(n-1);
//calculate and store table for carbon energy and force values
for (int index=1; index<=n; index++){
    double energy;
    double force;
    if (r<=cut){
        energy=ljfront(r,sigma,epsilon)*
(lj9(r, sigma, epsilon) - lj3(r, sigma, epsilon) + shift); 

force = -1 * ljfront(r, sigma, epsilon) * (dlj9(r, sigma, epsilon) - dlj3(r, sigma, epsilon));
}
else{
    energy = 0.0;
    force = 0.0;
}

} else {
    energy = 0.0;
    force = 0.0;
}

table << index << " " << r << " " << energy << " " << force << " " << endl;
    r += delta;
}
//begining new keyword [completely converted to Kcal/mol]
table << "\n";
table << "c(carbonyl)\n"; //keyword

//potential parameters for c(carbonyl)
sigma = 2.88;
epsilon = 2.18; //changed to Kcal/mol
cut = ljcut(sigma);
//initial r value and the change in r
r = 0.01;
delta = (cut - r) / double(n - 1);
//calculate and store table for c(carbonyl) energy and force
values
for (int index = 1; index <= n; index++){
    double energy;
    double force;
    if (r <= cut){
        energy = ljfront(r, sigma, epsilon) * (lj9(r, sigma, epsilon) - lj3(r, sigma, epsilon) + shift);
        force = -1 * ljfront(r, sigma, epsilon) * (dlj9(r, sigma, epsilon) - dlj3(r, sigma, epsilon));
    }
    else{
        energy = 0.0;
        force = 0.0;
    }
    table << index << " " << r << " " << energy << " " << force << " " << endl;
    r += delta;
}
//begining new keyword [completely converted to Kcal/mol]
table << "\n";
table << "avg(C)\n"; //keyword
table<<"N " << n << "\n\n"; //keyword-param
//potential parameters for avg(C)
sigma=3;
epsilon=1.74; //changed to Kcal/mol
cut=ljcut(sigma);
//initial r value and the change in r
r=.01;
delta=(cut-r)/double(n-1);
//calculate and store table for c(carbonyl) energy and force values
for (int index=1;index<=n; index++){
    double energy;
    double force;
    if (r<=cut){
        energy=ljfront(r,sigma,epsilon)*
            (lj9(r,sigma,epsilon)-lj3(r,sigma,epsilon)+shift);
        force=-1*ljfront(r,sigma,epsilon)*
            (dlj9(r,sigma,epsilon)-dlj3(r,sigma,epsilon));
    }
    else{
        energy=0.0;
        force=0.0;
    }
    table<< index<<" " <<r << " "<< energy <<" " " <<
    force<< " " << endl;
    r+=delta;
}

//begining new keyword [completely converted to kcal/mol]
table<< "\n";
table<<"o(carbonyl)\n"; //keyword
table<<"N " << n << "\n\n"; //keyword-param
//potential parameters for o(carbonyl)
double B=34.95; //changed to kcal/mol
double C=21.36; //changed to kcal/mol
cut=4.36; //unchanged when changing to kcal/mol
//shift value
double shift_morse=a_equation(cut)-b_equation(cut,B)+c_equation(cut,C);
//initial r value and the change in r
r=.01;
delta=(cut-r)/double(n-1);
//calculate and store table for o(carbonyl) energy and force values
for (int index=1;index<=n; index++){
    double energy;
    double force;
    if (r<=cut){

energy = a_equation(r) - b_equation(r, B) + c_equation(r, C) - shift_morse;
force = -1.0 * (da_equation(r) - db_equation(r, B) + dc_equation(r, C));
}
else{
    energy = 0.0;
    force = 0.0;
}
table << index << " " << r << " " << energy << " " << force << " " << endl;
    r += delta;
}
//beginning new keyword [completely converted to kcal/mol]
table << "\n";
table << "o(ester)\n"; // keyword
table << "N " << n << " \n\n"; // keyword-param
// potential parameters for o(ester)
B = 37.52; // changed to kcal/mol
C = 20.79; // changed to kcal/mol
cut = 4.41; // unchanged when changing to kcal/mol
// shift value
shift_morse = a_equation(cut) - b_equation(cut, B) + c_equation(cut, C);
// initial r value and the change in r
r = 0.01;
delta = (cut - r) / double(n - 1);
// calculate and store table for o(ester) energy and force values
for (int index = 1; index <= n; index++){
    double energy;
    double force;
    if (r <= cut){
        energy = a_equation(r) - b_equation(r, B) + c_equation(r, C) - shift_morse;
        force = -1.0 * (da_equation(r) - db_equation(r, B) + dc_equation(r, C));
    }
    else{
        energy = 0.0;
        force = 0.0;
    }
table << index << " " << r << " " << energy << " " << force << " " << endl;
r += delta;
}
table.close();
return 0;
F.1 Initial Equilibration of PMMA Around Nano Particle Space

#!/usr/bin/env python
#
#
#
#
#
f=open('in.pmma','w')
f.write("# pmma chain

units       real
atom_style  full
bond_style  harmonic
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic

read_data   data.pmma85

pair_style  lj/cut/coul/cut 10 15
pair_modify  shift yes
pair_coeff   1 1  .2028  3.7888
pair_coeff   1 2  .1394  3.9090
pair_coeff   2 2  .09586 4.0330
pair_coeff   1 3  .00338  5.3295
pair_coeff   2 3  .002324 5.4985
pair_coeff   3 3  .000056338 7.4966
pair_coeff   1 4  .08889  3.8570
pair_coeff   2 4  .06111  3.9793
pair_coeff   3 4  .001481 5.4254
pair_coeff   4 4  .0401  3.9067
pair_coeff   1 5  .2257  3.2312
pair_coeff   2 5  .1551  3.3337
pair_coeff   3 5  .003762  4.5451
pair_coeff   4 5  .1086  3.2385
pair_coeff   5 5  .3361  2.6251
pair_coeff   1 6  .2005  3.2632
pair_coeff   2 6  .1379  3.3666
pair_coeff   3 6  .003343  4.5900
pair_coeff   4 6  .09652  3.2705

164
pair_coeff 5 6 .2987 2.6511
pair_coeff 6 6 .2654 2.6773

group all id <= 6
region wall sphere 0.0 0.0 0.0 15 side out units box

velocity all create 300 376847 loop geom

neighbor 2 bin
neigh_modify every 1 delay 5 check yes one 5000

thermo 50
dump 2 all custom 50000 dump.pmma* type x y z
fix 1 all nve/limit .1
fix 3 all wall/region wall lj126 0.11749 4.00078 10
log log.pmma85initeq
run 10000
write_restart pmma85.initeq

....)
f.close()

import os
os.system('mpirun -np 4 ../../../lammps/src/lmp_openmpi<in.pmma')

F.2 Final Equilibration of PMMA Around Nano Particle Space

#!/usr/bin/env python
#
#
#
#
#
# f=open('in.pmma','w')
f.write('"""# pmma chain

units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic
pair_style lj/cut/coul/cut 10 15
pair_modify shift yes
read_restart pmma85.initeq

reset_timestep 0
group all id <= 6
region wall sphere 0.0 0.0 0.0 15 side out units box
velocity   all create 300 376847 loop geom

neighbor    2 bin
neigh_modify every 1 delay 5 check yes

timestep 4
run_style respa 2 2

thermo      50
dump        2 all custom 50000 dump.pmma* type x y z
fix         1 all nvt temp 300 600 100
fix         3 all wall/region wall lj126 0.11749 4.00078 10
log         log.pmma85finaleq1

run 12500
write_restart pmma85.finaleq1

unfix       1
fix         1 all nvt temp 600 600 100
log         log.pmma85finaleq2

run 25000
write_restart pmma85.finaleq2

unfix       1
fix         1 all nvt temp 600 298 100
log         log.pmma85finaleq3

run 12500
write_restart pmma85.finaleq3

"

import os
os.system('mpirun -np 4 .././lammps/src/lmp_openmpi<in.pmma')

F.3 Creating the Alumina Nano Particle

program alumina_nano_particle
C the distance for the sphere atoms is greater than sqrt(65)
implicit none
integer atomnum, unitnum
parameter (unitnum=729, atomnum=30)
c unitnum is the number of alumina PB crystals
c atomnumber is the number of atoms in the alumina PB crystal
double precision pbcrystal(5,atomnum),
atoms(5,atomnum*unitnum)
double precision hspherer
integer moleculenum
parameter (hspherer=15, moleculenum=20)
C hspherer is in angstrom.
C hspherer is the radius of the nano particle
C moleculenum is the molecule number assigned to the nano
particle
double precision a(3), b(3), c(3)
C The crystal axis in terms of the orthonormal coordinate system
integer i, j, k,l,m,n
C Represent the individual PB crystal unit: i is for the a(x) axis, j is for the b(y) axis,
C k is for the c(z) axis.
C i will have values between 4 and -4
C j will have values between 4 and -4
C k will have values between 4 and -4
integer count
C count will correspond to the atomnumber.
double precision dist
C Function to calculate distance between atoms and the origin (0,0,0)

C pbcrystal in A,B,C coordinate system
C 12 aluminum atoms
data pbcrystal/7, 1.4175, 0.0000, 0.0000, 0.3523,
& 7, 1.4175, 0.6667, 0.3333, 0.6856,
& 7, 1.4175, 0.3333, 0.6667, 0.0190,
& 7, 1.4175, 0.0000, 0.0000, 0.1477,
& 7, 1.4175, 0.6667, 0.3333, 0.4810,
C 5 total atoms
& 7, 1.4175, 0.3333, 0.6667, 0.8144,
& 7, 1.4175, 0.0000, 0.0000, 0.6477,
& 7, 1.4175, 0.6667, 0.3333, 0.9810,
& 7, 1.4175, 0.3333, 0.6667, 0.3144,
& 7, 1.4175, 0.0000, 0.0000, 0.8523,
C 10 total atoms
& 7, 1.4175, 0.6667, 0.3333, 0.1856,
& 7, 1.4175, 0.3333, 0.6667, 0.5190,
C 18 oxygen atoms
& 8, -.9450, 0.3064, 0.0000, 0.2500,
& 8, -.9450, 0.9731, 0.3333, 0.5833,
& 8, -.9450, 0.6397, 0.6667, 0.9167,
C 15 total atoms
& 8, -.9450, 0.0000, 0.3064, 0.2500,
& 8, -.9450, 0.6667, 0.6397, 0.5833,
& 8, -.9450, 0.3333, 0.9731, 0.9167,
& 8, -.9450, 0.3603, 0.0269, 0.5833, 
C 20 total atoms
& 8, -.9450, 0.0269, 0.3603, 0.9167, 
& 8, -.9450, 0.6936, 0.0000, 0.7500, 
& 8, -.9450, 0.3603, 0.3333, 0.0833, 
& 8, -.9450, 0.0269, 0.6667, 0.4167, 
& 8, -.9450, 0.0000, 0.6936, 0.7500,
C 25 total atoms
& 8, -.9450, 0.6667, 0.0269, 0.0833, 
& 8, -.9450, 0.3333, 0.3603, 0.4167, 
& 8, -.9450, 0.3064, 0.3064, 0.7500, 
& 8, -.9450, 0.9731, 0.6397, 0.0833, 
& 8, -.9450, 0.6397, 0.9731, 0.4167/
C 30 total atoms
C Defining crystal axis
& a/4.1170, 2.3770, 0.0000/, 
& b/0.0000, -4.7540, 0.0000/, 
& c/0.0000, 0.0000, -12.9900/
C write atom data in this file
open (25, file='alumina.pmma85')
c fill in atoms using pbcrystal, i, j, k, a, b and c
count = 1
do 10 i=-4,4
   do 20 j=-4,4
      do 30 k=-4,4
         do 60 l=1,atomnum
            do 70
               m=1,5
               if (m >= 3) then
                  atoms(m,count)= (pbcrystal(3,l)+i)*a(m-2) 
                                  +(pbcrystal(4,l))
                                  +(pbcrystal(5,l))
                  else
                     atoms(m,count)= pbcrystal(m,l)
                   endif
            70 continue
            count = count + 1
         60 continue
      30 continue
   20 continue
10 continue
C go through atoms.
c write the atoms that are within hspherer (note hsperer is centered at 0,0,0)
c include the atom number and molecule number within the written information
! count = 1
! do 40 l=1,atomnum*unitnum
! if (dist(atoms,l) .le. hspherer) then
! write(25,*) count, moleculenum, int(atoms(1,l)),
! atoms(2,l), atoms(3,l),
! & atoms(4,l), atoms(5,l), 0, 0, 0
! count = count + 1
! endif
40 continue
end

double precision function dist(atoms, l)
double precision atoms(5,*)
integer l
dist = sqrt(atoms(3,l)**2+atoms(4,l)**2+atoms(5,l)**2)
return
end

F.4 Inserting the Alumina Nano Particle

# npinsert.py
# For inserting a nanoparticle
#
# Created by Zachary Kraus on 6/21/12.
# Copyright (c) 2012 Georgia Tech. All rights reserved.
#
import lmpsdatarom pylab import *

# Read in the alumina nanoparticle
f=open('alumina.pmma85','r')
n nanoparticle=[]
for line in f:
 row=line.split()
 nanoparticle.append(row)

# Read in the polymer datafile
data=lmpsdata.Lmpsdata('pmma85data.finaleq3','full')

# Add in the mass information for the atom type 7, 8, and 9
# 7 is aluminum cation and 8 is oxgen anion and 9 is the surface aluminum cation
massinfo=[[7,'26.981539'],[8,'15.9994']]
data.adddata(massinfo,'Masses')
# Add in the alumina nanoparticle
data.addatoms(nanoparticle,False)

# Delete the data for the velocities and pair coefs
data.deletebodydata('Velocities')
data.deletebodydata('Pair Coefs')

# Delete Velocities and Pair Coefs from the keywords
data.keywords.remove('Velocities')
data.keywords.remove('Pair Coefs')

# Write the density calculations to the file testden.txt
r,rho=data.density(.5,0,32.5)
plot(r,rho)
xlabel('distance (angstrom)')
ylabel('density (amu/angstrom^3)')
#legend(loc=1)
show()

# Write the new nanocomposite into a new datafile
data.write('pmma85_nano_composite_data.initial',1)

F.5 Preparing PMMA for Bonding

#!/usr/bin/env python
#
# need to go back and change the cutoff for o(carbonyl) and o(ester)
# finish
#
f=open('in.pmma','w')
f.write("# pmma chain

units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic

read_data pmma85_nano_composite_data.initial

pair_style hybrid/overlay lj/cut/coul/cut 10 15 table linear 3000
coul/cut 15
pair_modify shift yes
pair_coeff 1 1 lj/cut/coul/cut .2028 3.7888

170
<table>
<thead>
<tr>
<th>Pair</th>
<th>Interaction</th>
<th>Lennard-Jones (ε, σ)</th>
<th>Coulomb (ε, σ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>lj/cut/coul/cut</td>
<td>1.394, 3.9090</td>
<td></td>
</tr>
<tr>
<td>2 2</td>
<td>lj/cut/coul/cut</td>
<td>0.9586, 4.0330</td>
<td></td>
</tr>
<tr>
<td>1 3</td>
<td>lj/cut/coul/cut</td>
<td>0.00338, 5.3295</td>
<td></td>
</tr>
<tr>
<td>2 3</td>
<td>lj/cut/coul/cut</td>
<td>0.02324, 5.4985</td>
<td></td>
</tr>
<tr>
<td>1 4</td>
<td>lj/cut/coul/cut</td>
<td>0.08889, 3.8570</td>
<td></td>
</tr>
<tr>
<td>2 4</td>
<td>lj/cut/coul/cut</td>
<td>0.06111, 3.9793</td>
<td></td>
</tr>
<tr>
<td>1 5</td>
<td>lj/cut/coul/cut</td>
<td>0.001481, 5.4254</td>
<td></td>
</tr>
<tr>
<td>2 5</td>
<td>lj/cut/coul/cut</td>
<td>0.0401, 3.9067</td>
<td></td>
</tr>
<tr>
<td>2 3</td>
<td>lj/cut/coul/cut</td>
<td>0.1551, 3.2312</td>
<td></td>
</tr>
<tr>
<td>3 5</td>
<td>lj/cut/coul/cut</td>
<td>0.003762, 4.5451</td>
<td></td>
</tr>
<tr>
<td>1 6</td>
<td>lj/cut/coul/cut</td>
<td>0.2005, 3.2632</td>
<td></td>
</tr>
<tr>
<td>2 6</td>
<td>lj/cut/coul/cut</td>
<td>0.1379, 3.3666</td>
<td></td>
</tr>
<tr>
<td>3 6</td>
<td>lj/cut/coul/cut</td>
<td>0.003343, 4.5900</td>
<td></td>
</tr>
<tr>
<td>4 6</td>
<td>lj/cut/coul/cut</td>
<td>0.09652, 3.2705</td>
<td></td>
</tr>
<tr>
<td>5 6</td>
<td>lj/cut/coul/cut</td>
<td>0.2987, 2.6511</td>
<td></td>
</tr>
<tr>
<td>6 6</td>
<td>lj/cut/coul/cut</td>
<td>0.2654, 2.6773</td>
<td></td>
</tr>
<tr>
<td>1 7</td>
<td>ch2/ch3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 7</td>
<td>ch2/ch3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 7</td>
<td>carbon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 7</td>
<td>c(carbonyl)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 7</td>
<td>o(carbonyl)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 7</td>
<td>o(ester)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#taken out for purposes of allowing restart to work

<table>
<thead>
<tr>
<th>Pair</th>
<th>Interaction</th>
<th>Lennard-Jones (ε, σ)</th>
<th>Coulomb (ε, σ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 8</td>
<td>avg(C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 8</td>
<td>avg(C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 8</td>
<td>avg(C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 8</td>
<td>avg(C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 8</td>
<td>avg(C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 8</td>
<td>avg(C)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#taken out for purposes of allowing restart to work

#taken out for purposes of allowing restart to work

171
pair_coeff 8 8 coul/cut

group    polymer type <= 6
group    alumina type >= 7
velocity all create 300 376847 loop geom
velocity alumina set 0 0 0 units box

neighbor 2 bin
neigh_modify every 1 delay 5 check yes

thermo 50
fix 1 polymer nve/limit .1
log log.pmma85composite_initeq
run 5000
write_restart pmma85composite.initeq

""")
f.close()

import os
os.system('mpirun -np 4 ../../lammps/src/lmp_openmpi<in.pmma')

F.6 Converting Restart File to Data File for Bonding

# npinsert.py
# For inserting a nanoparticle
# Created by Zachary Kraus on 6/21/12.
# Copyright (c) 2012 Georgia Tech. All rights reserved.
#
import lmpsdata
from pylab import *
def converttodata(restartfile,datafile):
    import os
    os.system('../../lammps/tools/restart2data ' + restartfile + ' ' + datafile)

# Convert restart file to datafile
converttodata('pmma85composite.initeq', 'pmma85compositedata.initeq')

# Read in the polymer datafile
data=lmpsdata.Lmpsdata('pmma85compositedata.initeq','full')

# Write the density calculations to the file testden.txt
r,rho=data.density(.5,0,32.5)
plot(r,rho)
xlabel('distance (angstrom)')
ylabel('density (amu/angstrom^3)')
#xticks(arange(0,21,1))
#legend(loc=1)
show()

# write out an xyz file
data.createxyz('pmma85composite_initeq.xyz')

# split into molecules and write out xyz files
#pmma=lmpsdata.molecules(data,1,4,'atom')
#for i in range(len(pmma)):
# pmma[i].createxyz('pmma80_molecule'+str(i)+'_initeq.xyz',data)
#alumina=lmpsdata.molecules(data,5,5,'atom')
#alumina[0].createxyz('alumina_initeq.xyz',data)

**F.7 Bonding PMMA to the Alumina Nano Particle**

# need to edit this one and pmma85 version of this as well. Need to make charges are correct for added atoms.
# pmma_alumina_bonding.py
#
# Created by Zachary Kraus on 6/21/12.
# Copyright (c) 2012 Georgia Tech. All rights reserved.
#
import lmpsdata, copy
data=lmpsdata.Lmpsdata('pmma85compositedata.initeq','full')

#copies pmma80compositedata.initeq to a new folder
directory='./bulk/'
data.write(directory+'pmma85_composite_data.initial', 0)

#orders atomdata before doing bonding procedure.
data.atomorder()

#seperate nanocomposite atoms into polymer and nanoparticle portion
# Note:Only the atom structures will be present in these variables.
polymer=lmpsdata.molecules(data,1,19)
nanoparticle=lmpsdata.molecules(data,20,20,'atom')

#seperate nanoparticle into its particle surface
surface=lmpsdata.particlesurface(nanoparticle[0], 1.94, 8, 'full')
surface.createxyz('alumina_surface_initeq.xyz',data)

#setup copies of molecules for different reaction processes
crosslink=copy.deepcopy(polymer)  #crosslink case is 2 bonds formed
weakbond=copy.deepcopy(polymer)   #weakbond case is 4 bonds formed
strongbond=copy.deepcopy(polymer) #strongbonds case is 8 bonds formed

#setup copies of surface for different reaction processes
surface_crosslink=copy.deepcopy(surface)
surface_weakbond=copy.deepcopy(surface)
surface_strongbond=copy.deepcopy(surface)

#add mass data for the bonded carbonyl type
massinfo=[[9,15.9994]]
data.adddata(massinfo,'Masses')

#setup copies of data for different reaction processes
data_crosslink=copy.deepcopy(data)
data_weakbond=copy.deepcopy(data)
data_strongbond=copy.deepcopy(data)

#from ssh_functions import *
#creating ssh connection
#host='71.204.28.16'
#username='zach'
#password='sweetcowfart3'
#ssh=create_ssh(host,username,password)

#create sftp connection
#sftp=create_sftp(ssh)

#setup the directory to send files to on the other computer
#otra_dir='/home/zach/Desktop/pmma 2nd step/PMMA80/'
#sftp.chdir(otra_dir)

#find possible bonding between crosslink(polymer) and
surface_crosslink
bondinglen=0
for molecule in crosslink:
molecule.findparticlebondingpoints(surface_crosslink,6,3,2,2)
#need to ensure the number of bonds are in multiples of 2
if len(molecule.bondinginformation)!=int(len(molecule.bondinginformation)/2.0)*2:
i=len(molecule.bondinginformation)-1
del molecule.bondinginformation[i]

print 'the length of the bonding information is',
len(molecule.bondinginformation)

bondinglen+=len(molecule.bondinginformation)
bondinglen=bondinglen/float(len(crosslink))

# Bond crosslink (polymer) to surface_crosslink

count=1
for molecule in crosslink:
    print 'the iteration is', count
    molecule.bondtoparticle(surface_crosslink, 1, '9', '-.7825')
    # add 1 atom to surface_crosslink per 2 atoms bonded
    for j in range(len(molecule.bondinginformation)/2):
        surface_crosslink.addatom(8, -.945, 7)
    count+=1

# extract the crosslink surface
surface_crosslink.extractparticle()

# extract molecule information to data_crosslink
data_crosslink.extractmolecules(crosslink)

# add in the crosslinked nanoparticle to data_crosslink
data_crosslink.addatoms(surface_crosslink.particle)

# delete the data for the velocities and pair coeffs from
data_crosslink
data_crosslink.deletebodydata('Velocities')

# delete the Velocities and Pair Coeffs from the keywords from
data_crosslink
data_crosslink.keywords.remove('Velocities')

# write the crosslink bonded nanocomposite into a new data file
directory='./crosslink/
data_crosslink.write(directory+'pmma85_composite_data.initial',1)
# write the crosslink bondinglen into a file in the crosslink directory
f=open(directory+'bondinglen.info','w')
f.write('{0}'.format(bondinglen))
f.close()

# send the crosslink bonded nanocomposite data file by sftp
# sftp.put(directory+'pmma80_composite_data.initial',directory +'pmma80_composite_data.initial')
# send the crosslink bondinglen info file by sftp
# sftp.put(directory+'bondinglen.info',directory +'bondinglen.info')
#find possible bonding between weakbond(polymer) and surface_weakbond
bondinglen=0
for molecule in weakbond:
molecule.findparticlebondingpoints(surface_weakbond,6,3.2,4)
#need to ensure the number of bonds are in multiples of 2
if len(molecule.bondinginformation)!=int(len(molecule.bondinginformation)/2.0)*2:
i=len(molecule.bondinginformation)-1
del molecule.bondinginformation[i]
print 'the length of the bonding information is',
len(molecule.bondinginformation)
bondinglen+=len(molecule.bondinginformation)
bondinglen=bondinglen/float(len(weakbond))

#Bond weakbond(polymer) to surface_weakbond
count=1
for molecule in weakbond:
print 'the iteration is', count
molecule.bondtoparticle(surface_weakbond,1,'9','-.7825')
#add 1 atom to surface_crosslink per 2 atoms bonded
for j in range(len(molecule.bondinginformation)/2):
surface_weakbond.addatom(8,-.945,7)
count+=1

#extract the weakbond surface
surface_weakbond.extractparticle()

# extract molecule information to data_weakbond
data_weakbond.extractmolecules(weakbond)

# add in the weakbonded nanoparticle to data_weakbond
data_weakbond.addatoms(surface_weakbond.particle)

# delete the data for the velocities and pair coeffs from data_weakbond
data_weakbond.deletebodydata('Velocities')

# delete the Velocities and Pair Coeffs from the keywords from data_weakbond
data_weakbond.keywords.remove('Velocities')

# write the weakbond bonded nanocomposite into a new data file
directory='./weakbond/'
data_weakbond.write(directory+'pmma85_composite_data.initial',1)
# write the weakbond bondinglen into a file in the crosslink directory
f=open(directory+'bondinglen.info','w')
f.write('{0}'.format(bondinglen))
f.close()

# send the crosslink bonded nanocomposite data file by sftp
#sftp.put(directory+'pmma80_composite_data.initial',directory + 'pmma80_composite_data.initial')
# send the crosslink bondinglen info file by sftp
#sftp.put(directory+'bondinglen.info',directory + 'bondinglen.info')

# find possible bonding between strongbond(polymer) and surface_strongbond
bondinglen=0
for molecule in strongbond:
    molecule.findparticlebondingpoints(surface_strongbond, 6,3.2,8)
    # need to ensure the number of bonds are in multiples of 2
    if len(molecule.bondinginformation)%2==0:
        i=len(molecule.bondinginformation)-1
        del molecule.bondinginformation[i]
    print 'the length of the bonding information is',
    len(molecule.bondinginformation)
    bondinglen+=len(molecule.bondinginformation)
bondinglen=bondinglen/float(len(strongbond))

# Bond strongbond(polymer) to surface_strongbond
count=1
for molecule in strongbond:
    print 'the iteration is', count
    molecule.bondtoparticle(surface_strongbond,1,'9','-.7825')
    # add 1 atom to surface_crosslink per 2 atoms bonded
    for j in range(len(molecule.bondinginformation)/2):
        surface_strongbond.addatom(8,-.945,7)
    count+=1

# extract the strongbond surface
surface_strongbond.extractparticle()

# extract molecule informtaion to data_strongbond
data_strongbond.extractmolecules(strongbond)

# add in the strongboded nanoparticle to data_strongbond
data_strongbond.addatoms(surface_strongbond.particle)
# delete the data for the velocities and pair coeffs from data_strongbond
data_strongbond.deletebodydata('Velocities')

# delete the Velocities and Pair Coeffs from the keywords from data_strongbond
data_strongbond.keywords.remove('Velocities')

# write the strongbond bonded nanocomposite into a new data file
directory='./strongbond/'
data_strongbond.write(directory+'pmma85_composite_data.initial', 1)
# write the strongbond bondinglen into a file in the crosslink directory
f=open(directory+'bondinglen.info','w')
f.write('{0}'.format(bondinglen))
f.close()

# send the crosslink bonded nanocomposite data file by sftp
#sftp.put(directory+'pmma80_composite_data.initial',directory+'pmma80_composite_data.initial')
# send the crosslink bondinglen info file by sftp
#sftp.put(directory+'bondinglen.info',directory+'bondinglen.info')

#closing sftp connection
#close_sftp(sftp)

#closing ssh connection
#close_ssh(ssh)

F.8 Non-Bonded Initial Equilibration of the Nano Composite

#!/usr/bin/env python
#
#
#
#
#
#f=open('in.pmma','w')
f.write("# pmma chain
units real
atom_style full
bond_style harmonic
angle_style harmonic

178
dihedral_style multi/harmonic
improper_style harmonic

read_data  pmma85_composite_data.initial

pair_style hybrid/overlay lj/cut/coul/cut 10 15 table linear 3000
coul/cut 15
pair_modify shift yes

pair_coeff 1 1 lj/cut/coul/cut .2028 3.7888
pair_coeff 1 2 lj/cut/coul/cut .1394 3.9090
pair_coeff 2 2 lj/cut/coul/cut .09586 4.0330
pair_coeff 1 3 lj/cut/coul/cut .00338 5.3295
pair_coeff 2 3 lj/cut/coul/cut .002324 5.4985
pair_coeff 3 3 lj/cut/coul/cut .000056338 7.4966
pair_coeff 1 4 lj/cut/coul/cut .08889 3.8570
pair_coeff 2 4 lj/cut/coul/cut .06111 3.9793
pair_coeff 3 4 lj/cut/coul/cut .001481 5.4254
pair_coeff 4 4 lj/cut/coul/cut .0401 3.9067
pair_coeff 1 5 lj/cut/coul/cut .2257 3.2312
pair_coeff 2 5 lj/cut/coul/cut .1551 3.3337
pair_coeff 3 5 lj/cut/coul/cut .003762 4.5451
pair_coeff 4 5 lj/cut/coul/cut .1086 3.2385
pair_coeff 5 5 lj/cut/coul/cut .3361 2.6251
pair_coeff 1 6 lj/cut/coul/cut .2005 3.2632
pair_coeff 2 6 lj/cut/coul/cut .1379 3.3666
pair_coeff 3 6 lj/cut/coul/cut .003343 4.5900
pair_coeff 4 6 lj/cut/coul/cut .09652 3.2705
pair_coeff 5 6 lj/cut/coul/cut .2987 2.6511
pair_coeff 6 6 lj/cut/coul/cut .2654 2.6773
pair_coeff 1 7 table pmma_alumina.table ch2/ch3
pair_coeff 1 7 coul/cut
pair_coeff 2 7 table pmma_alumina.table ch2/ch3
pair_coeff 2 7 coul/cut
pair_coeff 3 7 table pmma_alumina.table carbon
pair_coeff 3 7 coul/cut
pair_coeff 4 7 table pmma_alumina.table c(carbonyl)
pair_coeff 4 7 coul/cut
pair_coeff 5 7 table pmma_alumina.table o(carbonyl)
pair_coeff 5 7 coul/cut
pair_coeff 6 7 table pmma_alumina.table o(ester)
pair_coeff 6 7 coul/cut
# taking out so results can be converted to data
pair_coeff 7 7 coul/cut
pair_coeff 1 8 table pmma_alumina.table avg(C)
pair_coeff 1 8 coul/cut
pair_coeff 2 8 table pmma_alumina.table avg(C)
pair_coeff 2 8 coul/cut
pair_coeff 3 8 table pmma_alumina.table avg(C)
pair_coeff 3 8 coul/cut
pair_coeff 4 8 table pmma_alumina.table avg(C)
pair_coeff 4 8 coul/cut
pair_coeff 5 8 table pmma_alumina.table avg(C)
pair_coeff 5 8 coul/cut
pair_coeff 6 8 table pmma_alumina.table avg(C)
pair_coeff 6 8 coul/cut
# taking out so results can be converted to data
pair_coeff 7 8 coul/cut
# taking out so results can be converted to data
pair_coeff 8 8 coul/cut
group polymer type <= 6
group alumina type >= 7
velocity all create 300 376847 loop geom
velocity alumina set 0 0 0 units box
neighbor 2 bin
neigh_modify every 1 delay 5 check yes
thermo 50
fix 1 polymer nve/limit .1
log log.pmma85composite_initeq
run 5000
write_restart pmma85composite.initeq
"")
f.close()

import os
os.system('mpirun -np 4 ../../../lammps/src/lmp_openmpi<in.pmma')
f=open('simulated.txt','a')
f.write('initial equilibrium')
f.close()

F.9 Non-Bonded Final Equilibration of Nano Composite

#!/usr/bin/env python
#
#
#
#
#
#
f=open('in.pmma','w')
f.write("""# pmma chain

units real
atom_style full
bond_style harmonic

180
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic

read_restart pmma85composite.initeq

pair_modify shift yes
pair_coeff 1 1 lj/cut/coul/cut 0.2028 3.7888
pair_coeff 1 2 lj/cut/coul/cut 0.1394 3.9090
pair_coeff 2 2 lj/cut/coul/cut 0.09586 4.0330
pair_coeff 1 3 lj/cut/coul/cut 0.00338 5.3295
pair_coeff 2 3 lj/cut/coul/cut 0.002324 5.4985
pair_coeff 3 3 lj/cut/coul/cut 0.00056338 7.4966
pair_coeff 1 4 lj/cut/coul/cut 0.08889 3.8570
pair_coeff 2 4 lj/cut/coul/cut 0.06111 3.9793
pair_coeff 3 4 lj/cut/coul/cut 0.001481 5.4254
pair_coeff 4 4 lj/cut/coul/cut 0.0401 3.9067
pair_coeff 1 5 lj/cut/coul/cut 0.2257 3.2312
pair_coeff 2 5 lj/cut/coul/cut 0.1551 3.3337
pair_coeff 3 5 lj/cut/coul/cut 0.003762 4.5451
pair_coeff 4 5 lj/cut/coul/cut 0.106 3.2385
pair_coeff 5 5 lj/cut/coul/cut 0.3361 2.6251
pair_coeff 1 6 lj/cut/coul/cut 0.2005 3.2632
pair_coeff 2 6 lj/cut/coul/cut 0.1379 3.3666
pair_coeff 3 6 lj/cut/coul/cut 0.003343 4.5900
pair_coeff 4 6 lj/cut/coul/cut 0.09652 3.2705
pair_coeff 5 6 lj/cut/coul/cut 0.2987 2.6511
pair_coeff 6 6 lj/cut/coul/cut 0.2654 2.6773
pair_coeff 1 7 table pmma_alumina.table ch2/ch3
pair_coeff 1 7 coul/cut
pair_coeff 2 7 table pmma_alumina.table ch2/ch3
pair_coeff 2 7 coul/cut
pair_coeff 3 7 table pmma_alumina.table carbon
pair_coeff 3 7 coul/cut
pair_coeff 4 7 table pmma_alumina.table c(carbonyl)
pair_coeff 4 7 coul/cut
pair_coeff 5 7 table pmma_alumina.table o(carbonyl)
pair_coeff 5 7 coul/cut
pair_coeff 6 7 table pmma_alumina.table o(ester)
pair_coeff 6 7 coul/cut
# taking out so results can be converted to data
pair_coeff 7 7 coul/cut
pair_coeff 1 8 table pmma_alumina.table avg(C)
pair_coeff 1 8 coul/cut
pair_coeff 2 8 table pmma_alumina.table avg(C)
pair_coeff 2 8 coul/cut
pair_coeff 3 8 table pmma_alumina.table avg(C)
pair_coeff 3 8 coul/cut
pair_coeff 4 8 table pmma_alumina.table avg(C)

181
pair_coeff 4 8 coul/cut
pair_coeff 5 8 table pmma_alumina.table avg(C)
pair_coeff 5 8 coul/cut
pair_coeff 6 8 table pmma_alumina.table avg(C)
pair_coeff 6 8 coul/cut
# taking out so results can be converted to data
pair_coeff 7 8 coul/cut
# taking out so results can be converted to data
pair_coeff 8 8 coul/cut

reset_timestep 0
group polymer type <= 6
group alumina type >= 7
#velocity all create 300 376847 loop geom
#velocity alumina set 0 0 0 units box

neighbor 2 bin
neigh_modify every 1 delay 5 check yes
timestep 4
run_style respa 2 2

thermo 50
fix 1 polymer nvt temp 300 600 100
thermo_modify temp 1_temp
log log.pmma85composite_finaleq1

run 12500
write_restart pmma85composite.finaleq1

unfix 1
fix 1 polymer nvt temp 600 600 100
log log.pmma85composite_finaleq2

run 25000
write_restart pmma85composite.finaleq2

unfix 1
fix 1 polymer nvt temp 600 298 100
log log.pmma85composite_finaleq3

run 12500
write_restart pmma85composite.finaleq3

f.close()

import os

182
os.system('mpirun -np 4 ../../../lammps/src/lmp_openmpi<in.pmma')

f=open('simulated.txt','a')
f.write('\nfinal equilibrium')
f.close()

F.10 Non-Bonded NVT Simulation of Nano Composite

#! /usr/bin/env python
#
#
# need to redo this and change
#
#
f=open('in.pmma','w')
f.write("# pmma chain

units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic

read_restart pmma85composite.finaleq3

pair_modify shift yes
pair_coeff 1 1 lj/cut/coul/cut .2028 3.7888
pair_coeff 1 2 lj/cut/coul/cut .1394 3.9090
pair_coeff 2 2 lj/cut/coul/cut .09586 4.0330
pair_coeff 1 3 lj/cut/coul/cut .00338 5.3295
pair_coeff 2 3 lj/cut/coul/cut .002324 5.4985
pair_coeff 3 3 lj/cut/coul/cut .000056338 7.4966
pair_coeff 1 4 lj/cut/coul/cut .08889 3.8570
pair_coeff 2 4 lj/cut/coul/cut .06111 3.9793
pair_coeff 3 4 lj/cut/coul/cut .001481 5.4254
pair_coeff 4 4 lj/cut/coul/cut .0401 3.9067
pair_coeff 1 5 lj/cut/coul/cut .2257 3.2312
pair_coeff 2 5 lj/cut/coul/cut .1551 3.3337
pair_coeff 3 5 lj/cut/coul/cut .003762 4.5451
pair_coeff 4 5 lj/cut/coul/cut .1086 3.2385
pair_coeff 5 5 lj/cut/coul/cut .3361 2.6251
pair_coeff 1 6 lj/cut/coul/cut .2005 3.2632
pair_coeff 2 6 lj/cut/coul/cut .1379 3.3666
pair_coeff 3 6 lj/cut/coul/cut .003343 4.5900
pair_coeff 4 6 lj/cut/coul/cut .09652 3.2705
pair_coeff 5 6 lj/cut/coul/cut .2987 2.6511
pair_coeff 6 6 lj/cut/coul/cut .2654 2.6773
pair_coeff 1 7 table pmma_alumina.table ch2/ch3
pair_coeff 1 7 coul/cut
pair_coeff 2 7 table pmma_alumina.table ch2/ch3
pair_coeff 2 7 coul/cut
pair_coeff 3 7 table pmma_alumina.table carbon
pair_coeff 3 7 coul/cut
pair_coeff 4 7 table pmma_alumina.table c(carbonyl)
pair_coeff 4 7 coul/cut
pair_coeff 5 7 table pmma_alumina.table o(carbonyl)
pair_coeff 5 7 coul/cut
pair_coeff 6 7 table pmma_alumina.table o(ester)
pair_coeff 6 7 coul/cut
# taking out so results can be converted to data
pair_coeff 7 7 coul/cut
pair_coeff 1 8 table pmma_alumina.table avg(C)
pair_coeff 1 8 coul/cut
pair_coeff 2 8 table pmma_alumina.table avg(C)
pair_coeff 2 8 coul/cut
pair_coeff 3 8 table pmma_alumina.table avg(C)
pair_coeff 3 8 coul/cut
pair_coeff 4 8 table pmma_alumina.table avg(C)
pair_coeff 4 8 coul/cut
pair_coeff 5 8 table pmma_alumina.table avg(C)
pair_coeff 5 8 coul/cut
pair_coeff 6 8 table pmma_alumina.table avg(C)
pair_coeff 6 8 coul/cut
# taking out so results can be converted to data
pair_coeff 7 8 coul/cut
# taking out so results can be converted to data
pair_coeff 8 8 coul/cut
reset_timestep 0
group polymer type <= 6
group alumina type >= 7
#velocity all create 300 376847 loop geom
#velocity alumina set 0 0 0 units box
neighbor 2 bin
neigh_modify every 1 delay 5 check yes
timestep 4
run_style respa 2 2
thermo 50
fix 1 polymer nvt temp 298 298 100
thermo_modify temp 1_temp
log log.pmma85composite_bulkeq
run 25000
write_restart pmma85.finaleq4
log

log.pmma85composite_densitycalc

run 1250
write_restart pmma85.finaleq5

run 1250
write_restart pmma85.finaleq6

run 1250
write_restart pmma85.finaleq7

run 1250
write_restart pmma85.finaleq8

run 1250
write_restart pmma85.finaleq9

run 1250
write_restart pmma85.finaleq10

run 1250
write_restart pmma85.finaleq11

run 1250
write_restart pmma85.finaleq12

run 1250
write_restart pmma85.finaleq13

run 1250
write_restart pmma85.finaleq14

run 1250
write_restart pmma85.finaleq15

run 1250
write_restart pmma85.finaleq16

run 1250
write_restart pmma85.finaleq17

run 1250
write_restart pmma85.finaleq18

run 1250
write_restart pmma85.finaleq19

run 1250
write_restart pmma85.finaleq20
run 1250
write_restart pmma85.finaleq21
run 1250
write_restart pmma85.finaleq22
run 1250
write_restart pmma85.finaleq23
run 1250
write_restart pmma85.finaleq24
f.close()
import os
os.system('mpirun -np 4 ../../../lammps/src/lmp_openmpi < in.pmma')
f=open('simulated.txt','a')
f.write('
bulk equilibrium')
f.close()

F.11 Non-Bonded Young’s Modulus of Nano Composite

#!/usr/bin/env python
#
# currently strained at 1% or .01 which is probably too high
# changing strain rate from 1e10 to 1e8 to better match what is
# in the literature
# will still strain sample through 1k timesteps giving strain
# of .0001 or .01%
# now testing a strain sample through 10k timesteps giving strain
# of .001 or .1%

def lammpsstresscalc():
    import os
    f = open('in.tensile', 'w')  # file for running tensile test
    f.write('units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic
read_restart pmma98.finaleq4
pair_style hybrid/overlay lj/cut/coul/cut 10 15 table linear 3000
coul/cut 15 born 10
pair_modify shift yes
pair_coeff  1 1 lj/cut/coul/cut .2028 3.7888
pair_coeff  1 2 lj/cut/coul/cut .1394 3.9090
pair_coeff  2 2 lj/cut/coul/cut .09586 4.0330
pair_coeff  1 3 lj/cut/coul/cut .00338 5.3295
pair_coeff  2 3 lj/cut/coul/cut .002324 5.4985
pair_coeff  3 3 lj/cut/coul/cut .000056338 7.4966
pair_coeff  1 4 lj/cut/coul/cut .08889 3.8570
pair_coeff  2 4 lj/cut/coul/cut .06111 3.9793
pair_coeff  3 4 lj/cut/coul/cut .001481 5.4254
pair_coeff  4 4 lj/cut/coul/cut .0401 3.9067
pair_coeff  1 5 lj/cut/coul/cut .2257 3.2312
pair_coeff  2 5 lj/cut/coul/cut .1551 3.3337
pair_coeff  3 5 lj/cut/coul/cut .003762 4.5451
pair_coeff  4 5 lj/cut/coul/cut .1086 3.2385
pair_coeff  5 5 lj/cut/coul/cut .3361 2.6251
pair_coeff  1 6 lj/cut/coul/cut .2005 3.2632
pair_coeff  2 6 lj/cut/coul/cut .1379 3.3666
pair_coeff  3 6 lj/cut/coul/cut .003343 4.5900
pair_coeff  4 6 lj/cut/coul/cut .09652 3.2705
pair_coeff  5 6 lj/cut/coul/cut .2987 2.6511
pair_coeff  6 6 lj/cut/coul/cut .2654 2.6773
pair_coeff  1 7 table pmma_alumina.table ch2/ch3
coul/cut
pair_coeff  2 7 table pmma_alumina.table ch2/ch3
coul/cut
pair_coeff  3 7 table pmma_alumina.table carbon
coul/cut
pair_coeff  4 7 table pmma_alumina.table c(carbonyl)
coul/cut
pair_coeff  5 7 table pmma_alumina.table o(carbonyl)
coul/cut
pair_coeff  6 7 table pmma_alumina.table o(ester)
coul/cut
pair_coeff  7 7 born .068 .068 1.5704 324.02304 0.0
coul/cut
pair_coeff  1 8 table pmma_alumina.table avg(C)
coul/cut
pair_coeff  2 8 table pmma_alumina.table avg(C)
coul/cut
pair_coeff  3 8 table pmma_alumina.table avg(C)
coul/cut
pair_coeff  4 8 table pmma_alumina.table avg(C)
coul/cut
pair_coeff  5 8 table pmma_alumina.table avg(C)
coul/cut
pair_coeff  6 8 table pmma_alumina.table avg(C)
coul/cut
pair_coeff  7 8 born .172 .172 2.6067 797.38533 0.0
pair_coeff 7 8 coul/cut
pair_coeff 8 8 born .276 .276 3.643 1962.2782 0.0
pair_coeff 8 8 coul/cut

reset_timestep 0
group polymer type <= 6
group alumina type >= 7
velocity all create 300 376847 loop geom
#velocity alumina set 0 0 0 units box

neighbor 2 bin
neigh_modify every 1 delay 5 check yes
timestep 2
run_style verlet

thermo 50
#dump 2 all custom 100000 dump.pmma* type x y z
fix 1 all nvt temp 298 298 100
run 50000

unfix 1
fix 1 all npt temp 298 298 100 iso 0 0 100 drag 1
log log.pmma98composite_npt_init_eq
run 50000
write_restart pmma98composite.npt_init_eq

# store final cell length for strain calculations
variable tmp equal "lx"
variable LO equal {$tmp}
print "Initial Length, LO: ${LO}"

unfix 1
fix 1 all npt temp 298 298 100 y 0 0 100 z 0 0 100
variable srate equal 1e8
variable srate1 equal "v_srate / 1.0e15"
fix 2 all deform 1 x erate ${srate1} units box remap x
variable strain equal "(lx-v_LO)/v_LO"
variable p1 equal "-pxx/9.8692316931"
thermo_style custom step temp v_strain v_p1 press
log log.pmma98composite_npt_strain
run 50000
write_restart pmma98composite.npt_strain
unfix 2
log log.pmma9composites_final_eq
run 50000
write_restart pmma98composite.npt_final_eq

```python
f.close()
os.system('mpirun -np 4 ../../../lammps/src/lmp_openmpi<in.tensile')

def stresscalc(logfile):
    youngmod=[]
f=open('{0}'.format(logfile),'r')
strain=[] #initialize temperature list
stress=[] # initialize volume List
for line in f:
    row=line.split()
    try:
        int(row[0]) # Checks to make sure line in file
is a data line and not a text line
    except ValueError:
        if row[0]=='Loop': break #ends data aquisition
from file
    else: continue # otherwise continues reading the file
    else: #acquires important data
        strain.append(float(row[2]))
        stress.append(float(row[3]))
    f.close()
    # try calculating youngmodulus using 1000 step data.
stepsize=50
guesseddmod=0.0
for steps in range(50000,0,-stepsize):
    i=steps/stepsize
guesseddmod+=stress[i-1]/strain[i-1] #guessed youngmod
num=float(50000/stepsize)
youngmod.append(guesseddmod/num)
return youngmod

def stressexam(logfile):
    f=open('{0}'.format(logfile),'r')
strain=[] #initialize temperature list
stress=[] # initialize stress list
for line in f:
    row=line.split()
    try:
        int(row[0]) # Checks to make sure line in file
is a data line and not a text line
    except ValueError:
```
if row[0]=='Loop': break  # ends data acquisition
from file
    else: continue  # otherwise
continues reading the file
    else: #acquires important data
        strain.append(float(row[2]))
        stress.append(float(row[3]))
    f.close()
return strain, stress

def datawrite(header,youngmod):
    f=open('modulusdata.txt','a')
    f.write('{0} {1}\n'.format(header, youngmod))
    f.close()
from pylab import *
lammpsstresscalc()
youngmod=stresscalc('log.pmma98composite_npt_final_eq')
strain,stress=stressexam('log.pmma98composite_npt_strain')
datawrite('pmma98composite',youngmod)
#plot(strain,stress,'k-')
# graphing variables
#xlabel('strain')
#ylabel('''stress (MPa)'''')
#show()

f=open('simulated.txt','a')
f.write('\nbulk stress')
f.close()

F.12 Bonded Initial Equilibration of Nano Composite

#!/usr/bin/env python
#
# # need to go back and change the cutoff for o(carbonyl) and
o(ester)
#
f=open('in.pmma','w')
f.write('''# pmma chain

units real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic
''')
read_data  pmma85_composite_data.initial

pair_style hybrid/overlay lj/cut/coul/cut 10 15 table linear 3000
coul/cut 15
pair_modify shift yes
pair_coeff  1 1 lj/cut/coul/cut .2028 3.7888
pair_coeff  1 2 lj/cut/coul/cut .1394 3.9090
pair_coeff  2 2 lj/cut/coul/cut .09586 4.0330
pair_coeff  1 3 lj/cut/coul/cut .00338 5.3295
pair_coeff  2 3 lj/cut/coul/cut .002324 5.4985
pair_coeff  3 3 lj/cut/coul/cut .000056338 7.4966
pair_coeff  1 4 lj/cut/coul/cut .08889 3.8570
pair_coeff  2 4 lj/cut/coul/cut .06111 3.9793
pair_coeff  3 4 lj/cut/coul/cut .001481 5.4254
pair_coeff  4 4 lj/cut/coul/cut .0401 5.9067
pair_coeff  1 5 lj/cut/coul/cut .2257 3.2312
pair_coeff  2 5 lj/cut/coul/cut .1551 3.3337
pair_coeff  3 5 lj/cut/coul/cut .003762 4.5451
pair_coeff  4 5 lj/cut/coul/cut .1086 3.2385
pair_coeff  5 5 lj/cut/coul/cut .3361 2.6251
pair_coeff  1 6 lj/cut/coul/cut .2005 3.2632
pair_coeff  2 6 lj/cut/coul/cut .1379 3.3666
pair_coeff  3 6 lj/cut/coul/cut .003343 4.5900
pair_coeff  4 6 lj/cut/coul/cut .09652 3.2705
pair_coeff  5 6 lj/cut/coul/cut .2987 2.6511
pair_coeff  6 6 lj/cut/coul/cut .2654 2.6773
pair_coeff  1 7 table pmma_alumina.table ch2/ch3
pair_coeff  1 7 coul/cut
pair_coeff  2 7 table pmma_alumina.table ch2/ch3
pair_coeff  2 7 coul/cut
pair_coeff  3 7 table pmma_alumina.table carbon
pair_coeff  3 7 coul/cut
pair_coeff  4 7 table pmma_alumina.table c(carbonyl)
pair_coeff  4 7 coul/cut
pair_coeff  5 7 table pmma_alumina.table o(carbonyl)
pair_coeff  5 7 coul/cut
pair_coeff  6 7 table pmma_alumina.table o(ester)
pair_coeff  6 7 coul/cut
# taking out so results can be converted to data
pair_coeff  7 7 coul/cut
pair_coeff  1 8 table pmma_alumina.table avg(C)
pair_coeff  1 8 coul/cut
pair_coeff  2 8 table pmma_alumina.table avg(C)
pair_coeff  2 8 coul/cut
pair_coeff  3 8 table pmma_alumina.table avg(C)
pair_coeff  3 8 coul/cut
pair_coeff  4 8 table pmma_alumina.table avg(C)
pair_coeff  4 8 coul/cut
pair_coeff 5 8 table pmma_alumina.table avg(C)
pair_coeff 5 8 coul/cut
pair_coeff 6 8 table pmma_alumina.table avg(C)
pair_coeff 6 8 coul/cut
# taking out so results can be converted to data
pair_coeff 7 8 coul/cut
# taking out so results can be converted to data
pair_coeff 8 8 coul/cut
pair_coeff 1 9 table pmma_alumina.table avg(C)
pair_coeff 1 9 coul/cut
pair_coeff 2 9 table pmma_alumina.table avg(C)
pair_coeff 2 9 coul/cut
pair_coeff 3 9 table pmma_alumina.table avg(C)
pair_coeff 3 9 coul/cut
pair_coeff 4 9 table pmma_alumina.table avg(C)
pair_coeff 4 9 coul/cut
pair_coeff 5 9 table pmma_alumina.table avg(C)
pair_coeff 5 9 coul/cut
pair_coeff 6 9 table pmma_alumina.table avg(C)
pair_coeff 6 9 coul/cut
# taking out so results can be converted to data
pair_coeff 7 9 coul/cut
# taking out so results can be converted to data
pair_coeff 8 9 coul/cut
# taking out so results can be converted to data
pair coeff 9 9 coul/cut

group polymer type <= 6
group alumina type >= 7
velocity all create 300 376847 loop geom
velocity alumina set 0 0 0 units box

neighbor 2 bin
neigh_modify every 1 delay 5 check yes

thermo 50
fix 1 polymer nve/limit .1
log log.pmma85composite_initeq
run 5000
write_restart pmma85composite.initeq

"""
f.close()

import os
os.system('mpirun -n 4 ../../../lammps/src/lmp_openmpi<in.pmma')

f=open('simulated.txt', 'a')
f.write('initial equilibrium')
f.close()

F.13 Bonded Final Equilibration of Nano Composite

#!/usr/bin/env python
#
# f=open('in.pmma','w')
f.write('# pmma chain

units  real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic

read_restart  pmma85composite.initeq

pair_modify shift yes
pair_coeff 1 1 lj/cut/coul/cut .2028 3.7888
pair_coeff 1 2 lj/cut/coul/cut .1394 3.9090
pair_coeff 2 2 lj/cut/coul/cut .09586 4.0330
pair_coeff 1 3 lj/cut/coul/cut .00338 5.3295
pair_coeff 2 3 lj/cut/coul/cut .002324 5.4985
pair_coeff 3 3 lj/cut/coul/cut .000056338 7.4966
pair_coeff 1 4 lj/cut/coul/cut .08889 3.8570
pair_coeff 2 4 lj/cut/coul/cut .06111 3.9793
pair_coeff 3 4 lj/cut/coul/cut .001481 5.4254
pair_coeff 4 4 lj/cut/coul/cut .0401 3.9067
pair_coeff 1 5 lj/cut/coul/cut .2257 3.2312
pair_coeff 2 5 lj/cut/coul/cut .1551 3.3337
pair_coeff 3 5 lj/cut/coul/cut .003762 4.5451
pair_coeff 4 5 lj/cut/coul/cut .1086 3.2385
pair_coeff 5 5 lj/cut/coul/cut .3361 2.6251
pair_coeff 1 6 lj/cut/coul/cut .2005 3.2632
pair_coeff 2 6 lj/cut/coul/cut .1379 3.3666
pair_coeff 3 6 lj/cut/coul/cut .003343 4.5900
pair_coeff 4 6 lj/cut/coul/cut .09652 3.2705
pair_coeff 5 6 lj/cut/coul/cut .2987 2.6511
pair_coeff 6 6 lj/cut/coul/cut .2654 2.6773
pair_coeff 1 7 table pmma_alumina.table ch2/ch3
pair_coeff 1 7 coul/cut
pair_coeff 2 7 table pmma_alumina.table ch2/ch3
pair_coeff 2 7 coul/cut
pair_coeff 3 7 table pmma_alumina.table carbon
pair_coeff  3 7 coul/cut
pair_coeff  4 7 table pmma_alumina.table c(carbonyl)
pair_coeff  4 7 coul/cut
pair_coeff  5 7 table pmma_alumina.table o(carbonyl)
pair_coeff  5 7 coul/cut
pair_coeff  6 7 table pmma_alumina.table o(ester)
pair_coeff  6 7 coul/cut
# taking out so results can be converted to data
pair_coeff  7 7 coul/cut
pair_coeff  1 8 table pmma_alumina.table avg(C)
pair_coeff  1 8 coul/cut
pair_coeff  2 8 table pmma_alumina.table avg(C)
pair_coeff  2 8 coul/cut
pair_coeff  3 8 table pmma_alumina.table avg(C)
pair_coeff  3 8 coul/cut
pair_coeff  4 8 table pmma_alumina.table avg(C)
pair_coeff  4 8 coul/cut
pair_coeff  5 8 table pmma_alumina.table avg(C)
pair_coeff  5 8 coul/cut
pair_coeff  6 8 table pmma_alumina.table avg(C)
pair_coeff  6 8 coul/cut
# taking out so results can be converted to data
pair_coeff  7 8 coul/cut
# taking out so results can be converted to data
pair_coeff  8 8 coul/cut
pair_coeff  1 9 table pmma_alumina.table avg(C)
pair_coeff  1 9 coul/cut
pair_coeff  2 9 table pmma_alumina.table avg(C)
pair_coeff  2 9 coul/cut
pair_coeff  3 9 table pmma_alumina.table avg(C)
pair_coeff  3 9 coul/cut
pair_coeff  4 9 table pmma_alumina.table avg(C)
pair_coeff  4 9 coul/cut
pair_coeff  5 9 table pmma_alumina.table avg(C)
pair_coeff  5 9 coul/cut
pair_coeff  6 9 table pmma_alumina.table avg(C)
pair_coeff  6 9 coul/cut
# taking out so results can be converted to data
pair_coeff  7 9 coul/cut
# taking out so results can be converted to data
pair_coeff  8 9 coul/cut
# taking out so results can be converted to data
pair_coeff  9 9 coul/cut
reset_timestep 0

group polymer type <= 6
group alumina type >= 7

velocity all create 300 376847 loop geom
velocity    alumina set 0 0 0 units box
neighbor     2 bin
neigh_modify every 1 delay 5 check yes
timestep 4
run_style respa 2 2

thermo     50
fix        1 polymer nvt temp 300 600 100
log        log.pmma85composite_finaleq1

run 12500
write_restart pmma85composite.finaleq1

unfix      1
fix        1 polymer nvt temp 600 600 100
log        log.pmma85composite_finaleq2

run 25000
write_restart pmma85composite.finaleq2

unfix      1
fix        1 polymer nvt temp 600 298 100
log        log.pmma85composite_finaleq3

run 12500
write_restart pmma85composite.finaleq3

f.close()

import os
os.system('mpirun -np 4 ../../../lammps/src/lmp_openmpi<in.pmma')

f=open('simulated.txt','a')
f.write('final equilibrium')
f.close()

F.14 Bonded NVT Simulation of Nano Composite

#!/usr/bin/env python
#
# F.14 Bonded NVT Simulation of Nano Composite
units    real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic

read_restart pmma85composite.finaleq3

pair_modify shift yes
pair_coeff 1 1 lj/cut/coul/cut .2028 3.7888
pair_coeff 1 2 lj/cut/coul/cut .1394 3.9090
pair_coeff 2 2 lj/cut/coul/cut .09586 4.0330
pair_coeff 1 3 lj/cut/coul/cut .00338 5.3295
pair_coeff 2 3 lj/cut/coul/cut .00232 5.4985
pair_coeff 3 3 lj/cut/coul/cut .000056338 7.4966
pair_coeff 1 4 lj/cut/coul/cut .08889 3.8570
pair_coeff 2 4 lj/cut/coul/cut .06111 3.9793
pair_coeff 3 4 lj/cut/coul/cut .001481 5.4254
pair_coeff 4 4 lj/cut/coul/cut .0401 3.9067
pair_coeff 1 5 lj/cut/coul/cut .2257 3.2312
pair_coeff 2 5 lj/cut/coul/cut .1551 3.3337
pair_coeff 3 5 lj/cut/coul/cut .003762 4.5451
pair_coeff 4 5 lj/cut/coul/cut .1086 3.2385
pair_coeff 5 5 lj/cut/coul/cut .2987 2.6511
pair_coeff 1 6 lj/cut/coul/cut .2005 3.2632
pair_coeff 2 6 lj/cut/coul/cut .1379 3.3666
pair_coeff 3 6 lj/cut/coul/cut .003343 4.5900
pair_coeff 4 6 lj/cut/coul/cut .09652 3.2705
pair_coeff 5 6 lj/cut/coul/cut .2654 2.6611
pair_coeff 6 6 lj/cut/coul/cut .2654 2.6773
pair_coeff 1 7 table pmma_alumina.table ch2/ch3
pair_coeff 1 7 coul/cut
pair_coeff 2 7 table pmma_alumina.table ch2/ch3
pair_coeff 2 7 coul/cut
pair_coeff 3 7 table pmma_alumina.table carbon
pair_coeff 3 7 coul/cut
pair_coeff 4 7 table pmma_alumina.table c(carbonyl)
pair_coeff 4 7 coul/cut
pair_coeff 5 7 table pmma_alumina.table o(carbonyl)
pair_coeff 5 7 coul/cut
pair_coeff 6 7 table pmma_alumina.table o(ester)
pair_coeff 6 7 coul/cut
# taking out so results can be converted to data
pair_coeff 7 7 coul/cut
pair_coeff 1 8 table pmma_alumina.table avg(C)
pair_coeff 1 8 coul/cut
pair_coeff 2 8 table pmma_alumina.table avg(C)
pair_coeff 2 8 coul/cut
pair_coeff 3 8 table pmma_alumina.table avg(C)
pair_coeff 3 8 coul/cut
pair_coeff 4 8 table pmma_alumina.table avg(C)
pair_coeff 4 8 coul/cut
pair_coeff 5 8 table pmma_alumina.table avg(C)
pair_coeff 5 8 coul/cut
pair_coeff 6 8 table pmma_alumina.table avg(C)
pair_coeff 6 8 coul/cut
pair_coeff 7 8 coul/cut
pair_coeff 7 8 coul/cut
pair_coeff 8 8 coul/cut
pair_coeff 8 8 coul/cut
pair_coeff 9 8 coul/cut
pair_coeff 9 8 coul/cut
pair_coeff 1 9 table pmma_alumina.table avg(C)
pair_coeff 1 9 coul/cut
pair_coeff 2 9 table pmma_alumina.table avg(C)
pair_coeff 2 9 coul/cut
pair_coeff 3 9 table pmma_alumina.table avg(C)
pair_coeff 3 9 coul/cut
pair_coeff 4 9 table pmma_alumina.table avg(C)
pair_coeff 4 9 coul/cut
pair_coeff 5 9 table pmma_alumina.table avg(C)
pair_coeff 5 9 coul/cut
pair_coeff 6 9 table pmma_alumina.table avg(C)
pair_coeff 6 9 coul/cut
pair_coeff 7 9 coul/cut
pair_coeff 7 9 coul/cut
pair_coeff 8 9 coul/cut
pair_coeff 8 9 coul/cut
pair_coeff 9 9 coul/cut
pair_coeff 9 9 coul/cut

reset_timestep 0

group polymer type <= 6
group alumina type >= 7

#velocity all create 300 376847 loop geom
#velocity alumina set 0 0 0 units box

neighbor 2 bin
neigh_modify every 1 delay 5 check yes

timestep 4
run_style respa 2 2

thermo 50
fix 1 polymer nvt temp 298 298 100
log log.pmma85commposite_finaleq
run 25000
write_restart pmma85composite.finaleq4

log log.pmma85composite_densitycalc

run 1250
write_restart pmma85composite.finaleq5

run 1250
write_restart pmma85composite.finaleq6

run 1250
write_restart pmma85composite.finaleq7

run 1250
write_restart pmma85composite.finaleq8

run 1250
write_restart pmma85composite.finaleq9

run 1250
write_restart pmma85composite.finaleq10

run 1250
write_restart pmma85composite.finaleq11

run 1250
write_restart pmma85composite.finaleq12

run 1250
write_restart pmma85composite.finaleq13

run 1250
write_restart pmma85composite.finaleq14

run 1250
write_restart pmma85composite.finaleq15

run 1250
write_restart pmma85composite.finaleq16

run 1250
write_restart pmma85composite.finaleq17

run 1250
write_restart pmma85composite.finaleq18

run 1250
write_restart pmma85composite.finaleq19
run 1250
write_restart pmma85composite.finaleq20
run 1250
write_restart pmma85composite.finaleq21
run 1250
write_restart pmma85composite.finaleq22
run 1250
write_restart pmma85composite.finaleq23
run 1250
write_restart pmma85composite.finaleq24
```
```
f.close()
import os
os.system('mpirun -np 4 ../../../lammps/src/lmp_openmpi<in.pmma')
f=open('simulated.txt', 'a')
f.write('
bulk equilibrium')
f.close()

F.15 Bonded Young's Modulus of Nano Composite

#!/usr/bin/env python
#
# currently strained at 1% or .01 which is probably too high
# changing strain rate from 1e10 to 1e8 to better match what is
# in the literature
# will still strain sample through 1k timesteps giving strain
# of .0001 or .01%  
# now testing a strain sample through 10k timesteps giving strain
# of .01 or .1%

def lammpssstresscalc():  
    import os
    f=open('in.tensile','w') #file for running tensile test
    f.write('"""units  real
atom_style full
bond_style harmonic
angle_style harmonic
dihedral_style multi/harmonic
improper_style harmonic

read_data  pmma85composite_data_finaleq4
pair_style hybrid/overlay lj/cut/coul/cut 10 15 table
linear 3000 coul/cut 15 born 10
pair_modify shift yes
pair_coeff 1 1 lj/cut/coul/cut 0.2028 3.7888
pair_coeff 1 2 lj/cut/coul/cut 0.1394 3.9090
pair_coeff 2 2 lj/cut/coul/cut 0.09586 4.0330
pair_coeff 1 3 lj/cut/coul/cut 0.00338 5.3295
pair_coeff 2 3 lj/cut/coul/cut 0.002324 5.4985
pair_coeff 3 3 lj/cut/coul/cut 0.000056338 7.4966
pair_coeff 1 4 lj/cut/coul/cut 0.08889 3.8570
pair_coeff 2 4 lj/cut/coul/cut 0.06111 3.9793
pair_coeff 3 4 lj/cut/coul/cut 0.001481 5.4254
pair_coeff 4 4 lj/cut/coul/cut 0.0401 3.9067
pair_coeff 1 5 lj/cut/coul/cut 0.2257 3.2312
pair_coeff 2 5 lj/cut/coul/cut 0.1551 3.3337
pair_coeff 3 5 lj/cut/coul/cut 0.003762 4.5451
pair_coeff 4 5 lj/cut/coul/cut 0.1086 3.2385
pair_coeff 5 5 lj/cut/coul/cut 0.3361 2.6251
pair_coeff 1 6 lj/cut/coul/cut 0.2005 3.2632
pair_coeff 2 6 lj/cut/coul/cut 0.1379 3.3666
pair_coeff 3 6 lj/cut/coul/cut 0.003343 4.5900
pair_coeff 4 6 lj/cut/coul/cut 0.09652 3.2705
pair_coeff 5 6 lj/cut/coul/cut 0.2987 2.6511
pair_coeff 6 6 lj/cut/coul/cut 0.2654 2.6773
pair_coeff 1 7 table pmma_alumina.table ch2/ch3
pair_coeff 1 7 coul/cut
pair_coeff 2 7 table pmma_alumina.table ch2/ch3
pair_coeff 2 7 coul/cut
pair_coeff 3 7 table pmma_alumina.table carbon
pair_coeff 3 7 coul/cut
pair_coeff 4 7 table pmma_alumina.table c(carbonyl)
pair_coeff 4 7 coul/cut
pair_coeff 5 7 table pmma_alumina.table o(carbonyl)
pair_coeff 5 7 coul/cut
pair_coeff 6 7 table pmma_alumina.table o(ester)
pair_coeff 6 7 coul/cut
pair_coeff 7 7 born 0.068 0.068 1.5704 77.4434 0.0
pair_coeff 7 7 coul/cut
pair_coeff 1 8 table pmma_alumina.table avg(C)
pair_coeff 1 8 coul/cut
pair_coeff 2 8 table pmma_alumina.table avg(C)
pair_coeff 2 8 coul/cut
pair_coeff 3 8 table pmma_alumina.table avg(C)
pair_coeff 3 8 coul/cut
pair_coeff 4 8 table pmma_alumina.table avg(C)
pair_coeff 4 8 coul/cut
pair_coeff 5 8 table pmma_alumina.table avg(C)
pair_coeff 5 8 coul/cut
pair_coeff 6 8 table pmma_alumina.table avg(C)
pair_coeff 6 8 coul/cut
pair_coeff 7 8 born .172 .172 2.6067 190.5797 0.0
pair_coeff 7 8 coul/cut
pair_coeff 8 8 born .276 .276 3.643 468.9957 0.0
pair_coeff 8 8 coul/cut
pair_coeff 1 9 table pmma_alumina.table avg(C)
pair_coeff 1 9 coul/cut
pair_coeff 2 9 table pmma_alumina.table avg(C)
pair_coeff 2 9 coul/cut
pair_coeff 3 9 table pmma_alumina.table avg(C)
pair_coeff 3 9 coul/cut
pair_coeff 4 9 table pmma_alumina.table avg(C)
pair_coeff 4 9 coul/cut
pair_coeff 5 9 table pmma_alumina.table avg(C)
pair_coeff 5 9 coul/cut
pair_coeff 6 9 table pmma_alumina.table avg(C)
pair_coeff 6 9 coul/cut
pair_coeff 7 9 born .172 .172 2.6067 190.5797 0.0
pair_coeff 7 9 coul/cut
pair_coeff 8 9 born .276 .276 3.643 468.9957 0.0
pair_coeff 8 9 coul/cut
pair_coeff 9 9 born .276 .276 3.643 468.9957 0.0
pair_coeff 9 9 coul/cut

reset_timestep 0
begin group
polymer type <= 6
end group
begin group
alumina type >= 7
end group

# velocity all create 300 376847 loop geom
#velocity alumina set 0 0 0 units box

neighbor 2 bin
neigh_modify every 1 delay 5 check yes exclude group alumina

alumina

timestep .5
run_style verlet

thermo 50
set group alumina image 0 0 0

fix 1 polymer npt temp 298 298 100 iso 0 0 500 drag 1
fix 2 alumina rigid/nvt group 1 alumina temp 298 298 100 dilate all tparam 12 8 3
log log.pmma85composite_npt_init_eq

run 50000
write_restart pmma85composite.npt_init_eq

# store final cell length for strain calculations
variable tmp equal "lx"
variable L0 equal ${tmp}
print "Initial Length, L0: ${L0}"
unfix 1
unfix 2
fix 1 polymer npt temp 298 298 100 y 0 0 500 z 0 0 500
fix 2 alumina rigid/nvt group 1 alumina temp 298 298 100 dilate all tparam 12 8 3
variable srate equal 1e8
variable srate1 equal "v_srate / 1.0e15"
fix 3 all deform 1 x erate ${srate1} units box remap x
variable strain equal "(lx-v_L0)/v_L0"
variable p1 equal "-pxx/9.8692316931"
thermo_style custom step temp v_strain v_p1 press
log log.pmma85composite_npt_strain
run 200000
write_restart pmma85composite.npt_strain

unfix 3
log log.pmma85composite_npt_final_eq
run 50000
write_restart pmma85composite.npt_final_eq

f.close()

os.system('mpirun -np 4 ../../../lammps2/src/lmp_openmpi<in.tensile')
def stresscalc(logfile):
    youngmod=[]
    f=open('{0}'.format(logfile),'r')
    strain=[] #initialize temperature list
    stress=[] # initialize volume List
    for line in f:
        row=line.split()
        try:
            int(row[0]) # Checks to make sure line in file is a data line and not a text line
        except ValueError:
            if row[0]=='Loop': break #ends data aquisition from file
            else: continue # otherwise continues reading the file
else:  #acquires important data
    strain.append(float(row[2]))
    stress.append(float(row[3]))

f.close()

# try calculating youngmodulus using 1000 step data.

stepsize=50

guessedmod=0.0

for steps in range(50000,0,-stepsize):
    i=steps/stepsize
    guessedmod+=stress[i-1]/strain[i-1]  #guessed youngmod

num=float(50000/stepsize)

youngmod.append(guessedmod/num)

return youngmod

def stressexam(logfile):
    f=open('{0}'.format(logfile),'r')
    strain=[]  #initialize temperature list
    stress=[]  # initialize stress list

    for line in f:
        row=line.split()
        try:
            int(row[0])  # Checks to make sure line in file is a data line and not a text line
        except ValueError:
            if row[0] == 'Loop': break  #ends data aquisition from file
        else: continue  # otherwise continues reading the file
    else:  #acquires important data
        strain.append(float(row[2]))
        stress.append(float(row[3]))

f.close()

return strain,stress

def datawrite(header,youngmod):
    f=open('modulusdata.txt','a')
    f.write('{0} {1}
'.format(header, youngmod))

f.close()

#from pylab import *

lammppstrescalc()

youngmod=stresscalc('log.pmma85composite_npt_final_eq')

strain,stress=stressexam('log.pmma85composite_npt_strain')

datawrite('pmma85composite',youngmod)

#plot(strain,stress,'k-')

# graphing variables

#xlabel('strain')
#ylabel('""stress (GPa)""')

203
#show()

f=open('simulated.txt', 'a')
f.write('\nbulk stress')
f.close()
APPENDIX G

BONDING ALGORITHM

#
# lmpsdata.py
# For reading, writing and manipulating lammps data files
# For calculation of certain properties using lammps data files
# For creating VMD input text files using lammps data files
# All x,y,z calculations assume the information includes image flags

class Lmpsdata:
    def __init__(self,file,atomtype):
        """initiates lammps data structures""
        self.atomtype=atomtype
        self.keywords=[]
        self.atoms=[]
        self.angles=[]
        self.bonds=[]
        self.dihedrals=[]
        self.dipoles=[]
        self.impropers=[]
        self.masses=[]
        self.shapes=[]
        self.velocities=[]
        self.anglec=ef=[]
        self.bondc=ef=[]
        self.dihedralc=ef=[]
        self.improperc=ef=[]
        self.pairc=ef=[]
        self.read(file)

    def read(self,file):
        """Reads in lammps data file
        Skips the first line of the file (Comment line)
        First reads the header portion of the file
        (sectflag=1)
        blank lines are skipped
        header keywords delineate an assignment
        if no header keyword is found, body portion begins
        Second reads the body portion of the file (sectflag=2)
        first line of a section has only a keyword
        next line is skipped
        """
remaining lines contain values
a blank line signifies the end of that section
if no value is listed on a line than a keyword

must be used

File is read until the end
The keywords read in are stored in self.keywords

if file == '':
    print 'no file is given. Will have to build
keywords and class structures manually'

return

sectflag=1
f = open(file, 'r')
f.readline()
for line in f:
    row = line.split()
    if sectflag == 1:
        if len(row) == 0:
            #skip line the line is blank
            pass
        elif len(row) == 1:
            # Set Sectflag to 2, assume line is a

keyword
    sectflag=2
    checkkey=1 #ensures keyword will be
    checked in body portion
    keyword=row
    elif len(row) == 2:
        if row[1] == 'atoms':
            self.atomnum = row[0]
            self.keywords.append('atoms')
        elif row[1] == 'bonds':
            self.bondnum = row[0]
            self.keywords.append('bonds')
        elif row[1] == 'angles':
            self.anglenum = row[0]
            self.keywords.append('angles')
        elif row[1] == 'dihedrals':
            self.dihedralnum = row[0]
            self.keywords.append('dihedrals')
        elif row[1] == 'impropers':
            self.impropernum = row[0]
            self.keywords.append('impropers')
        else:
            # Set Sectflag to 2, assume line
is a keyword
    sectflag=2
be checked in body portion

```
checkkey=1 #ensures keyword will be checked in body portion

    keyword=row

elif len(row)==3:
        self.atomtypenum=row[0]
        self.keywords.append('atom
types')
    elif row[1]=='bond' and
        row[2]=='types':
        self.bondtypenum=row[0]
        self.keywords.append('bond
types')
    elif row[1]=='angle' and
        row[2]=='types':
        self.angletypenum=row[0]
        self.keywords.append('angle
types')
    elif row[1]=='dihedral' and
        row[2]=='types':
        self.dihedraltypenum=row[0]
        self.keywords.append('dihedral
types')
    elif row[1]=='improper' and
        row[2]=='types':
        self.impropertypenum=row[0]
        self.keywords.append('improper
types')
    else:
        # Set Sectflag to 2, assume line is a keyword
        sectflag=2
        checkkey=1 #ensures keyword will be checked in body portion

elif len(row)==4:
        self.xdimension=[row[0], row[1]]
        self.keywords.append('xlo xhi')
        self.ydimension=[row[0], row[1]]
        self.keywords.append('ylo yhi')
        self.zdimension=[row[0], row[1]]
        self.keywords.append('zlo zhi')
    else:
        # Set Sectflag to 2, assume line is a keyword
```
def parse(self, line):
    sectflag = 2
    checkkey = 1  # ensures keyword will be checked in body portion
    keyword = line
    elif len(row) == 5:
            self.extrabonds = row[0]
            self.keywords.append('extra bond per atom')
        else:
            sectflag = 2  # Set Sectflag to 2, assume line is a keyword
            checkkey = 1  # ensures keyword will be checked in body portion
            keyword = row
    elif len(row) == 6:
            self.tilt = [row[0], row[1], row[2]]
        else:
            self.keywords.append('xy xz yz')
    else:
        sectflag = 2  # Set Sectflag to 2, assume line is a keyword
        checkkey = 1  # ensures keyword will be checked in body portion
        keyword = row
    else:
        # set sectflag to 2, assume line is a keyword
        sectflag = 2
        checkkey = 1  # ensures keyword will be checked in body portion
        keyword = line
    elif sectflag == 2:
        if checkkey == 1:
            if len(keyword) == 1:
                if keyword[0] == 'Atoms' or keyword[0] == 'Velocities' or keyword[0] == 'Masses' or
                   keyword[0] == 'Shapes' or keyword[0] == 'Dipoles' or keyword[0] == 'Bonds' or
                   keyword[0] == 'Dihedrals' or keyword[0] == 'Impropers':
                    bodyflag = 1

        blanknum = 0
```python
self.keywords.append(keyword[0])
    checkkey=0
else:
    bodyflag=0
    checkkey=0
elif len(keyword)==2:
    if row[1]=='Coeffs' and
       (row[0]=='Pair' or row[0]=='Bond' or row[0]=='Angle' or
        row[0]=='Dihedral' or
        row[0]=='Improper'):  # class 2 force field keywords not included
        bodyflag=1

blanknum=0

self.keywords.append('{0}{1}'.format(keyword[0],keyword[1]))
    checkkey=0
else:
    bodyflag=0
    checkkey=0
else:
    # ignore line and set bodyflag=0
    bodyflag=0

checkkey=0

if bodyflag==0:  # bodyflag 0 means no body keyword has been found
    if len(row)==1:
        if row[0]=='Atoms' or
           row[0]=='Velocities' or row[0]=='Masses' or
           row[0]=='Shapes' or
           row[0]=='Dipoles' or row[0]=='Bonds' or
           row[0]=='Angles' or
           row[0]=='Dihedrals' or row[0]=='Impropers':
            bodyflag=1
            blanknum=0
            keyword=row

self.keywords.append(keyword[0])
    else:
        # ignore line
        pass
elif len(row)==2:
    if row[1]=='Coeffs' and
       (row[0]=='Pair' or row[0]=='Bond' or row[0]=='Angle' or
        row[0]=='Dihedral' or
        row[0]=='Improper'):  # class 2 force field keywords not included
        bodyflag=1
```

209
blanknum=0
keyword=row
self.keywords.append('{0}
{1}'.format(keyword[0],keyword[1]))
    else:
        #ignore line
        pass
    else:
        #ignore line
        pass
elif bodyflag==1:  #currently assumes 1 or more blank lines are between body data keywords
    if len(row)==0:
        blanknum+=1
    if blanknum>1:
        bodyflag=0
elif len(keyword)==1:
    if keyword[0]=='Atoms':
        try:
            int(row[0])
        except ValueError:
            keyword=row
            checkkey=1
        else:
            self.atoms.append(row)
    elif keyword[0]=='Velocities':
        try:
            int(row[0])
        except ValueError:
            keyword=row
            checkkey=1
        else:
            self.velocities.append(row)
    elif keyword[0]=='Masses':
        try:
            int(row[0])
        except ValueError:
            keyword=row
            checkkey=1
        else:
            self.masses.append(row)
    elif keyword[0]=='Shapes':
        try:
            int(row[0])
        except ValueError:
            keyword=row
        else:
checkkey=1
else:
    self.shapes.append(row)

elif keyword[0]=='Dipoles':
    try:
        int(row[0])
    except ValueError:
        keyword=row
    checkkey=1
else:
    self.dipoles.append(row)

elif keyword[0]=='Bonds':
    try:
        int(row[0])
    except ValueError:
        keyword=row
    checkkey=1
else:
    self.bonds.append(row)

elif keyword[0]=='Angles':
    try:
        int(row[0])
    except ValueError:
        keyword=row
    checkkey=1
else:
    self.angles.append(row)

elif keyword[0]=='Dihedrals':
    try:
        int(row[0])
    except ValueError:
        keyword=row
    checkkey=1
else:
    self.dihedrals.append(row)

elif keyword[0]=='Impropers':
    try:
        int(row[0])
    except ValueError:
        keyword=row
    checkkey=1
else:
    self.impropers.append(row)
else:
    # ignore line and change
    bodyflag = 0

elif len(keyword)==2:
    if keyword[0]=='Pair' and
        keyword[1]=='Coeffs':
        try:
            int(row[0])
        except ValueError:
            keyword=row
            checkkey=1
        else:
            self.paircoef.append(row)

elif keyword[0]=='Bond' and
    keyword[1]=='Coeffs':
        try:
            int(row[0])
        except ValueError:
            keyword=row
            checkkey=1
        else:
            self.bondcoef.append(row)

elif keyword[0]=='Angle' and
    keyword[1]=='Coeffs':
        try:
            int(row[0])
        except ValueError:
            keyword=row
            checkkey=1
        else:
            self.anglecoef.append(row)

elif keyword[0]=='Dihedral' and
    keyword[1]=='Coeffs':
        try:
            int(row[0])
        except ValueError:
            keyword=row
            checkkey=1
        else:
            self.dihedralcoef.append(row)

elif keyword[0]=='Improper' and
    keyword[1]=='Coeffs':
        try:
            int(row[0])
        except ValueError:
            keyword=row
            checkkey=1
        else:
except ValueError:
    keyword=\n    checkkey=1
else:
    self.impropercoef.append(row)
else:
    #egnore line and change bodyflag to 0
    bodyflag=0
else:
    #egnore line and change bodyflag to 0
    bodyflag=0
f.close()

def write(self,file,modflag):
    """Write lammps data files using the lammps keywords and lammpsdata structures.
    writes first line of the file as a Comment line
    if no modifications to any of the lammpsdata structures (modflag=0)
    Use Keywords to write lammpsdata structures
    if modifications to any of the lammpsdata structures (modflag=1)
    Key Lammpsdata structures like atom numbers, coefficient numbers
    need to be modified to match the other modified lammpsdata structures.
    This section will still use the keywords to write lammpsdata structures.
    For all modflags, the code will write data to the file until all of the
    keyword's data structures have been finished writing.
    The keywords are stored in self.keywords""
    f=open(file,'w')
    f.write('polymer data file\n')
    for row in self.keywords:
        if row=='atoms':
            if modflag==0:
                f.write('{}\n'.format(self.atomnum,row))
        elif modflag==1:
            f.write('{}\n'.format(len(self.atoms),row))
        elif row=='bonds':
            if modflag==0:
f.write('{0}
'.format(self.bondnum,row))
    elif modflag==1:
        f.write('{0}
'.format(len(self.bonds),row))
    elif row=='angles':
        if modflag==0:
            f.write('{0}
'.format(self.anglenum,row))
        elif modflag==1:
            f.write('{0}
'.format(len(self.angles),row))
    elif row=='dihedrals':
        if modflag==0:
            f.write('{0}
'.format(self.dihedralnum,row))
        elif modflag==1:
            f.write('{0}
'.format(len(self.dihedrals),row))
    elif row=='impropers':
        if modflag==0:
            f.write('{0}
'.format(self.impropernum,row))
        elif modflag==1:
            f.write('{0}
'.format(len(self.impropers),row))
    elif row=='atom types':
        if modflag==0:
            f.write('{0}
'.format(self.atomtypenum,row))
        elif modflag==1:
            f.write('{0}
'.format(len(self.masses),row))
    elif row=='bond types':
        if modflag==0:
            f.write('{0}
'.format(self.bondtypenum,row))
        elif modflag==1:
            f.write('{0}
'.format(len(self.bondcoef),row))
    elif row=='angle types':
        if modflag==0:
            f.write('{0}
'.format(self.angletypenum,row))
        elif modflag==1:
            f.write('{0}
'.format(len(self.anglecoef),row))
    elif row=='dihedral types':
        if modflag==0:
f.write('{0}

'.format(self.dihedraltypenum,row))
    elif modflag==1:
        f.write('{0}

'.format(len(self.dihedralcoef),row))
    elif row=='improper types':
        if modflag==0:
            f.write('{0}

'.format(self.impropertypenum,row))
        elif modflag==1:
            f.write('{0}

'.format(len(self.impropercoef),row))
    elif row=='xlo xhi':
        f.write('{0} {1}

'.format(self.xdimension[0],self.xdimension[1],row))
    elif row=='ylo yhi':
        f.write('{0} {1}

'.format(self.ydimension[0],self.ydimension[1],row))
    elif row=='zlo zhi':
        f.write('{0} {1}

'.format(self.zdimension[0],self.zdimension[1],row))
    elif row=='extra bond per atom':
        f.write('{0}

'.format(self.extrabonds,row))
    elif row=='xy xz yz':
        f.write('{0} {1} {2}

'.format(self.tilt[0],self.tilt[1],self.tilt[2],row))
    elif row=='Atoms':
        for line in self.atoms:
            f.write('
')
            for item in line:
                f.write(' {0}'.format(item))
            f.write('
')
    elif row=='Velocities':
        for line in self.velocities:
            f.write('
')
            for item in line:
                f.write(' {0}'.format(item))
            f.write('
')

#new line between header and body portion or two body keywords
def write(line):
    for line in self.atoms:
        f.write('
')
        for item in line:
            f.write(' {0}'.format(item))
        f.write('
')
    f.write('
')
    for line in self.velocities:
        f.write('
')
        for item in line:
            f.write(' {0}'.format(item))
        f.write('
')
f.write(' {0}'.format(item))
#adds in each piece of body data with space imbetween
f.write('
') #allows space to be added
between the end of body data and a new keyword
elif row=='Masses':
f.write('
{0}'.format(row)) #new line
between header and body portion or two body keywords
f.write('
') #new line between body
each keyword and body data
    for line in self.masses:
        f.write('
') #creates a new line for
        adding body data
            for item in line:
                f.write(' {0}'.format(item))
#adds in each piece of body data with space imbetween
f.write('
') #allows space to be added
between the end of body data and a new keyword
elif row=='Shapes':
f.write('
{0}'.format(row)) #new line
between header and body portion or two body keywords
f.write('
') #new line between body
each keyword and body data
    for line in self.shapes:
        f.write('
') #creates a new line for
        adding body data
            for item in line:
                f.write(' {0}'.format(item))
#adds in each piece of body data with space imbetween
f.write('
') #allows space to be added
between the end of body data and a new keyword
elif row=='Dipoles':
f.write('
{0}'.format(row)) #new line
between header and body portion or two body keywords
f.write('
') #new line between body
each keyword and body data
    for line in self.dipoles:
        f.write('
') #creates a new line for
        adding body data
            for item in line:
                f.write(' {0}'.format(item))
#adds in each piece of body data with space imbetween
f.write('
') #allows space to be added
between the end of body data and a new keyword
elif row=='Bonds':
f.write('
{0}'.format(row)) #new line
between header and body portion or two body keywords
f.write('
') #new line between body
each keyword and body data
    for line in self.bonds:
f.write(\n') #creates a new line for adding body data
    for item in line:
        f.write(' {0}'.format(item))
#adds in each piece of body data with space imbetween
    f.write(\n') #allows space to be added between the end of body data and a new keyword
elif row=='Angles':
    f.write('\n{0}'.format(row)) #new line between header and body portion or two body keywords
    f.write(\n') #new line between body keyword and body data
    for line in self.angles:
        f.write(\n') #creates a new line for adding body data
        for item in line:
            f.write(' {0}'.format(item))
#adds in each piece of body data with space imbetween
        f.write(\n') #allows space to be added between the end of body data and a new keyword
elif row=='Dihedrals':
    f.write('\n{0}'.format(row)) #new line between header and body portion or two body keywords
    f.write(\n') #new line between body keyword and body data
    for line in self.dihedrals:
        f.write(\n') #creates a new line for adding body data
        for item in line:
            f.write(' {0}'.format(item))
#adds in each piece of body data with space imbetween
        f.write(\n') #allows space to be added between the end of body data and a new keyword
elif row=='Impropers':
    f.write('\n{0}'.format(row)) #new line between header and body portion or two body keywords
    f.write(\n') #new line between body keyword and body data
    for line in self.impropers:
        f.write(\n') #creates a new line for adding body data
        for item in line:
            f.write(' {0}'.format(item))
#adds in each piece of body data with space imbetween
        f.write(\n') #allows space to be added between the end of body data and a new keyword
elif row=='Pair Coeffs':
    f.write('\n{0}'.format(row)) #new line between header and body portion or two body keywords
    f.write(\n') #new line between body keyword and body data

for line in self.paircoef:
    f.write('
') #creates a new line for adding body data
    for item in line:
        f.write(' {0}'.format(item))
#adds in each peice of body data with space imbetween
    f.write('
') #allows space to be added between the end of body data and a new keyword
    if row=='Bond Coeffs':
        f.write('
{0}'.format(row)) #new line between header and body portion or two body keywords
        f.write('
') #new line between body keyword and body data
for line in self.bondcoef:
    f.write('
') #creates a new line for adding body data
    for item in line:
        f.write(' {0}'.format(item))
#adds in each peice of body data with space imbetween
    f.write('
') #allows space to be added between the end of body data and a new keyword
    if row=='Angle Coeffs':
        f.write('
{0}'.format(row)) #new line between header and body portion or two body keywords
        f.write('
') #new line between body keyword and body data
for line in self.anglecoef:
    f.write('
') #creates a new line for adding body data
    for item in line:
        f.write(' {0}'.format(item))
#adds in each peice of body data with space imbetween
    f.write('
') #allows space to be added between the end of body data and a new keyword
    if row=='Dihedral Coeffs':
        f.write('
{0}'.format(row)) #new line between header and body portion or two body keywords
        f.write('
') #new line between body keyword and body data
for line in self.dihedralcoef:
    f.write('
') #creates a new line for adding body data
    for item in line:
        f.write(' {0}'.format(item))
#adds in each peice of body data with space imbetween
    f.write('
') #allows space to be added between the end of body data and a new keyword
    if row=='Improper Coeffs':
        f.write('
{0}'.format(row)) #new line between header and body portion or two body keywords
        f.write('
') #new line between body keyword and body data
f.write('\n{0}'.format(row)) #new line between header and body portion or two body keywords
f.write('\n') #new line between body keyword and body data
for line in self.impropercoef:
    f.write('\n') #creates a new line for adding body data
    for item in line:
        f.write(' {0}'.format(item)) #adds in each piece of body data with space imbetween
    f.write('\n') #allows space to be added between the end of body data and a new keyword
else:
    pass
f.close()

def atomorder(self):
    """Takes self.atoms and organizes the atom id from least to greatest.
    If the atom ids are allready ordered this algorithm will do nothing.""
    current=range(len(self.atoms[0])) # initialize current
    [assumes self.atoms coloumn #'s does not change]
    for i in range(1,len(self.atoms[0])): #when i=0, self.atoms will not change; therefore its skipped
        for k in range(len(self.atoms[i])):
            current[k]=self.atoms[i][k]
        for j in range(i-1,-1,-1):
            for k in range(len(self.atoms[j])):
                self.atoms[j+1][k]=self.atoms[j][k]
            if int(current[0]) > int(self.atoms[j][0]):
                for k in range(len(current)):
                    self.atoms[j+1][k]=current[k]
                break
        elif j==0:
            for k in range(len(current)):
                self.atoms[j][k]=current[k]
            #dont need a break here because this is the last j value in the for loop

def addatoms(self, atoms, retlist=False):
    """Appends atoms to self.atoms. Assumes atoms are written in the correct atomtype format.
    Change added atom numbers in self.atoms so self.atoms will be in increasing sequential order.
    If retlist is True return a list of the modified atom numbers otherwise exit."
    initpos=len(self.atoms) #Store index of first appended atoms
for item in atoms:
    self.atoms.append(range(len(item)))

#initializing spots for the new atoms to go
numberchange= []  # initiate number change where the changed atom numbers will be stored
count = 0
for i in range(initpos, len(self.atoms)):
    for j in range(len(self.atoms[i])):
        if j == 0:
            self.atoms[i][0] = str(i+1)
            numberchange.append(str(i+1))
        else:
            self.atoms[i][j] = atoms[count][j]
        count += 1
if retlist: return numberchange
else: return # need to redo this algorithm so there is no referencing going on.

def adddata(self, data, keyword):
    """Adds data to a keyword's existing data structure
    All body keywords are viable except Atoms which has its own method
    All header keywords will do nothing in this algorithm because they have their own algorithm
    changes the body id number so id numbers will be in sequential order""
    if keyword == 'Velocities':
        pos = len(self.velocities)
        for item in data:
            self.velocities.append(item)  # adds data to existing data structure
            self.velocities[pos][0] = str(pos+1)  # changing body id number
            pos += 1
    elif keyword == 'Masses':
        pos = len(self.masses)
        for item in data:
            self.masses.append(item)  # adds data to existing data structure
            self.masses[pos][0] = str(pos+1)  # changing body id number
            pos += 1
    elif keyword == 'Shapes':
        pos = len(self.shapes)
        for item in data:
            self.shapes.append(item)  # adds data to existing data structure
            self.shapes[pos][0] = str(pos+1)  # changing body id number
            pos += 1
elif keyword=='Dipoles':
pos=len(self.dipoles)
    for item in data:
        self.dipoles.append(item) #adds data to existing data structure
        self.dipoles[pos][0]=str(pos+1) #changing body id number
        pos+=1
elif keyword=='Bonds':
pos=len(self.bonds)
    for item in data:
        self.bonds.append(item) #adds data to existing data structure
        self.bonds[pos][0]=str(pos+1)
        pos+=1
elif keyword=='Angles':
pos=len(self.angles)
    for item in data:
        self.angles.append(item) #adds data to existing data structure
        self.angles[pos][0]=str(pos+1) #changing body id number
        pos+=1
elif keyword=='Dihedrals':
pos=len(self.dihedrals)
    for item in data:
        self.dihedrals.append(item) #adds data to existing data structure
        self.dihedrals[pos][0]=str(pos+1) #changing body id number
        pos+=1
elif keyword=='Impropers':
pos=len(self.impropers)
    for item in data:
        self.impropers.append(item) #adds data to existing data structure
        self.impropers[pos][0]=str(pos+1) #changing body id number
        pos+=1
elif keyword=='Pair Coeffs':
pos=len(self.paircoef)
    for item in data:
        self.paircoef.append(item) #adds data to existing data structure
        self.paircoef[pos][0]=str(pos+1) #changing body id number
        pos+=1
elif keyword=='Bond Coeffs':

```python
pos = len(self.bondcoef)
for item in data:
    self.bondcoef.append(item)  # adds data to existing data structure
    self.bondcoef[pos][0] = str(pos+1)  # changing body id number
    pos += 1

elif keyword == 'Angle Coeffs':
    pos = len(self.anglecoef)
    for item in data:
        self.anglecoef.append(item)  # adds data to existing data structure
        self.anglecoef[pos][0] = str(pos+1)  # changing body id number
        pos += 1

elif keyword == 'Dihedral Coeffs':
    pos = len(self.dihedralcoef)
    for item in data:
        self.dihedralcoef.append(item)  # adds data to existing data structure
        self.dihedralcoef[pos][0] = str(pos+1)  # changing body id number
        pos += 1

elif keyword == 'Improper Coeffs':
    pos = len(self.impropercoef)
    for item in data:
        self.impropercoef.append(item)  # adds data to existing data structure
        self.impropercoef[pos][0] = str(pos+1)  # changing body id number
        pos += 1
```

# def modifydata(data, keyword): Will not implement
# """for modifying header data"""

def changeatomnum(self, data, originalatoms, atomchanges, vflag=False):
    # need to modify for dealing with velocities which are different
    # """Takes data which contains atom ids and alters those ids from the original atom id system
    # in originalatoms to the new atom id system in atomchanges."""
    # set up boolean array to match the size of data and with all True values
    #
    array = booleanarray(len(data), len(data[0]), True)
    #
    if vflag:  # for changing atomnumbers for velocities
```
for i in range(len(originalatoms)):  # len of originalatoms should match len of atomchanges
    if originalatoms[i][0] == atomchanges[i]:
        continue
    for j in range(len(data)):
        for k in range(1):
            if data[j][k] == originalatoms[i][0]:
                # checks if the atom id in data matches the atom id in originalatoms
                if array.getelement(j, k):  # if the boolean array is true
                    # set the boolean array to False and change the atom id in data to atomchanges
                    array.setelement(j, k, False)
                    data[j][k] = atomchanges[i]
                    break  # the change of boolean array to False insures will only be changed once.
            else:
                # for changing atom numbers for everything else
                for i in range(len(originalatoms)):  # len of originalatoms should match len of atomchanges
                    if originalatoms[i][0] == atomchanges[i]:
                        continue
                    for j in range(len(data)):
                        for k in range(2, len(data[j])):
                            if data[j][k] == originalatoms[i][0]:
                                # checks if the atom id in data matches the atom id in originalatoms
                                if array.getelement(j, k):  # if the boolean array is true
                                    # set the boolean array to False and change the atom id in data to atomchanges
                                    array.setelement(j, k, False)
                                    data[j][k] = atomchanges[i]
                                    break  # the change of boolean array to False insures
```python
def deletebodydata(self, keyword):
    """Changes a keyword's class data structure to an empty structure"""
    if keyword == 'Velocities':
        self.velocities = []
    elif keyword == 'Masses':
        self.masses = []
    elif keyword == 'Shapes':
        self.shapes = []
    elif keyword == 'Dipoles':
        self.dipoles = []
    elif keyword == 'Bonds':
        self.bonds = []
    elif keyword == 'Angles':
        self.angles = []
    elif keyword == 'Dihedrals':
        self.dihedrals = []
    elif keyword == 'Impropers':
        self.impropers = []
    elif keyword == 'Pair Coeffs':
        self.paircoef = []
    elif keyword == 'Bond Coeffs':
        self.bondcoef = []
    elif keyword == 'Angle Coeffs':
        self.anglecoef = []
    elif keyword == 'Dihedral Coeffs':
        self.dihedralcoef = []
    elif keyword == 'Improper Coeffs':
        self.impropercoef = []
    elif keyword == 'Atoms':
        self.atoms = []

    #the atom id in data
    #print data
    return data
```
Return a list of atom id changes for each molecule.

Step 3: Utilize each molecules list of atom id changes to change their data's atom id numbers.

Uses changeatomnum and returns the altered molecule's data.

Step 4: Add the altered molecules' data to lmpsdata's data using the method adddata""

#Use molecule index 0's keyword to change the equivalent lmpsdata structures to empty
print 'extracting the molecules back to data'
for keyword in molecule[0].keywords: # step 1
    self.deletebodydata(keyword)
atomchanges=[] #initializing step 2
for i in range(len(molecule)):# step 2
    atomchanges.append(self.addatoms(molecule[i].atoms,True))
    for i in range(len(molecule)):# step 3
        for keyword in molecule[i].keywords:
            if keyword=='Angles':
                molecule[i].angles=self.changeatomnum(molecule[i].angles,molecule[i].atoms,atomchanges[i])
                if keyword=='Bonds':
                    molecule[i].bonds=self.changeatomnum(molecule[i].bonds,molecule[i].atoms,atomchanges[i])
                    elif keyword=='Dihedrals':
                        molecule[i].dihedrals=self.changeatomnum(molecule[i].dihedrals,molecule[i].atoms,atomchanges[i])
                        elif keyword=='Velocities':
                            molecule[i].velocities=self.changeatomnum(molecule[i].velocities,molecule[i].atoms,atomchanges[i],True)
                            elif keyword=='Impropers':
                                molecule[i].impropers=self.changeatomnum(molecule[i].impropers,molecule[i].atoms,atomchanges[i])
                                for i in range(len(molecule)):# step 4
                                    for keyword in molecule[i].keywords:
                                        if keyword=='Angles':
self.adddata(molecule[i].angles,keyword)

    elif keyword== 'Bonds':

self.adddata(molecule[i].bonds,keyword)

    elif keyword== 'Dihedrals':

self.adddata(molecule[i].dihedrals,keyword)

    elif keyword== 'Velocities':

self.adddata(molecule[i].velocities,keyword)

    elif keyword== 'Impropers':

    


def density(self,ringsize,init,final,file=' '):
    """Create spherical shells from the initial radius to
    the final radius
    The spherical shell's thickness is defined by ringsize
    Calculate the mass of particles within each spherical shell
    Calculate the volumoe of each spherical shell
    Then calculate the density in each spherical shell
    Current code is written with the assumption the
    spheres are centered around the origin
    If a file is listed place the results in the variable
    file otherwise return the results
    The distance calculations are written assuming image
    flags are in the data file""
    rho=[]
    r=[init]  #initial radius to examine
    radius=init+ringsize
    while radius<final:  #finding the rest of the radii to
    examine
        r.append(radius)
        radius+=ringsize
        for radius in r:
            volume=4.0/3.0*3.1416*((radius+ringsize)**3 -
            radius**3)
            mass=0.0
            for row in self.atoms:
                if self.atomtype== 'angle' or
                self.atomtype== 'atomic' or self.atomtype== 'bond' or
                self.atomtype== 'charge' or
                self.atomtype== 'colloid' or self.atomtype== 'electron' or
                self.atomtype== 'full' or
                self.atomtype== 'granular' or self.atomtype== 'molecular' or

226
self.atomtype='peri':
l=len(row)-1

dist=float(row[l-3])**2+float(row[l-4])**2+float(row[l-5])**2
    if dist<(radius+ringsize)**2 and dist>=radius**2:
        if self.atomtype=='atomic' or
            self.atomtype=='colloid' or
            self.atomtype=='electron' or
            self.atomtype=='granular' or
            self.atomtype=='peri':
                type=int(row[1])-1
                mass+=float(self.masses[type]
[1])
            else:
                type=int(row[2])-1
                mass+=float(self.masses[type]
[1])

    #assumes self.masses is written
in atomtype order

eif self.atomtype=='dipole':
l=len(row)-1

dist=float(row[l-6])**2+float(row[l-7])**2+float(row[l-8])**2
    if dist<(radius+ringsize)**2 and dist>=radius**2:
        type=int(row[1])-1
        mass+=float(self.masses[type]
[1])

    #assumes self.masses is written
in atomtype order

eif self.atomtype=='ellipsoid':
l=len(row)-1

dist=float(row[l-7])**2+float(row[l-8])**2+float(row[l-9])**2
    if dist<(radius+ringsize)**2 and dist>=radius**2:
        type=int(row[1])
        mass+=float(self.masses[type]
[1])

    #assumes self.masses is written
in atomtype order

eif self.atomtype=='hybrid':

dist=float(row[2])**2+float(row[3])**2+float(row[4])**2
    if dist<(radius+ringsize)**2 and dist>=radius**2:
        type=int(row[1])
        mass+=float(self.masses[type]
[1])
# assumes self.masses is written

in atomtype order
rho.append(mass/volume)

if file==' ':
    return r,rho
else:
f=open(file,'w')
for i in range(len(r)):
    f.write('{0} {1}
'.format(r[i],rho[i]))
f.close()
return


def createxyz(self,file, routine='mass', values=None): 
    #edit for the data class 
    """This shows the particle surface. To show the 
particle surface after bonding has occurred, 
you will need to extract the particle than reinsert 
the particle into the class and use createxyz. 
Two possible routines one to use the masses from data 
and the other to use the atom type and values supplied by the 
user. 
The mass version is assessed by setting the routine to 
'mass' which is the default method. 
The other version is assessed by setting the routine 
to 'atomtype'. 
The other version takes values which is a list 
containing the value the user wants to assign those atomtypes to. 
The atomtypes of the values in the list will start at 
1 even if no atoms in molecule use 1. 
This makes it easier to find the atomtype and assign the value from the list 
All atom data is assumed to have image flags in the 
data."""

f=open(file,'w')
f.write('{0}
'.format(len(self.atoms)))
f.write('atoms
')
if routine=='mass':
    for line in self.atoms:
        if self.atomtype=='angle' or 
        self.atomtype=='bond' or self.atomtype=='full' or 
        self.atomtype=='molecular':
            type=int(line[2])-1 
            #3rd position
        else:
            type=int(line[1])-1 
            #2nd position
            mass=self.masses[type][1]
            if mass=='12.0107': elementnum=6
            elif mass=='15.9994': elementnum=8
elif mass=='26.9815':
elementnum=13
else:
    print 'no matching mass value. The method will exit'
    return
l=len(line)-1
if self.atomtype=='angle' or
    self.atomtype=='atomic' or self.atomtype=='bond' or
    self.atomtype=='colloid' or self.atomtype=='electron' or
    self.atomtype=='full' or
    self.atomtype=='granular' or self.atomtype=='molecular' or
    self.atomtype=='peri':
    f.write('{0} {1} {2}
{3}'.format(elementnum,line[l-5],line[l-4],line[l-3]))
elif self.atomtype=='dipole':
    f.write('{0} {1} {2}
{3}'.format(elementnum,line[l-8],line[l-7],line[l-6]))
elif self.atomtype=='ellipsoid':
    f.write('{0} {1} {2}
{3}'.format(elementnum,line[l-9],line[l-8],line[l-7]))
elif self.atomtype=='hybrid':
    f.write('{0} {1} {2}
{3}'.format(elementnum,line[2],line[3],line[4]))
f.close()
elif routine=='atomtype':
    for line in self.atoms:
        if self.atomtype=='angle' or
            self.atomtype=='bond' or self.atomtype=='full' or
            self.atomtype=='molecular':
            type=int(line[2])-1 #3rd position
        else:
            type=int(line[1])-1 #2nd position
            elementnum=values[type]
l=len(line)-1
        if self.atomtype=='angle' or
            self.atomtype=='atomic' or self.atomtype=='bond' or
            self.atomtype=='charge' or
            self.atomtype=='colloid' or self.atomtype=='electron' or
            self.atomtype=='full' or
            self.atomtype=='granular' or self.atomtype=='molecular' or
            self.atomtype=='peri':
            f.write('{0} {1} {2}
{3}'.format(elementnum,line[l-5],line[l-4],line[l-3]))
        elif self.atomtype=='dipole':
            f.write('{0} {1} {2}
{3}'.format(elementnum,line[l-8],line[l-7],line[l-6]))
elif self.atomtype=='ellipsoid':
    f.write('{0} {1} {2}
'.format(elementnum,line[l-9],line[l-8],line[l-7]))
elif self.atomtype=='hybrid':
    f.write('{0} {1} {2}
'.format(elementnum,line[2],line[3],line[4]))
    f.close()

class booleanarray:
    """A class that stores boolean values in a list of lists."""
    def __init__(self,rownum, colnum, initval):
        """ initialise a list of lists (array) with rownum corresponding to the number of lists in the list and colnum corresponding to the number of elements in the list's list. initval is the value the list of lists will be initialized with. initval should be a boolean value."""
        # initializing self.array
        self.array=[]
        for i in range(rownum):
            self.array.append(range(colnum))
        # setting elements of self.array to initval
        for i in range(rownum):
            for j in range(colnum):
                self.setelement(i,j,initval)

    def setelement(self,rownum, colnum, value):
        """Assigns value to the list of lists (array) element at rownum and colnum."""
        self.array[rownum][colnum]=value

    def getelement(self,rownum,colnum):
        """Returns element in the list of lists (array) at rownum and colnum."""
        return self.array[rownum][colnum]

def atomdistance(a,b,atomtype):
    """Returns the distance between atom a and b. Atomtype is the style the atom data is written in. All atom data is assumed to have image flags in the data. Atom a and atom b are assumed to be the same atomtype."""
    from math import sqrt
    if atomtype=='angle' or atomtype=='atomic' or atomtype=='bond' or\
atomtype=='charge' or atomtype=='colloid' or
atomtype=='electron' or
atomtype=='full' or atomtype=='granular' or
atomtype=='molecular' or
atomtype=='peri':
l=len(a)-1
dist=sqrt((float(a[l-3])-float(b[l-3]))**2\
 +(float(a[l-4])-float(b[l-4]))**2\
 +(float(a[l-5])-float(b[l-5]))**2)
elif atomtype==''dipole'':
l=len(a)-1
dist=sqrt((float(a[l-6])-float(b[l-6]))**2\
 +(float(a[l-7])-float(b[l-7]))**2\n +(float(a[l-8])-float(b[l-8]))**2)
elif atomtype==''ellipsoid'':
l=len(a)-1
dist=sqrt((float(a[l-7])-float(b[l-7]))**2\n +(float(a[l-8])-float(b[l-8]))**2\n +(float(a[l-9])-float(b[l-9]))**2)
elif atomtype==''hybrid'':
dist=sqrt((float(a[2])-float(b[2]))**2\n +(float(a[3])-float(b[3]))**2\n +(float(a[4])-float(b[4]))**2)
return dist
def distance(a,coord,atomtype):
    """Returns the distance between atom a and coord.
    Atomtype is the style the atom data is written in.
    All atom data is assumed to have image flags in the data.""
    from math import sqrt #need to alter this slightly
    if atomtype=='angle' or atomtype=='atomic' or
    atomtype=='bond' or
    atomtype=='charge' or atomtype=='colloid' or
    atomtype=='electron' or
    atomtype=='full' or atomtype=='granular' or
    atomtype=='molecular' or
    atomtype=='peri':
l=len(a)-1
dist=sqrt((float(a[l-3])-coord[2])**2\
 +(float(a[l-4])-coord[1])**2\n +(float(a[l-5])-coord[0])**2)
elif atomtype==''dipole'':
l=len(a)-1
dist=sqrt((float(a[l-6])-coord[2])**2\n +(float(a[l-7])-coord[1])**2\n +(float(a[l-8])-coord[0])**2)
elif atomtype==''ellipsoid'':
l=len(a)-1
dist=sqrt((float(a[l-7])-coord[2])**2\n
231
elif atomtype=='hybrid':
    dist=sqrt((float(a[2])-coord[0])**2
    +(float(a[3])-coord[1])**2
    +(float(a[4])-coord[0])**2)
return dist

class particlesurface:
    def __init__(self,particle,cutoff,atomid,atomtype,shape='sphere'):
        """Builds a particle surface with a specific shape from a particle
        The atoms choosen from the surface will have the specific atomid
        atomid will be given in terms of an integer and not a string.""
        self.particle=particle.atoms
        self.atomtype=atomtype
        self.cutoff=cutoff
        self.surface=[]
        self.particlelocator=[] #stores surface particle locations with respect to the particle
        self.deletedatoms=[]
        if shape=='sphere': self.createsphereresurf(atomid)

    def createsphereresurf(self,atomid):
        """Assumes sphere is centered around 0,0,0.
        Finds maximum distance particle with atomid.
        Than all atoms within cutoff distance of max distance will be included into self.surface
        Also will build a list of where the surface particles are in relationship to the particle.
        Will create a booleanarray to keep track of any changes made to the surface.""
        particledist=[]
        for row in self.particle: #Stores all of the distances of the particle in particledist
            if self.atomtype=='angle' or self.atomtype=='bond' or self.atomtype=='full' or self.atomtype=='molecular':
                type=int(row[2]) #3rd position
            else:
                type=int(row[1]) #2nd position
            if type==atomid: #only calculates the distance if the atom has the correct atomid number
particledist.append(distance(row, [0,0,0],self.atomtype))

else:
    particledist.append(0.0)
self.maxdist=max(particledist)# finds the max dist.

for i in range(len(particledist)):
    if particledist[i]>=(self.maxdist-self.cutoff):
        self.particlelocator.append(i)
        self.surface.append(self.particle[i])
self.bool=booleanarray(len(self.surface),1,True)

def getsurfatom(self,rownum):
    return self.surface[rownum]

def getbool(self,rownum):
    return self.bool.getelement(rownum,0)

def setbool(self,rownum,value):
    self.bool.setelement(rownum,0,value)

def removesurfatom(self,rownum):
    """note this does not actually remove the surface atom. Instead this adds a value to the list deletedatoms. This list is later used in extractparticle to actually delete those atoms""
    self.deletedatoms.append(self.particlelocator[rownum])

def extractparticle(self):
    """"""deletesatoms from particle from the list of deletedatoms and than returns the particle.""

    self.particle=self.deleteatoms(self.particle,self.deletedatoms)

    return self.particle

def deleteatoms(self,structure,rows):
    """"""delete atoms from particle and shifts the structure down""

    new=[]
    #multilple copying of b to the rows being replaced.
    if rows==[]:
        for line in structure:
            new.append(line) #if no rows need replacing copy structure
    return new
for i in range(rows[0]):
    new.append(structure[i])  # copy structure to new

until the first replaced row
count=0
for i in range(rows[0],len(structure)-len(rows)):
    # to replace rows and shift undeleted rows over
    for j in range(i+1+count,len(structure)):
        for val in rows:
            if val==j:
                count+=1
                break
            if val==j: continue
        else:
            new.append(structure[j])
            break

return new

def addatom(self,atomtype,charge=None,moleculenum=None):
    """Adds an atom to the particle surface between maxdist and the cutoff distance.
    This atom is stored in self.particle and self.surface.
    In the current coding the particle is centered at (0,0,0).
    This method has a required input of atomtype and optional inputs of charge and moleculenum.
    This method currently does not support correctly self.atomtype of dipole, electron, ellipsoid, granular, peri or hybrid
    Will use the image flags 0,0,0.
    Note: The atomtype here is different from the atomtype attribute as part of this class
    Note: If the added atom will be required for bonding later than you will need to extract the particle data and rebuild the surface, because this method doesn't include updates to the required variables due to some coding issues"

    import math, random
    # Checks to make sure self.atomtype is not set to an unsupported value
    if self.atomtype=='dipole' or
       self.atomtype=='electron' or
       self.atomtype=='ellipsoid' or self.atomtype=='peri' or
       self.atomtype=='hybrid':
        print self.atomtype, 'is not currently supported. But, you can add support by adding the needed functionality and inputs'
        return

    print 'adding an atom to the surface'
# Randomly assigns the position of a new atom in a spherical shell between self.cutoff and self.maxdist
foundpoint=False
while (foundpoint==False):
    x=random.uniform(-self.maxdist,self.maxdist)
    y=random.uniform(-self.maxdist,self.maxdist)
    try:
        z=random.uniform(math.sqrt(self.cutoff**2-x**2-y**2),math.sqrt(self.maxdist**2-x**2-y**2))
    except ValueError:
        continue
    distance=math.sqrt(x**2+y**2+z**2)
    if distance<self.maxdist and distance>=self.maxdist-self.cutoff:
        foundpoint=True

# initialize new row in self.particle and self.surface
self.particle.append([])
self.surface.append([])

pposition=len(self.particle)-1 #particle postion
sposition=len(self.surface)-1 #surface position

# Adds the atom id number to the new row
if self.atomtype=='angle' or self.atomtype=='bond' or self.atomtype=='full' or self.atomtype=='molecular':
    self.particle[pposition].append(str(pposition+1))
    self.surface[sposition].append(str(pposition+1))

    # Adds the atomtype and the moleculenum if needed to the new row
    self.particle[pposition].append(str(moleculenum))
    self.surface[sposition].append(str(moleculenum))
    self.particle[pposition].append(str(atomtype))
    self.surface[sposition].append(str(atomtype))

else:
    self.particle[pposition].append(str(atomtype))
    self.surface[sposition].append(str(atomtype))

    # Adds the charge if needed to the new row
    if self.atomtype=='charge' or self.atomtype=='full':
        self.particle[pposition].append(str(charge))
        self.surface[sposition].append(str(charge))
# Adds the atom's position to the new row
self.particle[pposition].append(str(x))
self.surface[sposition].append(str(x))
self.particle[pposition].append(str(y))
self.surface[sposition].append(str(y))
self.particle[pposition].append(str(z))
self.surface[sposition].append(str(z))

# Adds the atom's image flags to the new row
self.particle[pposition].append('0')
self.surface[sposition].append('0')
self.particle[pposition].append('0')
self.surface[sposition].append('0')
self.particle[pposition].append('0')
self.surface[sposition].append('0')

def createxyz(self, file, data, routine='mass', values=None):
    """This shows the particle surface. To show the particle surface after bonding has occurred, you will need to extract the particle than reinsert the particle into the class and use createxyz. Two possible routines one to use the masses from data and the other to use the atom type and values supplied by the user.

The mass version is assessed by setting the routine to 'mass' which is the default method.
The other version is assessed by setting the routine to 'atomtype'.
The other version takes values which is a list containing the value the user wants to assign those atomtypes to.
The atomtypes of the values in the list will start at 1 even if no atoms in molecule use 1.
This makes it easier to find the atomtype and assign the value from the list
All atom data is assumed to have image flags in the data.""

f = open(file, 'w')
f.write('{0}\n'.format(len(self.surface)))
f.write('atoms\n')
if routine == 'mass':
    for line in self.surface:
        if self.atomtype == 'angle' or self.atomtype == 'bond' or self.atomtype == 'full' or self.atomtype == 'molecular':
            type = int(line[2]) - 1 # 3rd position
        else:
type=int(line[1])-1 #2nd position
mass=data.masses[type][1]
if mass=='12.0107': elementnum=6
elif mass=='15.9994': elementnum=8
elif mass=='26.981539':elementnum=13
else:
    print 'no matching mass value. The method will exit'
    return
l=len(line)-1
if self.atomtype=='angle' or
    self.atomtype=='atomic' or self.atomtype=='bond' or\
    self.atomtype=='charge' or
    self.atomtype=='colloid' or self.atomtype=='electron' or\
    self.atomtype=='full' or
    self.atomtype=='granular' or self.atomtype=='molecular' or\
    self.atomtype=='peri':
    f.write('{0} {1} {2}
{3}\'.format(elementnum,line[l-5],line[l-4],line[l-3]))
elif self.atomtype=='dipole':
    f.write('{0} {1} {2}
{3}\'.format(elementnum,line[l-8],line[l-7],line[l-6]))
elif self.atomtype=='ellipsoid':
    f.write('{0} {1} {2}
{3}\'.format(elementnum,line[l-9],line[l-8],line[l-7]))
elif self.atomtype=='hybrid':
    f.write('{0} {1} {2}
{3}\'.format(elementnum,line[2],line[3],line[4]))
    f.close()
elif routine=='atomtype':
    for line in self.surface:
        if self.atomtype=='angle' or
            self.atomtype=='bond' or self.atomtype=='full' or\
            self.atomtype=='molecular':
            type=int(line[2])-1 #3rd position
        else:
            type=int(line[1])-1 #2nd position
            elementnum=values[type]
l=len(line)-1
    if self.atomtype=='angle' or
        self.atomtype=='atomic' or self.atomtype=='bond' or\
        self.atomtype=='charge' or
        self.atomtype=='colloid' or self.atomtype=='electron' or\
        self.atomtype=='full' or
        self.atomtype=='granular' or self.atomtype=='molecular' or\
        self.atomtype=='peri':
f.write('{0} {1} {2}
'.format(elementnum,line[l-5],line[l-4],line[l-3]))

elif self.atomtype=='dipole':
    f.write('{0} {1} {2}
'.format(elementnum,line[l-8],line[l-7],line[l-6]))

elif self.atomtype=='ellipsoid':
    f.write('{0} {1} {2}
'.format(elementnum,line[l-9],line[l-8],line[l-7]))

elif self.atomtype=='hybrid':
    f.write('{0} {1} {2}
'.format(elementnum,line[2],line[3],line[4]))

f.close()

def molecules(data,init,final,method='all'):
    """ There are two ways to initialize the class Lmpsmolecule. Method controls which way is chosen. Acceptable values for Method are 'all', 'atom'. The default method is 'all' """
    molecule=[]
    from multiprocessing import Pool
    p=Pool(4) #insures Pool is using the maximum number of processors available
    for i in range(init,final+1):
        molecule.append(p.apply_async(Lmpsmolecule, (i,data,method,)))
    for i in range(len(molecule)):
        molecule[i]=molecule[i].get()
    p.close()
    p.join()
    return molecule

class Lmpsmolecule:
    #Technically should be a meta class but written as a seperate class for easier coding.
    def __init__(self,moleculenum,data,method):
        """initiates lammps molecule structures and than extract the appropriate molecular structures from the base class data""
    # ***** Lammps Molecule Structures
    self.keywords=['Atoms'] #include atoms here since this will be in every instance of this class
    self.atoms=[] #extract
    self.angles=[] #extract if keyword is there
    self.bonds=[] #extract if keyword is there
    self.dihedrals=[] #extract if keyword is there
    self.impropers=[] #extract if keyword is there
    self.velocities=[] #extract if keyword is there
    # ***** Molecule number
    self.moleculenum=moleculenum
    # ***** Extracting the molecule's atom information from data
self.extract(data, method)
# ***** If methods is 'all' then self.extract will extract all of
the molecule's information from data

def extract(self, data, method):
    """uses base class data to extract the molecules atoms
and any other molecule data from molecule moleculenum.
This extraction is done in a two step process.
Step 1: extract data.atoms with moleculenum and
renumber self.atoms beginning from 1
    store extracted atom numbers from
data.atoms in a list called temp
    Step 2: pass temp and data to method changeatomnum
which extracts the rest of the Lammps
Molecule structures from data
    and changes the atomnumbers in those
structures to match the ones already in self.atoms""
    #checking to make sure atomtype is a valid molecule
    #if not a valid molecule print error and exit method
    if data.atomtype=="full" or
data.atomtype=="molecular":
    # ***** Sets the molecule's atomtype
        self.atomtype=data.atomtype
    else:
        print "not a valid molecular structure"
        return
    #extract the molecule self.moleculenum from data.atom
to self.atom
    atomnum=1
    temp=[]
    for i in range(len(data.atoms)):
        if int(data.atoms[i][1])==self.moleculenum:
            temp.append(data.atoms[i][0]) #store
    extracted atomnumbers into temp
    self.atoms.append(data.atoms[i]) #store
    extracted atom into self.atom
    self.atoms[atomnum-1][0]=str(atomnum)
    #change extracted atom to the correct atomnumber
    atomnum+=1 #update atomnumber
    #extract the rest of the Lammps Molecule structures
from data using changeatomnum
    if method=="atom":
        return
    else:
        self.changeatomnum(temp, data)

    def changeatomnum(self, temp, data='):
        """changes the atomnumbers in Lmpsmolecule structures
angles, bonds, dihedrals, impropers,
and velocities in a three step process.
If data is defined extract the rest of the keywords used for Lmpsmolecule.
Step 1A: If data is defined copy from data one of the above structures.
   If data is not defined skip this step.
Step 1B: If data is defined remove the rows of copy which do not contain any values from temp
Step 2: If data is defined extract from copy to an above Lmpsmolecule structure and remove the extracted rows
   If data is not defined skip this step.
Step 3: Use temp and self.atoms to convert the atomnumbers in Lmpsmolecule structures to the correct values.
   temp and self.atoms line up, so temp[i] and self.atoms[i][0] correspond to the current Lmpsmolecule structures atom numbers
   and correct values will use booleanarray class to ensure that lmpsmolecule's structure values are altered only once."
   #Step 1 and Step 2
   if data!=' ': #extract molecular keywords from data.keywords
      for keywords in data.keywords:
         if keywords=='Angles' or keywords=='Bonds' or keywords=='Dihedrals' or keywords=='Impropers' or keywords=='Velocities':
            self.keywords.append(keywords)
         for keywords in self.keywords:
            if keywords!='Atoms': print 'extracting the data from', keywords
      if keywords=='Angles':
         copy=self.copy(data.angles) #Step 1A
dnr=[] #dnr means do not remove a 0 means remove and a 1 means keep
         for j in range(len(copy)): #Step 1B
            dnr.append(0) #adds 0 to all list elements of dnr
         for item in temp: #finds copied structure that has temp values
            for j in range(len(copy)):
               for i in range(2,len(copy[j])):
                  if copy[j][i]==item:
                     dnr[j]=1
   #changes jth list element of dnr to 1

240
remove=[]
for j in range(len(dnr)): # finds dnr values that are still 0
    if dnr[j]==0:
        remove.append(j) #and appends their index value to the list remove

for j in range(len(dnr)):
    remove.append(j) #removes all unneeded rows from copy


structnum=1

print 'the length of data is',

len(copy)

for item in temp: #Step 2
    found=[]
    for j in range(len(copy)):
        for i in range(2,len(copy[j])):
            if copy[j][i]==item:
                found.append(j)


self.angles.append(copy[j])

l=len(self.angles)-1

[0]=str(structnum)

structnum+=1

break

# print 'the item is', item
#
# print 'found is', found

else:
    copy=self.copy(data.bonds) #Step 1A

for j in range(len(copy)): #Step 1B
    dnr.append(0) #adds 0 to all list elements of dnr

for item in temp: #finds copied structure that has temp values
    for j in range(len(copy)):
        for i in range(2,len(copy[j])):
            if copy[j][i]==item:
                dnr[j]=1

#changes jth list element of dnr to 1

break

remove=[]
for j in range(len(dnr)):  # finds dnr values that are still 0
    if dnr[j] == 0:
        remove.append(j)  # and appends their index value to the list remove
        copy = self.deleterows(copy, remove)  #removes all unneeded rows from copy
structnum = 1
print 'the length of data is', len(copy)
for item in temp:  #Step 2
    found = []
    for j in range(len(copy)):
        for i in range(2, len(copy[j])):
            if copy[j][i] == item:
                found.append(j)

    self.bonds.append(copy[j])
    l = len(self.bonds) - 1
    self.bonds[l][0] = str(structnum)
    structnum += 1
    break
    # print 'the item is', item
    # print 'found is', found

    copy = self.deleterows(copy, found)

elif keywords == 'Dihedrals':
    copy = self.copy(data.dihedrals)  #Step 1A
    dnr = []  # dnr means do not remove a 0
    # means remove and a 1 means keep
    for j in range(len(copy)):  #Step 1B
        dnr.append(0)  #adds 0 to all
        for item in temp:  #finds copied structure that has temp values
            for j in range(len(copy)):
                for i in range(2, len(copy[j])):
                    if copy[j][i] == item:
                        dnr[j] = 1
        #changes jth list element of dnr to 1
        break
    remove = []
for j in range(len(dnr)):  # finds dnr values that are still 0
    if dnr[j]==0:
        remove.append(j)  # and appends their index value to the list remove
for j in range(len(dør)):  # removes all unneeded rows from copy
    remove = self.deleterows(copy, remove)

len(copy)

structnum=1

print 'the length of data is',

for item in temp:  # Step 2
    found = []
    for j in range(len(copy)):
        for i in range(2,len(copy[j])):
            if copy[j][i]==item:
                found.append(j)

    self.dihedrals.append(copy[j])

    l=len(self.dihedrals)-1

    self.dihedrals[l][0]=str(structnum)
    structnum+=1
    break

# print 'the item is', item
# print 'found is', found

elif keywords=='Impropers':
    copy = self.copy(data.impropers)  # Step 1B

    dnr=[]  # dnr means do not remove a 0
    means remove and a 1 means keep
    for j in range(len(copy)):
        dnr.append(0)  # adds 0 to all list elements of dnr
    for item in temp:  # finds copied structure that has temp values
        for j in range(len(copy)):
            for i in range(2,len(copy[j])):
                if copy[j][i]==item:
                    dnr[j]=1
# changes jth list element of dnr to 1

    remove=[]
for j in range(len(dnr)):  # finds dnr values that are still 0
    if dnr[j]==0:
        remove.append(j)  # and appends their index value to the list remove

# removes all unneeded rows from copy

structnum=1
print 'the length of data is',

len(copy)

for item in temp:  # Step 2
    found=[]
    for j in range(len(copy)):
        for i in range(2,len(copy[j])):
            if copy[j][i]==item:
                found.append(j)

self.improps.append(copy[j])

l=len(self.impropers)-1

self.impropers[l][0]=str(structnum)

    structnum+=1
    break

# print 'the item is', item
# print 'found is',

found

if keywords=='Velocities':
    copy=self.copy(data.velocities)  # Step 1

else:
    copy=self.deleterows(copy,found)

else:
    dnr=[]  # dnr means do not remove a 0
    means remove and a 1 means keep

for j in range(len(copy)):  # Step 1B
    dnr.append(0)  # adds 0 to all list elements of dnr

for item in temp:  # finds copied structure that has temp values
    for j in range(len(copy)):
        for i in range(1):
            if copy[j][i]==item:
                dnr[j]=1

# changes jth list element of dnr to 1

    break

remove=[]

for j in range(len(dnr)):  # finds dnr values that are still 0
```python
if dnr[j]==0:
    remove.append(j)  # and
append their index value to the list remove

# removes all unneeded rows from copy
remove = self.deleterows(copy, remove)

structnum=1
print 'the length of data is',
len(copy)

for item in temp:  # Step 2
    found=[]
    for j in range(len(copy)):
        for i in range(1):
            if copy[j][i]==item:
                found.append(j)

self.velocities.append(copy[j])

l=len(self.velocities)-1

self.velocities[l][0]=str(structnum)
    structnum+=1
    break
# print 'the item is', item
# print 'found is',

found

for i in range(len(temp)):
    if temp[i]==self.atoms[i][0]: continue
for j in range(len(copy)):
    for k in range(2,len(copy[j])):
        if copy[j][k]==temp[i]:
            if array.get_element(j,k):
                # if the boolean array is true
                self.angles[j][k]=self.atoms[i][0]

array.set_element(j,k,False)
```

---

245
elif keywords=='Bonds':
    copy=self.copy(self.bonds)

    array=bool_array(len(copy),len(copy[0]),True) #creating a booleanarray with true values
    for i in range(len(temp)):
        if temp[i]==self.atoms[i][0]:
            continue
        for j in range(len(copy)):
            for k in range(2,len(copy[j])):
                if copy[j][k]==temp[i]:
                    if array.element(j,k):
                        self.bonds[j][k]=self.atoms[i][0]
                        array.set_element(j,k,False)
                        break

elif keywords=='Dihedrals':
    copy=self.copy(self.dihedrals)

    array=bool_array(len(copy),len(copy[0]),True) #creating a booleanarray with true values
    for i in range(len(temp)):
        if temp[i]==self.atoms[i][0]:
            continue
        for j in range(len(copy)):
            for k in range(2,len(copy[j])):
                if copy[j][k]==temp[i]:
                    if array.element(j,k):
                        self.dihedrals[j][k]=self.atoms[i][0]
                        array.set_element(j,k,False)
                        break

elif keywords=='Impropers':
    copy=self.copy(self.impropers)

    array=bool_array(len(copy),len(copy[0]),True) #creating a booleanarray with true values
    for i in range(len(temp)):
        if temp[i]==self.atoms[i][0]:
            continue
        for j in range(len(copy)):
            for k in range(2,len(copy[j])):
                if copy[j][k]==temp[i]:
                    if array.element(j,k):
                        self.impropers[j][k]=self.atoms[i][0]
                        array.set_element(j,k,False)
                        break
array.setelement(j,k,False)  
break

elif keywords=='Velocities':
    copy=self.copy(self.velocities)

array=booleanarray(len(copy),len(copy[0]),True)  #creating a 
booleanarray with true values
    for i in range(len(temp)):
        if temp[i]==self.atoms[i][0]: continue
        for j in range(len(copy)):
            for k in range(1):
                if copy[j][k]==temp[i]:
                    if array.getelement(j,k):
                        #if the boolean array is true
                            self.velocities[j][k]=self.atoms[i][0]
            array.setelement(j,k,False)  
break

def copy(self,structure):
    """copies structure to same and returns same.""
    same=[]
    for row in structure:
        same.append(row)
    return same

def deleterows(self,structure,rows): # run through this 
{structure is list of strings and rows is list 
#of numbers}
    """delete rows in a structure and shifts the structure 
up 
rows must be in increasing order for this algorithm to 
work correctly""
    new=[]
    #multiple copying of b to the rows being replaced.
    if rows==[]:
        for line in structure:
            new.append(line)  #if no rows need replacing 
    copy structure
        return new
    for i in range(rows[0]):
        new.append(structure[i])  #copy structure to new
until the first replaced row
    count=0
    for i in range(rows[0],len(structure)-len(rows)):
        #to 
replace rows and shift undeleted rows over
        for j in range(i+1+count,len(structure)):
for val in rows:
    if val==j:
        count+=1
        break
    if val==j: continue
else:
    new.append(structure[j])
    break
return new

def modifyatom(self,atomnumber,column,newvalue):
    """modifies self.atom[atomnumber-1][column] to newvalue.
    *note: newvalue needs to be a string
    if column is 0 than changeatomnum method might need running for all atoms
    that have had their atomnumber(column=0) modified.
    changeatomnum method is not ran in this method when column=0"
    self.atoms[atomnumber-1][column]=newvalue

def deleteatoms(self,atomnumbers,atomid):
    """Algorithm to find all atoms bonded to atomnumbers in the direction of atomid.
    Atomid can be a list or a single integer. The single integer corresponds to atom's atomtype value.
    The list corresponds to the atom's atomid values. The only difference between both cases is in the top portion of code.
    When all bonded atoms have been found; ie: (the modified return values from findbonds yields an empty list);
    delete those bonded atoms and all molecule structures which contain those atoms.
    Convert the list of bonded atoms into rows and delete those rows from molecule.atoms"
    print 'finding atoms to delete'
    bondedatoms=[]
    # 1st iteration
    nextatoms=self.findbonds(atomnumbers)
    try:
        atomid[0]
    except TypeError:
        #need to remove atoms from nextatoms which dont have the proper atom id
        testflag=True
        i=0
        while testflag:
            testflag=False
            for val in nextatoms:
                if val==j: 
                    count+=1
                    break
                if val==j: continue
            else:
                new.append(structure[j])
                break
    return new
if i>len(nextatoms)-1: break #For the case where i becomes greater than the len of nextatoms break loop
if int(self.atoms[nextatoms[i]-1][2])!=atomid:#uses the atom id for the row in atoms and than checks atom id
    del nextatoms[i] #deletes the atom at i
    i=1 #and than decreases i by 1 so next atom in the list will line up when i is increased
else:
    #need to remove atoms from nextatoms which dont have the proper atom id
    testflag=True
    i=0
    while testflag:
        if i>len(nextatoms)-1: break #For the case where i becomes greater than the len of nextatoms break loop
        keep=False
        for id in atomid:
            if int(self.atoms[nextatoms[i]-1][0])==id: #checking if atomid is in next atom
                keep=True #keep this atom
                break
        if not keep:
            del nextatoms[i] #deletes the atom at i
            i=1 #and than decreases i by 1 so next atom in the list will line up when i is increased
            i+=1 #increase i by 1

#append next atoms into bondedatoms
#copy next atoms into prevatoms
prevatoms=[]
for atom in nextatoms:
    bondedatoms.append(atom)
    prevatoms.append(atom)

#2nd iteration
if prevatoms==[]:
    print 'no bonds were found in first iteration that had atomid criteria'
    return

nextatoms=self.findbonds(prevatoms)
#need to remove atoms from nextatoms which are in atomnumbers
for atom in atomnumbers:
    for i in range(len(nextatoms)):
if nextatoms[i] == atom:  # checking if atom is in next atom
def nextatoms[i]  # delete the atom at i
break
if nextatoms == []:  # all bonds from find bonds have allready been added to bondedatoms

# append next atoms into bondedatoms
# copy next atoms into prevatoms
prevatoms = []
for atom in nextatoms:
bondedatoms.append(atom)
prevatoms.append(atom)

# iterative process for finding the rest of the atoms bonded to atomnumbers in the direction of atomid.
while prevatoms != []:
    nextatoms = self.findbonds(prevatoms)
    # need to remove atoms from nextatoms which are in the prevatoms
    for atom in bondedatoms:
        for i in range(len(nextatoms)):
            if nextatoms[i] == atom:  # checking if atom is in next atom
                del nextatoms[i]  # delete the atom at i
                break
    if nextatoms == []:  # all bonds from find bonds have allready been added to bondedatoms
        # append next atoms into bondedatoms
        # copy next atoms into prevatoms
        prevatoms = []
        for atom in nextatoms:
            bondedatoms.append(atom)
            prevatoms.append(atom)

print 'the atoms to delete are', bondedatoms
print 'deleting atoms from structures'
# delete bonded atoms from the molecule's structures
except the atom structure
for i in range(1, len(self.keywords)):  # goes through all keywords except the atom keyword
    print self.keywords[i]
    if self.keywords[i] == 'Angles':

        rows = self.findatomnumbers(bondedatoms, self.angles, False)
        rows = self.listorder(rows)  # to order the rows in increasing order
self.angles=self.deleterows(self.angles,rows) #requires that rows be in increasing order
    elif self.keywords[i]=='Bonds':
        rows=self.findatomnumbers(bondedatoms,self.bonds,False)
        rows=self.listorder(rows) #to order the rows in increasing order
        self.bonds=self.deleterows(self.bonds,rows) #requires that rows be in increasing order
    elif self.keywords[i]=='Dihedrals':
        rows=self.findatomnumbers(bondedatoms,self.dihedrals,False)
        rows=self.listorder(rows) #to order the rows in increasing order
        self.dihedrals=self.deleterows(self.dihedrals,rows) #requires that rows be in increasing order
    elif self.keywords[i]=='Impropers':
        rows=self.findatomnumbers(bondedatoms,self.impropers,False)
        rows=self.listorder(rows) #to order the rows in increasing order
        self.impropers=self.deleterows(self.impropers,rows) #requires that rows be in increasing order
    elif self.keywords[i]=='Velocities':
        rows=self.findatomnumbers(bondedatoms,self.velocities,True)
        rows=self.listorder(rows) #to order the rows in increasing order
        self.velocities=self.deleterows(self.velocities,rows) #requires that rows be in increasing order

print 'Atoms'
#convert bondedatoms from atom numbers to row numbers
for i in range(len(bondedatoms)):
    bondedatoms[i]=1
bondedatoms=self.listorder(bondedatoms) #to order the row numbers in increasing order
#delete bonded atoms (row numbers) from the atom structure
self.atoms=self.deleterows(self.atoms,bondedatoms) #requires that row numbers be in increasing order

def findatomnumbers(self,atomnumbers,structure,vflag): #need to read through this algorithm
Algorithm to find atom numbers in a molecule structure except.
Atoms structure is not handled in here.
Returns a list of the rows in which the atom numbers are contained in the molecule structure"

rows=[]
if vflag: # for handling velocity structure
    for atom in atomnumbers:
        for i in range(len(structure)):
            if int(structure[i][0])==atom:
                # duplicate rows for vflag=True
                rows.append(i)
                break
else: # for handling all other structures except atoms
    for atom in atomnumbers:
        for i in range(len(structure)):
            for j in range(2,len(structure[i])):
                if int(structure[i][j])==atom:
                    # need to make sure duplicate value of rows are not being added
                    if rows==[]:
                        rows.append(i)
                    else:
                        # checking for duplicate values of rows
                        duplicateflag=0
                        for k in range(len(rows)):
                            if rows[k]==i:
                                duplicateflag=1
                                break
                        if duplicateflag==0:
                            # if no duplicates adds bond number
                            rows.append(i)
                        break

    print 'the finished row is', rows
return rows

def listorder(self,struct):
    """ Takes struct and organizes the list from least to greatest.
    If the list is already ordered this algorithm will do nothing.""

252
if len(struct)==1: return struct # with the length at 1; there is only one element and therefore nothing to order
for i in range(1,len(struct)): # when i=0, struct will not change; therefore it's skipped
    copy=struct[i]
    for j in range(i-1,-1,-1):
        struct[j+1]=struct[j]
        if copy > struct[j]:
            struct[j+1]=copy
            break
    elif j==0:
        struct[j]=copy
        # don't need a break here because this is the last j value in the for loop
    print 'the organized row is', struct
    return struct

def findbonds(self,atomnumbers):
    """Algorithm to find all atoms bonded to atomnumbers. Returns a list of atomnumbers""
    # finds the bonds in which the atomnumbers are located
    bondids=[]
    for i in range(len(atomnumbers)):
        for j in range(len(self.bonds)):
            for k in range(2,len(self.bonds[j])):
                if int(self.bonds[j][k])==atomnumbers[i]:
                    if bondids==[]:
                        bondids.append(int(self.bonds[j][0])) # appends the bond number
                    else:
                        # checking for duplicates of bondids
                        duplicateflag=0
                        for l in range(len(bondids)):
                            if bondids[l]==int(self.bonds[j][0]):
                                duplicateflag=1
                                break
                        if duplicateflag==0: # if no
                            bondids.append(int(self.bonds[j][0]))
                            break
Using bondids find the atoms bonded to atomnumbers

```python
bondedatoms=[]
from math import fabs
for id in bondids:
    for atoms in atomnumbers:
        found=False
        for i in range(2,len(self.bonds[id-1])):
            if int(self.bonds[id-1][i])==atoms:
                j=int(fabs(i-5))  # switches the index from the atomnumber location to the other location
                bondedatoms.append(int(self.bonds[id-1][j]))  # appends the atomnuber at the other location
                found=True
                break
        if found==True: break
    return bondedatoms
```

def findparticlebondingpoints(self,particle,atomid,cutoffdistance,bondnumber):
    """Particle is a particlesurfaceobject, atomid is an int.
    Finds atoms with atomid in the molecule which are less than the cutoffdistance from atoms in the particle.
    The found atoms and particles locations are stored in a 3 dimensional list called possiblebonds
    The first index corresponds with the molecule's atom
    The second index corresponds with the molecule/particle combination
    The third index corresponds with whether the value is the molecule or the particle
    Always bonds the two ends of the molecule that meet cutoff requirement.
    All other possible bonds are randomly choosen until the required bondnumbers are met.
    After every bond is choosen, the particle object's boolean list is updated, and possiblebonds is updated.
    The update to possiblebonds involves removing the row from which the bonded molecule's atom is located
    and also removing the particle atom and it's corresponding bonded atom from other rows of possiblebonds.
    The final bonds are all stored in self.bonding as a 2 dimensional list""
    possiblebonds=[]
    for i in range(len(self.atoms)):
        if int(self.atoms[i][2])==atomid:
            row=[]
            for j in range(len(particle.surface)):
# assumes molecule and particle have same atomtype if not true than atomdistance will give bad value

```python
if atomdistance(self.atoms[i], particle.surface[j], self.atomtype) <= cutoffdistance:
    if particle.getbool(j):
        row.append([i, j])  # if not bonded than can form a possible bond
        if row!=[]: possiblebonds.append(row)  # need to correct this....
```

# initiate section which assigns bonds into bonding information
bonds = 0
self.bondinginformation = []  # initiate new class member

# Checks to see if no bonds are possible [2 possible cases]
if possiblebonds == []:
    print 'no possible bonds can be formed'
    return
if bondnumber == 0:
    print 'bondnumber is 0; so, no possible bonds can form'
    return

# section which assigns a bond to the first molecule atom which can be bonded.
self.bondinginformation.append(self.particlebondinginfo(particle, possiblebonds, 0))
bonds += 1
if bonds == bondnumber: return
del possiblebonds[0]  # deletes possible bonds to the first molecule which can be bonded
l = len(self.bondinginformation) - 1

possiblebonds = self.updatepossiblebonds(possiblebonds, self.bondinginformation[l][1])  # updates possiblebonds by removing any bonds which contain the newly bonded particle.

if possiblebonds == []: return
# section which finds the last molecule atom which can be bonded
l = len(possiblebonds) - 1
while possiblebonds[l] == []:  # to find the last molecule atom which can be bonded
if possiblebonds == []: return
del possiblebonds[l] #since there are no more possible bonds in this location delete
l = l - 1 # go to next possible spot where the last molecule atom could be bonded

# section which assigns a bond to the last molecule atom which can be bonded.

self.bondinginformation.append(self.particlebondinginfo(particle, possiblebonds, l))
   bonds += 1
   if bonds == bondnumber: return
   del possiblebonds[l] # deletes possible bonds to the last molecule which can be bonded
   l = len(self.bondinginformation) - 1

possiblebonds = self.updatepossiblebonds(possiblebonds, self.bondinginformation[l][1])

if bondnumber - bonds >= len(possiblebonds): # the rest of the bonds are assigned in order
   while possiblebonds != []:
      if possiblebonds[0] == []: # if row of possible bonds is empty then delete the row
         del possiblebonds[0]
      else: # else find a bond in the row of possible bonds
         self.bondinginformation.append(self.particlebondinginfo(particle, possiblebonds, 0))
         # dont need to update bonds since possiblebonds will become empty at the same time or before bonds
         # is equal to bondnumber
         del possiblebonds[0]
         l = len(self.bondinginformation) - 1

possiblebonds = self.updatepossiblebonds(possiblebonds, self.bondinginformation[l][1])
return
else: # the rest of the bonds are assigned randomly
   from random import randint # use to randomly choose an index to bond.
   while bonds < bondnumber:
if possiblebonds==[]: break  # exits while loop when possiblebonds has become an empty set.
# this ensures that in the case there are not enough viable bonds from possiblebonds to
# reach the bondnumber. Then, the while loop will not become infinite.
l=len(possiblebonds)-1
i=randint(0,l)
if possiblebonds[i]==[]:  # if row of possible bonds is empty then delete the row
    del possiblebonds[i]
else:  # else find a bond in the row of possible bonds
    self.bondinginformation.append(self.particlebondinginfo(particle, possiblebonds,i))
    bonds+=1
del possiblebonds[i]
l=len(self.bondinginformation)-1

possiblebonds=self.updatepossiblebonds(possiblebonds,self.bondinginformation[l][1])
return

def particlebondinginfo(self,particle,possiblebonds,index1):
    """Takes list of possiblebonds and assigns a bond from possiblebonds[index1].
    Then assigns false to particle.setbool at the bonding location
    to ensure no more bonds can form with that particle.
    Returns the assigned bond."""
    from random import randint  # use to randomly choose 0 and 1 where 1 is keep.
    for i in range(len(possiblebonds[index1])):
        if i==len(possiblebonds[index1])-1:  # automatically bonds under this condition
            break
        else:  # bond has random chance of forming
            if randint(0,1)==1:  # bond forms
                break
            else:  # no bond forms
                continue
            particle.setbool(possiblebonds[index1][i][1],False)  # makes sure no more bonds can form
        return possiblebonds[index1][i]
def updatepossiblebonds(self, possiblebonds, particlenum):
    """finds the particle number in the remaining possible bonds than deletes that bonding information. This algorithm assumes that particlenum can exist only once in each row of possiblebonds."""
    for i in range(len(possiblebonds)):
        for j in range(len(possiblebonds[i])):
            if possiblebonds[i][j][1] == particlenum:
                del possiblebonds[i][j]
                break  # go to next row of possiblebonds
    return possiblebonds

def bondtoparticle(self, particle, atomid, newid, newcharge):
    """Bonds molecule to particle. Particle is a particlesurface object. Moves the molecule's atoms bonded to the particle to the particle's position.
    Alters the atomtype of the molecule's atoms bonded to the particle to newid.
    Alters the charge of the molecule's atoms bonded to the particle to newcharge.
    Because bonds have formed between the molecule's atoms and the particle's atoms, atoms with atomid on the molecule need to be removed otherwise the molecule's atoms will have to many bonds.
    self.deleteatoms will take care of deleting the atoms atomid and atoms down the polymer chain in the direction away from the molecule/particle bond.
    Now remove the bonded particle atoms from the particle surface because the bonded molecule atoms have replaced those particle atoms and their bonds with the rest of the particle.
    Newid must be a string. Atomid can now be a list or an integer.
    The list is a list of atom's id values and the single integer is atom's atomtype"""
    #moves the molecule's atoms bonded to the particle to the particle's position
    #need to do this step first before the atom id's and row indexes get out of sync
    #which will occur after atoms get deleted.
    print 'modifying the bonded molecules atom information'
    for i in range(len(self.bondinginformation)):
        #patom=particle.getsurfatom(self.bondinginformation[i][1])
        #if self.atomtype=='molecular':#3-5-
        >x,y,z[molecular]
# for j in range(3,6):
#
self.modifyatom(self.bondinginformation[i][0]+1,j,patom[j])#uses the atomid here
#elif self.atomtype=='full':#4-6->x,y,z[full]
# for j in range(4,7):
#
self.modifyatom(self.bondinginformation[i][0]+1,j,patom[j])#uses the atomid here
#alters the atomtype to newid of the molecule's atoms bonded to the particle.
self.modifyatom(self.bondinginformation[i][0]+1,2,newid) #uses the atomid here
if self.atomtype=='full':
    # Alters the charge of the molecule's atoms bonded to the particle to newcharge
    self.modifyatom(self.bondinginformation[i][0]+1,3,newcharge)

#This is a separate loop so atom id's and row indexes for previous steps wont get out of sync
#create atomnumbers to begin deletion process.
atomnumbers=[]
for i in range(len(self.bondinginformation)):
    atomnumbers.append(self.bondinginformation[i][0]+1)#using actual atomnumbers rather than the row index

#Call algorithm to find all atoms bonded to atomnumbers in the direction of atomid.
#Than delete those atoms and all molecule structures which contain those atoms.
self.deleteatoms(atomnumbers,atomid)

print 'begining deletion process of the surface atoms for which the molecule atoms have replaced'
#Goes through the bondinginformation and superficially removes the surfaceatom
for i in range(len(self.bondinginformation)):
    particle.removesurfatom(self.bondinginformation[i][1]) #uses the row number
    #Allows the particle extract method used outside this class to remove this atom.

def createxyz(self,file,data,routine='mass', values=None):
    """Two possible routines one to use the masses from data and the other to use the atom type and values supplied by the user."""
The mass version is assessed by setting the routine to 'mass' which is the default method. The other version is assessed by setting the routine to 'atomtype'. The other version takes values which is a list containing the value the user wants to assign those atomtypes to. The atomtypes of the values in the list will start at 1 even if no atoms in molecule use 1. This makes it easier to find the atomtype and assign the value from the list. All atom data is assumed to have image flags in the data.

```python
f = open(file, 'w')
f.write('{0}
'.format(len(self.atoms)))
f.write('atoms
')
if routine == 'mass':
    for line in self.atoms:
        type = int(line[2]) - 1
        mass = data.masses[type][1]
        if mass == '12.0107': elementnum = 6
        elif mass == '15.9994': elementnum = 8
        elif mass == '26.9815': elementnum = 13
        else:
            print 'no matching mass value. The method will exit'
            return
        l = len(line) - 1
        f.write('{0} {1} {2}
'.format(elementnum, line[l - 5], line[l - 4], line[l - 3]))
f.close()
elif routine == 'atomtype':
    for line in self.atoms:
        type = int(line[2]) - 1
        elementnum = values[type]
        l = len(line) - 1
        f.write('{0} {1} {2}
'.format(elementnum, line[l - 5], line[l - 4], line[l - 3]))
f.close()```
APPENDIX H

METAL POTENTIAL FITTING

H.1 Main Controller (potfit.f)

```fortran
program potfit
implicit none
C global Variables
integer a, b, c, m, n
common/parameters/a,b,c,m,n
double precision a0, rcut
COMMON/pot_fitting/a0,rcut
double precision mass, at
common/mdprop/mass,at
double precision ecoh, bulk, omega
common/matprop/ecoh,bulk,omega
integer aold,bold,cold
common/oldies/aold,bold,cold
double precision sratest,srbtest,srctest
common/srlettertest/sratest,srbtest,srctest
integer switch
common/roseswitch/switch
integer elenum, fitmethod, potmethod
character*2 element(6)
common/control/elenum, element, fitmethod, potmethod
C program variables
double precision sr,srtestold,srtestnew, dummy1, dummy2
integer amax,bmax,cmax,atest,btest,ctest,testnum
integer i

C ****Apparently the original method in mishin does not work for
C **** the temperature rescale method
C **** new method: Choose a,b, or c on average sr of fit and test
C **** fit potentials until a,b,and c are at amax,bmax,and cmax.
C **** a,b, and c values along with average sr of fit and test
C **** are stored in srstorage
C **** choose the potential with the lowest average sr in srstorage

C Need to change back to the original method outlined in Mishin's
C paper
C get rid of average sr of fit and test
C Get rid of srstorage
```
c number of properties calculated in fitting
   open (15,file='./controls/fit.txt',status='old')
   read (15,*) n
   close (15)
c number of properties calculated in testing
   open (15,file='./controls/test.txt',status='old')
   read (15,*) testnum
   close (15)
C Control information (number of elements, name of elements, fitting and potential method)
   open (3,file='./controls/control.txt',status='old')
   read (3,*) elenum
   if (elenum.gt.6) stop
   do 99 i=1,elenum
      read (3,*) element(i)
   99 continue
   read (3,*) fitmethod
   read (3,*) potmethod
   close (3)
C code for pure metal (ie. 1 element)
   if (elenum.eq.1) then
      c Read in Important Material Properties
      open (3,file='./database/element(1)/materialconst.txt',status='old')
      read (3,*) mass,at, a0, ecoh, bulk
      c ecohj=ecoh*1.60217733d-19
      omega=a0**3/4*1d-30
      close (3)
      C Set up Structures for Property Calculations Used in Fitting and Testing.
      call fccstruct(a0)
      call hexagonalclosepackstruct(a0)
      call struct110(a0)
      call struct111(a0)
   C code for mishin's spline potential method
   if (potmethod.eq.1) then
      c number of points for pair potential, electron density and then embedding energy
      a=8
      b=6
      c=5
      C number of parameters to minimize
      m=a+b+c-5
      C maximum number of points for pair potential, electron density and then embedding energy
      amax=9
      bmax=8
C Open File to write Important information From the potential Fitting Process
    open (28,file='potfit.txt')
    open (95,file='srsolvefit.txt')

C Fit Spline to Fitting Database Using Optimization Algorithm for initial a,b,c
    write (*,*) 'starting first fitting using optimization algorithm'
    write (28,*) 'starting first fitting using optimization algorithm'
    call optimize(sr)
    write (28,*), 'fitting sr is', sr
    aold=a
    bold=b
    cold=c

C Open File to write important information about Sr Solve TEST
    open (27,file='srsolvetest.txt')

C Calculate the correct rcut for srsolvetest
    open (29,file=element(1)//'.den')
    do 20 i=1,3000
        read (29,*), dummy1,dummy2
        if (i .eq. 3000) rcut=dummy1
    20 continue
    close (29)

C Find initial testing Sr
    call srsolvetest(srtestold,testnum)
    write (28,*), 'First Test Sr is', srtestold
    write (28,*), 'funtion written for a=', a, ', b=', b, ', c=', c
    call functionfolderwriter
    call lammpspot
    stop

C Begining of iterative method to find the highest a, b , c values
C which still decreases srtest.
    1       atest=1
    btest=1
    ctest=1
C if a,b,c are at max value, no further decrease can be made,
C run one last fit for max values and than go to section to write
final c functions and end potential fitting procedure.
    if (aold .eq. amax .and. bold .eq. bmax .and. cold .eq. cmax) then
write (*,*) 'a,b,and c maxed out write final functions and end fitting procedure'
write (28,*) 'a,b,and c maxed out write final functions and end fitting procedure'
goto 5
endif

C if a is at max value set atest=0 and skip to btest otherwise increase a
C and calculate srfit and then srtest if srfit is below 1.
if (a .eq. amax) then
write (28,*)
write (*,*) 'a at max skip to b'
write (28,*) 'a at max skip to b'
atest=0
else
a = aold+1
m = a+b+c-5
write (28,*)
write (*,*) 'a increased by 1 and than fit with an optimization algorithm'
write (28,*) 'a increased from', aold, ' to', a, ' then fit with an optimization algorithm'
call optimize(sr)
write (28,*) 'fitting sr from a being increased is', sr
if (sr .ge. 20) then
write (*,*) 'sr bad dont use fitted potential, atest set to 0'
write (28,*) 'sr bad dont use fitted potential, atest set to 0'
atest=0
a = aold
goto 2
else
C Calculate the correct rcut for srsolvetest
open (29,file=element(1)//'.den')
do 30 i=1,3000
read (29,*) dummy1,dummy2
if (i .eq. 3000) rcut=dummy1
30 continue
close (29)
call srsolvetest(sratest,testnum)
write (28,*) 'testing sr from a being increased is', sratest
write (28,*) 'funtion written for a=', a, ',', b=', b, ',', c=', c
call functionfolderwriter
C if b is at max value set btest=0 and skip to ctest otherwise increase b
C and calculate srfit and then srttest if srfit is below 1.
2  if (b .eq. bmax) then
   write (28,*)
   write (*,*) 'b at max skip to c'
   write (28,*) 'b at max skip to c'
   btest=0
else
   b=bold+1
   m=a+b+c-5
   write (28,*)
   write (*,*) 'b increased by 1 and than fit with an optimization algorithm'
   write (28,*) 'b increased from', bold,' to', b,' then fit with an optimization algorithm'
   call optimize(sr)
   write (28,*) 'fitting sr from b being increased is', sr
   if (sr .ge. 20) then
      write (*,*) 'sr bad dont use fitted potential, btest set to 0'
      write (28,*) 'sr bad dont use fitted potential, btest set to 0'
      btest=0
      b=bold
      goto 3
   else
      C Calculate the correct rcut for srsolvetest
      open (29,file=element(1)//'.den')
      do 40 i=1,3000
         read (29,*) dummy1,dummy2
         if (i .eq. 3000) rcut=dummy1
         continue
      40  continue
      close (29)
      call srsolvetest(srbtest,testnum)
      write (28,*) 'testing sr from b being increased is', srbtest
      write (28,*) 'funtion written for a=', a, ', b=', b, ', c=', c
      call functionfolderwriter
   endif
   endif

b=bold
C if c is at max value set ctest=0 and skip to checking a,b, and ctests
C otherwise increase c and calculate srfit and then srtest if srfit is below 1.
3  if (c .eq. cmax) then
   write (28,*)
   write (*,*) 'c at max skip to next part of code'
   write (28,*) 'c at max skip to next part of code'
   ctest=0
else
   c=cold+1
   m=a+b+c-5
   write (28,*)
   write (*,*) 'c increased by 1 and than fit with an
   optimization algorithm'
   write (28,*) 'c increased from', cold, ' to', c,' then
   fit with an optimization algorithm'
   call optimize(sr)
   write (28,*) 'fitting sr from c being increased is',
   sr
   c   if (sr .ge. 20) then
   c     write (*,*) 'sr bad dont use fitted potential,'
   ctest set to 0'
   c   write (28,*) 'sr bad dont use fitted potential,'
   ctest set to 0'
   c   ctest=0
   c   c=cold
   c   goto 4
   c else
C Calculate the correct rcut for srsolvetest
   open (29,file=element(1)/'.den')
   do 50 i=1,3000
      read (29,*) dummy1,dummy2
      if (i .eq. 3000) rcut=dummy1
50   continue
   close (29)
   call srsolvetest(srctest,testnum)
   write (28,*) 'testing sr from c being increased
   is', srctest
   write (28,*) 'funtion written for a=', a, ','
   b=' , b, ',', c
   call functionfolderwriter
   c endif
   c=cold
endif

C before finding srtstnew make sure a,b and ctest are not equal to 0
C if they are equal to 0 than use functions from previous test
4  if (atest .eq. 0 .and. btest .eq. 0 .and. ctest .eq. 0) goto 5

C beginning method to find srtestnew (average of fit and test sr)
C use code inside the if statement below if the three
srsolvetests above were done
   if (atest .ne. 0 .and. btest .ne. 0 .and. ctest .ne. 0) then
      if (sratest .le. srbtest) then
         if (sratest .le. srctest) then
            C sratest is the lowest
            call letterchoozer('a',srtestnew)
         else
            C srctest is the lowest
            call letterchoozer('c',srtestnew)
         endif
 Sharia
C srctest is the lowest
   elseif (srbtest .le. srctest) then
      C srbtest is the lowest
      call letterchoozer('b',srtestnew)
   else
      C srctest is the lowest
      call letterchoozer('c',srtestnew)
   endif

C if only one srsolvetest was done, set srtestnew to the one
completed
C if only two srsolvetests were done, set the lowest value to
srtestnew
   elseif (atest .eq. 0) then
      if (btest .eq. 0) then
         C srctest is the lowest
         call letterchoozer('c',srtestnew)
      elseif (ctest .eq. 0) then
         C srctest is the lowest
         call letterchoozer('c',srtestnew)
      endif
      elseif (sratest .eq. 0) then
         C sratest is the lowest
         call letterchoozer('a',srtestnew)
      elseif(sratest .le. srctest) then
         C sratest is the lowest
         call letterchoozer('a',srtestnew)
      else
         C srctest is the lowest
         call letterchoozer('c',srtestnew)
      endif
   elseif (btest .eq. 0) then
      if (ctest .eq. 0) then
         C srctest is the lowest
         call letterchoozer('c',srtestnew)
      elseif(sratest .eq. 0) then
         C sratest is the lowest
         call letterchoozer('a',srtestnew)
      elseif(sratest .le. srctest) then
         C sratest is the lowest
         call letterchoozer('a',srtestnew)
call letterchoozer('a',srtestnew)
else
C srtest is the lowest
call letterchoozer('c',srtestnew)
endif
elseif (ctest .eq. 0) then
  if (sratest .le. srtest) then
    C sratest is the lowest
    call letterchoozer('a',srtestnew)
  else
    C srtest is the lowest
    call letterchoozer('b',srtestnew)
  endif
endif
!
write (*,*), 'srtestnew is', srtestnew
write (*,*), 'the old values for a, b, c are', aold, bold, cold
write (*,*), 'the new values for a, b, c are', a, b, c
write (28,*), 'srtestnew is', srtestnew
write (28,*), 'the old values for a, b, c are', aold, bold, cold
write (28,*), 'the new values for a, b, c are', a, b, c
!
C if srtestnew is greater than srtestold, set a, b and c to aold, bold and cold
C than end the iterative method by skipping to label 5
if (srtestnew .gt. srtestold) then
  a=aold
  b=bold
  c=cold
  goto 5
else
C Set a, b, c old and sroldtest to new values of a, b, c and srnewtest
C and go back to beginig of routine (label 1)
aold=a
bold=b
cold=c
srtestold=srtestnew
write (*,*), 'go back to beginig of routine and do another iteration'
write (28,*)
write (28,*), 'go back to beginig of routine and do another iteration'
goto 1
endif
5 write (28,*) 'funtion written from a=', a, ', b=', b, ', c=', c,
c call functionwriter
write (*,*) 'end of fitting routine'
write (*,*) 'running properties one more time to get final
values'
C switch set to 0 so rose et al. will not run
switch=0
C Calculate the correct rcut for srsolvefit and srsolvetest
 open (29,file=element(1)//'.den')
do 10 i=1,3000
   read (29,*) dummy1,dummy2
   if (i .eq. 3000) rcut=dummy1
10 continue
 close (29)
C create a Lammps potential from the chosen potential
call lammpspot
C recalculate the fit and test properties to make sure the
functions correct.
call srsolvefit(sr)
call srsolvetest(sr,testnum)
close (27)
close (28)
close (95)
C ending mishin method
endif
C ending pure metal
endif
end

H.2 Building a BCC structure (bccstruct.f)

subroutine bccstruct(latticeconstant)

  implicit none
  c Progam variables
    double precision latticeconstant
  c Local Variables
    double precision perfect(250,3)
    integer i,j,k,l
  c this code makes a perfect fcc crystal which is a 5x5x5 super
cell. This super cell consists of 500 atoms.
  C the output file is called fcc.xyz. The output structures first
line contains the number of atoms in the structure file. The rest
of the lines are atomic positions in x,y,z order.
    l=1
  C this do loop creates a structure with an atom at the corner
  (i,j,k)*latticeconstant with one atom .5 latticecotant in x dir,
one atom .5 lattice constant in y dir and one atom .5 lattice
count in z dir.
  do 10 i=0,4
    do 20 j=0,4
      do 30 k=0,4
        perfect(l,1)=i*latticeconstant
        perfect(l,2)=j*latticeconstant
        perfect(l,3)=k*latticeconstant
        l=1+l
        perfect(l,1)=i*latticeconstant+.5*latticeconstant
        perfect(l,2)=j*latticeconstant+.5*latticeconstant
        perfect(l,3)=k*latticeconstant+.5*latticeconstant
        l=l+1
      30 continue
    20 continue
  10 continue
l=0
C this do loop moves the atoms over to the left and back and down
by 2 lattice units and writes the information into a file
  c bcc.xyz
    open (1,file='bcc.xyz')
    write (1,*) 250
    do 40 l=1,250
      perfect(l,1)=perfect(l,1)-2*latticeconstant
      perfect(l,2)=perfect(l,2)-2*latticeconstant
      perfect(l,3)=perfect(l,3)-2*latticeconstant
      if (abs(perfect(l,1)) .lt. 1e-14) perfect(l,1)=0
      if (abs(perfect(l,2)) .lt. 1e-14) perfect(l,2)=0
      if (abs(perfect(l,3)) .lt. 1e-14) perfect(l,3)=0
      write (1,*) perfect(l,1), perfect(l,2), perfect(l,3)
 40 continue
    close (1)
    return
  end

H.3 Shifting Layers in a Crystal (crystalcreation.f)

subroutine crystalcreation(const1,m,crystal,atoms)
C global variables
  integer satoms, indexatoms(15), layers
  common/specialatomcount/satoms, indexatoms, layers
  double precision index(15,150)
  common/indexing/index
C program Variables
integer m, atoms
double precision const1(m), crystal(atoms, 3)
C local Variables
    integer i, j, k, l

C move and place each layer of atoms into the final crystal.
C the movement in the z direction of each layer is controlled by
C const1(i)
C The atoms in each layer are defined by index and indexatoms
C where index corresponds to a row in the original crystal
C and indexatoms is number of atoms in each layer
C layers is the number of layers in both the original and final
crystal.

    do 5 i=1, layers
        if (const1(i) .eq. 0d0) goto 5
        do 10 j=1, indexatoms(i)
            crystal(index(i, j), 3) = crystal(index(i, j), 3) + const1(i)
        10    continue
    5  continue
return
end

H.4 Matrix Multiplication (dmvcalc.f)

    subroutine crystalcreation(const1, m, crystal, atoms)
C global variables
    integer satoms, indexatoms(15), layers
    common/specialatomcount/satoms, indexatoms, layers
    double precision index(15, 150)
    common/indexing/index
C program Variables
    integer m, atoms
    double precision const1(m), crystal(atoms, 3)
C local Variables
    integer i, j, k, l

C move and place each layer of atoms into the final crystal.
C the movement in the z direction of each layer is controlled by
C const1(i)
C The atoms in each layer are defined by index and indexatoms
C where index corresponds to a row in the original crystal
C and indexatoms is number of atoms in each layer
C layers is the number of layers in both the original and final
crystal.

    do 5 i=1, layers
        if (const1(i) .eq. 0d0) goto 5
        do 10 j=1, indexatoms(i)
            crystal(index(i, j), 3) = crystal(index(i, j), 3) + const1(i)
        10    continue
    5  continue
return
end
do 10 j=1,indexatoms(i)
crystal(index(i,j),3)=crystal(index(i,j),3)+const1(i)
10  continue
5  continue
return
derend

H.5 Dot Multiplication (dot.f)

subroutine dot(n, a, b, c)
* .. Scalar Arguments ..
   INTEGER n
   double precision a(n), b(n), c
* local variables
   integer i

* takes vector a and vector b both of size n and calculates
* the dot product.
* this summation is accomplished by using a simple do loop
   c=0
   do 1 i = 1,n
      c = c + a(i)*b(i)
1  continue
return
derend

H.6 Calculating Elastic Constants (elastic.f)

C****************************************************************
************
C       A program to determine Stress, RSS, Det(Q(n))
C       Input: Deformation gradient, Crystal orientation
C
C****************************************************************
************
subroutine elastic
IMPLICIT REAL*8 (A-H,O-Z)

PARAMETER(NSLIP=12)

PARAMETER(ZERO=0.D0, ONE=1.D0, ONE_HALF=0.5D0, TWO=2.D0,
+       ONE_THIRD=1.D0/3.D0, TWO_THIRD=2.D0/3.D0,
+       THREE_HALF=1.5D0, THREE=3.D0)

double precision c11,c12,c44,e_vacancy
REAL*8 AIDENT(3,3), Q(3,3),
SMAT_ALL(NSLIP,3),
+ ELMAT_C(3,3,3,3), ELMAT_G_ALL(3,3,3,3),
+ F_TAU(3,3), F_TAU_INV(3,3), R_TAU(3,3),
+ U_TAU(3,3), AUX1(3,3), AUX2(3,3),
+ U_TAU_INV(3,3),
+ AMAT(3,3), FSTAR_TAU(3,3),
+ E_TAU(3,3), TSTAR_TAU(3,3), TSTAR_TAU_VEC(6),
+ S_ALPHA0(3,3),
+ RSS_TAU(NSLIP),
+ C_ALPHA(3,3), CMAT_VEC(NSLIP,6),
+ T_TAU(3,3), DEV(3,3),
+ STRESS(3,3),
+ S(6), H(9),
+ Q_TRANS(3,3), TEMP1(3,3), TEMP2(3,3), T_TAU_C(3,3),
+ F_TAU_C(3,3),
+ AMV_GLOBAL(NSLIP,3), ANV_GLOBAL(NSLIP,3),
+ AMV_GLOBAL_CU(NSLIP,3), ANV_GLOBAL_CU(NSLIP,3),
+ T_TAU_VEC(6), PMAT_C_VEC(NSLIP,6), PMAT_C(NSLIP,3,3),
+ NSLIPSYS(6,NSLIP)

real*8
ABCH_a0, ABCH_a1, ABCH_a2, ABCH_a3, ABCH_a4, ABCH_a5, ABCH_a6
real*8 ABCH_aaa1, ABCH_aaa2, ABCH_rr1, ABCH_rr2
real*8 ABCH_r1, ABCH_r2, ABCH_r3, ABCH_r4, ABCH_r5, ABCH_r6
real*8 RCUT, STRESS_IN_GPA
real*8 DS(43,3)
real*8 C(3,3,3,3), C_G(3,3,3,3), DETERMINANT, DETnln(NSLIP)

real*8 pair1(3000), pair2(3000), pair_der2(3000)
real*8 den1(3000), den2(3000), den_der2(3000)
real*8 embed1(3000), embed2(3000), embed_der2(3000)
common/sp_pair/pair1, pair2, pair_der2
common/sp_den/den1, den2, den_der2
common/sp_embed/embed1, embed2, embed_der2

COMMON/pot_fitting/a0, rcut
common/fit/c11, c12, c44, e_vacancy

DATA NSLIPSYS/1, 1, 1, -2, 1, 1,
& 1, 1, 1, 1, -2, 1,
& 1, 1, 1, 1, 1, -2,
& -1, 1, 1, 2, 1, 1,
& -1, 1, 1, 1, 2, -1,
& -1, 1, 1, 1, -1, 2,
& 1, -1, 1, 2, 1, -1,
& 1, -1, 1, 1, 2, 1,
& 1, -1, 1, -1, 1, 2,
& 1, 1, -1, 2, -1, 1,
& 1, 1, -1, 1, -2, -1,
& 1, 1, -1, 1, 1, 2/

C potential parameters
C Ni: Voter-Chen; E_coh = 4.45eV
C \( a_0 = 3.52 \)
C \( r_{cut} = 4.7895 \)
call `loadtable`

\[
\begin{align*}
\text{PH} &= 0 \\
\text{TH} &= 0 \\
\text{OM} &= 0
\end{align*}
\]

C CONVERT ANGLES INTO RADIANS

\[
\text{PI} = 3.1415926
\]

\[
\text{TH} = \text{TH} \cdot \text{PI}/180.
\]

\[
\text{PH} = \text{PH} \cdot \text{PI}/180.
\]

\[
\text{OM} = \text{OM} \cdot \text{PI}/180.
\]

C Calculate the rotation matrix [Q]
C

CALL `ROTMAT(TH,PH,OM,Q)`
CALL `MTRANS(Q,Q_TRANS)`

C write(*,*) 'Rotation, Q'

C Do 5 I = 1, 3
C write(*,*) Q(I,1),Q(I,2),Q(I,3)
C5 Continue

AIDENT(1,1) = ONE
AIDENT(1,2) = ZERO
AIDENT(1,3) = ZERO
AIDENT(2,1) = ZERO
AIDENT(2,2) = ONE
AIDENT(2,3) = ZERO
AIDENT(3,1) = ZERO
AIDENT(3,2) = ZERO
AIDENT(3,3) = ONE
Normalize the slip system vectors

\[
\begin{align*}
\text{DO } & 10 \ I = 1,\text{NSLIP} \\
\text{AMV}_C(I,1) & = \text{NSLIPSYS}(4,I) \\
\text{AMV}_C(I,2) & = \text{NSLIPSYS}(5,I) \\
\text{AMV}_C(I,3) & = \text{NSLIPSYS}(6,I) \\
\text{ANV}_C(I,1) & = \text{NSLIPSYS}(1,I) \\
\text{ANV}_C(I,2) & = \text{NSLIPSYS}(2,I) \\
\text{ANV}_C(I,3) & = \text{NSLIPSYS}(3,I) \\
\text{AMV}_C\_MAG & = \text{DSQRT}((\text{AMV}_C(I,1)\_MAG)^2 + (\text{AMV}_C(I,2)\_MAG)^2 + (\text{AMV}_C(I,3)\_MAG)^2) \\
\text{ANV}_C\_MAG & = \text{DSQRT}((\text{ANV}_C(I,1)\_MAG)^2 + (\text{ANV}_C(I,2)\_MAG)^2 + (\text{ANV}_C(I,3)\_MAG)^2) \\
\text{AMV}_C(I,1) & = \text{AMV}_C(I,1)/\text{AMV}_C\_MAG \\
\text{AMV}_C(I,2) & = \text{AMV}_C(I,2)/\text{AMV}_C\_MAG \\
\text{AMV}_C(I,3) & = \text{AMV}_C(I,3)/\text{AMV}_C\_MAG \\
\text{ANV}_C(I,1) & = \text{ANV}_C(I,1)/\text{ANV}_C\_MAG \\
\text{ANV}_C(I,2) & = \text{ANV}_C(I,2)/\text{ANV}_C\_MAG \\
\text{ANV}_C(I,3) & = \text{ANV}_C(I,3)/\text{ANV}_C\_MAG \\
10 & \text{ CONTINUE}
\end{align*}
\]

Change slip systems from crystal basis to global basis

\[
\begin{align*}
\text{DO } & 20 \ ISLIP = 1,\text{NSLIP} \\
\text{DO } & 21 \ I = 1,3 \\
\text{AMV\_GLOBAL}(ISLIP,I) & = Q(I,1)\_\text{AMV\_C}(ISLIP,1) + Q(I,2)\_\text{AMV\_C}(ISLIP,2) + Q(I,3)\_\text{AMV\_C}(ISLIP,3) \\
\text{ANV\_GLOBAL}(ISLIP,I) & = Q(I,1)\_\text{ANV\_C}(ISLIP,1) + Q(I,2)\_\text{ANV\_C}(ISLIP,2) + Q(I,3)\_\text{ANV\_C}(ISLIP,3) \\
21 & \text{ Continue}
\end{align*}
\]

AMV slip direction

\[
\text{write}(\_\_\_*,\_\_\_*) \text{ AMV\_GLOBAL}(ISLIP,1),\text{AMV\_GLOBAL}(ISLIP,2),\text{AMV\_GLOBAL}(ISLIP,3)
\]
AMV_GLOBAL(ISLIP,3)
CONTINUE

Copy the old and new Deformation gradients into $F_T$ and $F_{TAU}$ respectively

$$F_{TAU}(1,1) = 1$$
$$F_{TAU}(1,2) = 0$$
$$F_{TAU}(1,3) = 0$$
$$F_{TAU}(2,1) = 0$$

$$F_{TAU}(2,2) = 1$$
$$F_{TAU}(2,3) = 0$$
$$F_{TAU}(3,1) = 0$$
$$F_{TAU}(3,2) = 0$$
$$F_{TAU}(3,3) = 1$$

Express $F_{TAU}$ in crystal basis

CALL MPROD(Q_TRANS,F_TAU,TEMP1)
CALL MPROD(TEMP1,Q,F_TAU_C)

Calculate Cauchy stress and modulus

$$H(1) = F_{TAU}_C(1,1)$$
$$H(2) = F_{TAU}_C(2,1)$$
$$H(3) = F_{TAU}_C(3,1)$$
$$H(4) = F_{TAU}_C(1,2)$$
$$H(5) = F_{TAU}_C(2,2)$$
$$H(6) = F_{TAU}_C(3,2)$$
$$H(7) = F_{TAU}_C(1,3)$$
$$H(8) = F_{TAU}_C(2,3)$$
$$H(9) = F_{TAU}_C(3,3)$$

Call STRESSMISHIN(H,S,C)

$$T_{TAU}_C(1,1) = S(1) \times 1.E9/100E18$$
$$T_{TAU}_C(2,2) = S(2) \times 1.E9/100E18$$
$$T_{TAU}_C(3,3) = S(3) \times 1.E9/100E18$$
$$T_{TAU}_C(2,3) = S(4) \times 1.E9/100E18$$
$$T_{TAU}_C(1,3) = S(5) \times 1.E9/100E18$$
$$T_{TAU}_C(1,2) = S(6) \times 1.E9/100E18$$
T_TAU_C(3,2) = T_TAU_C(2,3)
T_TAU_C(3,1) = T_TAU_C(1,3)
T_TAU_C(2,1) = T_TAU_C(1,2)

C
C Express Cauchy stress and modulus in global basis
C

CALL MPROD(Q,T_TAU_C,TEMP)
CALL MPROD(TEMP,Q_TRANS,T_TAU)
DO 141 III=1,3
    DO 141 JJJ=1,3
        DO 141 KKK=1,3
            DO 141 LLL=1,3
                C_G(III,JJJ,KKK,LLL) = 0.
                DO 142 M1=1,3
                    DO 142 M2=1,3
                        DO 142 M3=1,3
                            DO 142 M4=1,3
                                C_G(III,JJJ,KKK,LLL) = C_G(III,JJJ,KKK,LLL)
                                +Q(III,M1) *Q(JJJ,M2) *Q(KKK,M3) *Q(LLL,M4) *C(M1,M2,M3,M4)
                                CONTINUE
                            CONTINUE
                        CONTINUE
                    CONTINUE
                CONTINUE
            CONTINUE
        CONTINUE
    CONTINUE
DO 141 CONTINUE
C_G(III,JJJ,KKK,LLL) = C_G(III,JJJ,KKK,LLL)
     * 1.E9/100E18
141 CONTINUE

C11=C_G(1,1,1,1)/1.e9*1e20
C12=C_G(1,1,2,2)/1.E9*1e20
C44=C_G(1,2,1,2)/1.E9*1e20

RETURN
END

C****************************************************************
************

SUBROUTINE MATINV(A,A_INV,DET_A_INV)
C
C This subroutine calculates the inverse of a {3 x 3} matrix and the
derminant of the inverse
C
C****************************************************************
************

IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION A(3,3), A_INV(3,3)

PARAMETER(ZERO=0.D0, ONE=1.D0)

DET_A = A(1,1)*(A(2,2)*A(3,3) - A(3,2)*A(2,3)) -
       A(2,1)*(A(1,2)*A(3,3) - A(3,2)*A(1,3)) +
       A(3,1)*(A(1,2)*A(2,3) - A(2,2)*A(1,3))

IF (DET_A .LE. ZERO) THEN
  WRITE(80,*) 'WARNING: SUBROUTINE MATINV:'
  WRITE(80,*) 'WARNING: DET of MAT is zero/negative!!'
  WRITE(80,*) 'DET_A = ',DET_A
ENDIF

DET_A_INV = ONE/DET_A

A_INV(1,1) = DET_A_INV*(A(2,2)*A(3,3) - A(3,2)*A(2,3))
A_INV(1,2) = DET_A_INV*(A(3,2)*A(1,3) - A(1,2)*A(3,3))
A_INV(1,3) = DET_A_INV*(A(1,2)*A(2,3) - A(2,2)*A(1,3))
A_INV(2,1) = DET_A_INV*(A(3,1)*A(2,3) - A(2,1)*A(3,3))
A_INV(2,2) = DET_A_INV*(A(1,1)*A(3,3) - A(3,1)*A(1,3))
A_INV(2,3) = DET_A_INV*(A(2,1)*A(1,3) - A(1,1)*A(2,3))
A_INV(3,1) = DET_A_INV*(A(2,1)*A(3,2) - A(3,1)*A(2,2))
A_INV(3,2) = DET_A_INV*(A(3,1)*A(1,2) - A(1,1)*A(3,2))
A_INV(3,3) = DET_A_INV*(A(1,1)*A(2,2) - A(2,1)*A(1,2))

RETURN
END

C****************************************************************
************
SUBROUTINE ROTMAT(TH, PH, OM, Q)
C****************************************************************
************

IMPLICIT REAL*8 (A-H, O-Z)
DIMENSION Q(3,3)

STH = DSIN(TH)
CTH = DCOS(TH)

SPH = DSIN(PH)
CPH = DCOS(PH)

SOM = DSIN(OM)
COM = DCOS(OM)

Q(1,1) = CPH*COM - SPH*SOM*CTH

278
\[ Q(1,2) = \text{SPH} \cdot \text{COM} + \text{CPH} \cdot \text{SOM} \cdot \text{CTH} \]
\[ Q(1,3) = \text{SOM} \cdot \text{STH} \]
\[ Q(2,1) = -\text{CPH} \cdot \text{SOM} - \text{SPH} \cdot \text{COM} \cdot \text{CTH} \]
\[ Q(2,2) = -\text{SPH} \cdot \text{SOM} + \text{CPH} \cdot \text{COM} \cdot \text{CTH} \]
\[ Q(2,3) = \text{COM} \cdot \text{STH} \]
\[ Q(3,1) = \text{SPH} \cdot \text{STH} \]
\[ Q(3,2) = -\text{CPH} \cdot \text{STH} \]
\[ Q(3,3) = \text{CTH} \]

RETURN
END

C****************************************************************
************
SUBROUTINE PRTMAT(A,M,N)
C
Print the matrix A of dimension \{m x n\}
C****************************************************************
************
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION A(M,N)
 DO 10 I = 1,M
 c WRITE(80,*) (A(I,J),J=1,N)
10 CONTINUE
RETURN
END

C****************************************************************
************
SUBROUTINE MTRANS(A,ATRANS)
C
THIS SUBROUTINE CALCULATES THE TRANSPOSE OF AN 3 BY 3
C MATRIX \[A\], AND PLACES THE RESULT IN ATRANS.
C****************************************************************
************
REAL*8 A(3,3), ATRANS(3,3)
 DO 1 I=1,3
 DO 1 J=1,3
 ATRANS(J,I) = A(I,J)
1 CONTINUE
**SUBROUTINE** MPROD(A,B,C)

C THIS SUBROUTINE MULTIPLIES TWO 3 BY 3 MATRICES [A] AND [B], 
C AND PLACE THEIR PRODUCT IN MATRIX [C].

C****************************************************************
*************

REAL*8 A(3,3),B(3,3),C(3,3)

DO 2 I = 1, 3
   DO 2 J = 1, 3
      C(I,J) = 0.D0
      DO 1 K = 1, 3
         C(I,J) = C(I,J) + A(I,K) * B(K,J)
      1 CONTINUE
   2 CONTINUE

CRETURN
END

C****************************************************************
******
C C       User subroutine for instability criterion
C
C SUBROUTINE STABILITY(C,STRESS,ANV_GLOBAL_CU,DETnln)

REAL*8 C(3,3,3,3),STRESS(3,3)
REAL*8 DELTA(3,3),ANV_GLOBAL_CU(12,3)
REAL*8 ALnln(3,3), DETnln(12)

DO 1200 I1=1,3
   DO 1200 J1=1,3
      IF(I1.EQ.J1) THEN
         DELTA(I1,J1) = 1.
      ELSE
         DELTA(I1,J1) = 0.
      ENDIF
   1200 CONTINUE

DO 1250 ISLIP=1,12
DO 1260 I2=1,3
DO 1260 I3=1,3
   ALnln(I2,I3) = 0.
1260   CONTINUE

DO 1270 J1=1,3
   DO 1270 K1=1,3
   DO 1270 I1=1,3
   DO 1270 L1=1,3
   ALnln(J1,K1) = ALnln(J1,K1)+(C(I1,J1,K1,L1)+STRESS(I1,L1)
*DELTA(J1,K1))*ANV_GLOBAL_CU(ISLIP,I1)*ANV_GLOBAL_CU(ISLIP,L1)
1270   CONTINUE

CALL DETER(ALnln,DETnln(ISLIP))

if(ISLIP.eq.9) then
   write(*,*) 'nln matrix'
   DO 1280 I2=1,3
   write(*,*) ALnln(I2,1),ALnln(I2,2),ALnln(I2,3)
1280   CONTINUE
endif
1250   CONTINUE
RETURN
END

SUBROUTINE DETER(AL,DETERMINANT)

REAL*8  AL(3,3),A(6,3)
REAL*8  APLUS,AMINUS,PRODUCT,DETERMINANT

M = 3

DO 20 I=1,M
   DO 20 J=1,M
   A(I,J) = AL(I,J)
   A(I+M,J) = AL(I,J)
20   CONTINUE

APLUS = 0.
DO 30 I=1,M
   PRODUCT = 1.
   DO 35 J=1,M
      PRODUCT = PRODUCT*A(I-1+J,J)
   CONTINUE
   APLUS = APLUS + PRODUCT
30 CONTINUE

AMinus = 0.

DO 40 I=1,M
   PRODUCT = 1
   DO 45 J=1,M
      PRODUCT = PRODUCT*A(I-1+J,M+1-J)
   CONTINUE
   AMinus = AMinus + PRODUCT
40 CONTINUE

DETERMINANT = APLUS - AMinus

END

C****************************************************************
*  Mishin Cu potential
*  Load in spline points
C****************************************************************

subroutine loadtable

IMPLICIT REAL*8 (A-H,O-Z)
real*8  pair1(3000),pair2(3000),pair_der2(3000)
real*8  den1(3000),den2(3000),den_der2(3000)
real*8  embed1(3000),embed2(3000),embed_der2(3000)
common/sp_pair/pair1,pair2,pair_der2
common/sp_den/den1,den2,den_der2
common/sp_embed/embed1,embed2,embed_der2
   integer elenum, fitmethod, potmethod
   character*2 element(6)
   common/control/elenum, element, fitmethod, potmethod
CHARACTER*80 FILE1, FILE2, FILE3
integer i,n
real*8 yp1,ypn

C  file1 = 'cu.pair'
C  file2 = 'cu.den'
C  file3 = 'cu.embed'
open(65,file=element(1)//'.'pair')
n = 3000
do 5 i=1,n
   read(65,*) pair1(i),pair2(i)
   read(70,*) den1(i),den2(i)
   read(75,*) embed1(i),embed2(i)
5 continue
close(65)
close(70)
close(75)

cy1 = 1e32
yn = 1e32
call spline(pair1,pair2,n,yp1,yn,pair_der2)
call spline(den1,den2,n,yp1,yn,den_der2)
call spline(embed1,embed2,n,yp1,yn,embed_der2)
return
end

C****************************************************************
* Mishin Cu potential
C Stress and Modulus calculation
C****************************************************************
*
subroutine stressmishin(H,S,C)

IMPLICIT REAL*8 (A-H,O-Z)
dimension pair1(3000),pair2(3000),pair_der2(3000)
dimension den1(3000),den2(3000),den_der2(3000)
dimension embed1(3000), embed2(3000), embed_der2(3000)
C dimension H(9), ds(3,70), dx(70,4), S(6)
C dimension v(70), dvdr(70), d2vdr2(70)
C dimension x(70), dxdr(70), d2xdr2(70)
dimension H(9), ds(3,43), dx(43,4), S(6)
dimension v(43), dvdr(43), d2vdr2(43)
dimension x(43), dxdr(43), d2xdr2(43)
dimension B(3,3), BL(3,3), C(3,3,3,3), T(3,3)

common/sp_pair/pair1, pair2, pair_der2
common/sp_den/den1, den2, den_der2
common/sp_embed/embed1, embed2, embed_der2

COMMON/pot_fitting/a0,rcut

data ds / 0., 5., 5.,
* .5, 0., 5.,
* .5, 5., 0.,
* 0., 5., -.5,
* .5, 0., -.5,
* .5, -.5, 0.,
* 1., 0., 0.,
* 0., 1., 0.,
* 0., 0., 1.,
* 1., 5., 5.,
* .5, 1., 5.,
* .5, 5., 1.,
* 1., 5., -.5,
* .5, 1., -.5,
* .5, -.5, 1.,
* -1., 5., 5.,
* .5, -1., 5.,
* .5, 5., -1.,
* -1., 5., -.5,
* .5, -1., -.5,
* .5, -.5, -1.,
* 0., 1., 1.,
* 1., 0., 1.,
* 1., 1., 0.,
* 0., 1., -1.,
* 1., 0., -1.,
* 1., -1., 0.,
* 1.5, 5., 0.,
* -1.5, 5., 0.,
* 1.5, 0., .5,
* -1.5, 0., .5,
* .5, 1.5, 0.,
* .5, -1.5, 0.,
* 0, 1.5, .5,
c      rcut = 4.7895

  a0 = 3.52

USTRESS_IN_GPA = ((1.60217733e-19)/1e-30*1e-9)

n = 3000

neighbors = 43

volume = a0 * a0 * a0 *
*     abs(  H(1)*H(5)*H(9)-H(6)*H(8)) +
*           H(2)*H(6)*H(7)-H(4)*H(9)) +
*               H(3)*(H(4)*H(8)-H(7)*H(5)) )/4

do 10 i = 1,neighbors

    dx(i,1) = a0 *(ds(1,I)*H(1)+ds(2,I)*H(4)+ds(3,I)*H(7))
    dx(i,2) = a0 *(ds(1,I)*H(2)+ds(2,I)*H(5)+ds(3,I)*H(8))
    dx(i,3) = a0 *(ds(1,I)*H(3)+ds(2,I)*H(6)+ds(3,I)*H(9))
    dx(i,4) = sqrt(dx(I,1)*dx(I,1)+dx(I,2)*dx(I,2)+dx(I,
3)*dx(I,3))
10    continue

xsum = 0.

do 20 i = 1,neighbors

    if (dx(i,4).lt.rcut) then
        call splint(den1,den2,den_der2,n,dx(i,4),x(i))
        xsum = xsum + x(i)
    endif
20    continue

xsum = xsum*2
call splintder1(embed1,embed2,embed_der2,n,xsum,dudx)
call splintder2(embed1,embed2,embed_der2,n,xsum,d2udx2)

do 30 i =1,6
    S(I) = 0.
continue

do 35  i=1,3
   do 35  j=1,3
      B(i,j) = 0.
   continue

do 40  i = 1,neighbors
   if (dx(i,4).lt.rcut) then
      call splintder1(pair1,pair2,pair_der2,n,dx(i,4),dvdr(i))
      call splintder1(den1,den2,den_der2,n,dx(i,4),dxdr(i))

      call splintder2(pair1,pair2,pair_der2,n,dx(i,4),d2vdr2(i))
      call splintder2(den1,den2,den_der2,n,dx(i,4),d2xdr2(i))

      coeff = dvdr(i)+2*dudx*dxdr(i)

      S(1) = S(1) + coeff * dx(I,1) * dx(I,1) / dx(I,4)
      S(2) = S(2) + coeff * dx(I,2) * dx(I,2) / dx(I,4)
      S(3) = S(3) + coeff * dx(I,3) * dx(I,3) / dx(I,4)
      S(4) = S(4) + coeff * dx(I,2) * dx(I,3) / dx(I,4)
      S(5) = S(5) + coeff * dx(I,1) * dx(I,3) / dx(I,4)
      S(6) = S(6) + coeff * dx(I,1) * dx(I,2) / dx(I,4)

      B(1,1) = B(1,1) + dxdr(I)* dx(I,1) * dx(I,1) / dx(I,4)
      B(2,2) = B(2,2) + dxdr(I)* dx(I,2) * dx(I,2) / dx(I,4)
      B(3,3) = B(3,3) + dxdr(I)* dx(I,3) * dx(I,3) / dx(I,4)
      B(2,3) = B(2,3) + dxdr(I)* dx(I,2) * dx(I,3) / dx(I,4)
      B(1,3) = B(1,3) + dxdr(I)* dx(I,1) * dx(I,3) / dx(I,4)
      B(1,2) = B(1,2) + dxdr(I)* dx(I,1) * dx(I,2) / dx(I,4)
   endif

continue

S(1) = S(1) * USTRESS_IN_GPA / volume
S(2) = S(2) * USTRESS_IN_GPA / volume
S(3) = S(3) * USTRESS_IN_GPA / volume
S(4) = S(4) * USTRESS_IN_GPA / volume
S(5) = S(5) * USTRESS_IN_GPA / volume
S(6) = S(6) * USTRESS_IN_GPA / volume

B(2,1) = B(1,2)
B(3,1) = B(1,3)
B(3,2) = B(2,3)
do 50 ii=1,3
  do 50 jj=1,3
    do 50 kk=1,3
      do 50 ll=1,3
        C(ii,jj,kk,ll) = 0.
      50  continue
    do 80 i=1,neighbors
      if (dx(i,4).lt.rcut) then
        do 60 ii=1,3
          do 60 jj=1,3
            T(ii,jj) = dx(i,ii) * dx(i,jj) / dx(i,4)
            BL(ii,jj) = dxdr(i) * T(ii,jj)
          60  continue
        do 70 ii=1,3
          do 70 jj=1,3
            do 70 kk=1,3
              do 70 ll=1,3
                C(ii,jj,kk,ll) = C(ii,jj,kk,ll) +
                * 2*d2udx2*B(kk,ll)*2*BL(ii,jj) +
                * ((d2vdr2(i)-dvdr(i)/dx(i,4)) +
                * 2*dudx*(d2xdr2(i)-dxdr(i)/dx(i,4))) *
                * T(kk,ll) * T(ii,jj)
          70  continue
        endif
      80  continue
    do 90 II=1,3
      do 90 JJ=1,3
        do 90 KK=1,3
          do 90 LL=1,3
            C(ii,jj,kk,ll) = C(ii,jj,kk,ll) *
            * USTRESS_IN_GPA / volume
          90  continue
        return
      end
C
C Cubic Spline function from Numerical Recipe
C
SUBROUTINE spline(x,y,n,yp1,ypn,y2)

IMPLICIT REAL*8 (A-H,O-Z)
INTEGER n,NMAX
REAL*8 yp1,ypn,x(n),y(n),y2(n)
PARAMETER (NMAX=5000)
INTEGER i,k
REAL*8 p,qn,sig,un,u(NMAX)
if (yp1.gt..99e30) then
  y2(1)=0.
  u(1)=0.
else
  y2(1)=-0.5
  u(1)=(3./(x(2)-x(1)))*((y(2)-y(1))/(x(2)-x(1))-yp1)
endif
do 11 i=2,n-1
  sig=(x(i)-x(i-1))/(x(i+1)-x(i-1))
  p=sig*y2(i-1)+2.
  y2(i)=(sig-1.)/p
  u(i)=(6.*((y(i+1)-y(i))/(x(i+1)-x(i))-
     *(y(i)-y(i-1))/(x(i)-x(i-1)))-(y(i)-y(i-1))/(x(i+1)-x(i-1))-
     *u(i-1))/p
11 continue
if (ypn.gt..99e30) then
  qn=0.
  un=0.
else
  qn=0.5
  un=(3./(x(n)-x(n-1)))*ypn-(y(n)-y(n-1))/(x(n)-x(n-1))
endif
y2(n)=(un-qn*u(n-1))/(qn*y2(n-1)+1.)
do 12 k=n-1,1,-1
  y2(k)=y2(k)*y2(k+1)+u(k)
12 continue
return
END

c---- Subroutine splint is the Numerical Recipes sister routine
for the cubic spline routine, spline. It takes the data
points, xa and ya vectors of length n, and the second
derivatives computed by spline, y2a, and it computes the
y value for the specified x value.
c subroutine splint(xa,ya,y2a,n,x,y)
IMPLICIT REAL*8 (A-H,O-Z)
dimension xa(n),ya(n),y2a(n)
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.

return
end

subroutine splintder1(xa,ya,y2a,n,x,yder1)
IMPLICIT REAL*8 (A-H,O-Z)
dimension xa(n),ya(n),y2a(n)
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

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

yder1=-ya(klo)/h+ya(khi)/h+
&    (((-3*a**2/h+1./h)*y2a(klo)+(3*b**2/h-1./h)*y2a(khi)))*(h**2)/6.

return
end

subroutine splintder2(xa,ya,y2a,n,x,yder2)
IMPLICIT REAL*8 (A-H,O-Z)
dimension xa(n),ya(n),y2a(n)
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
yder2 = ((6*a/h**2)*y2(a(klo))+(6*b/h**2)*y2(a(khi)))*(h**2)

6.
return
end

H.7 Calculating the Equilibrium Structure at 293 K (eq293k.py)

#!/usr/bin/env python
#
#  read in improtant information from ./controls/control.txt
#  reads in important property information from ./
database/'[element names]'materialconst.txt
#  writes a file named in.eq293
#  Run Lammps and than Exits

f=open('controls/control.txt','r')
elenum=int(f.readline().rstrip('
'))
elements=''
elelist=[]
for i in range(elenum):
    elelist.append(f.readline().rstrip('
'))
    elements += '='+elelist[i]

f.close()

propfile='database/'+elements+'materialconst.txt'
f=open(propfile,'r')
a=f.read()  
a=a.split(',',)
latticesize=float(a[1])

# calculate the berendson pressure control parameter using the bulk modulus
bulkmod=a[4]
bulkmod=bulkmod.rstrip('
')
bulkmod=bulkmod.replace('d','e')
bulkmod=float(bulkmod)
cbulk=1.383e11  #copper bulk modulus
cberendson=100  #copper berendson pressure control parameter
berendson=bulkmod*cberendson/cbulk
f.close()

f=open('in.eq293','w')
f.write('''# bulk Cu lattice
log      log.eq293

variable x index 1
variable y index 1
variable z index 1

variable xx equal 20*$x
variable yy equal 20*$y
variable zz equal 20*$z

dimension 3
boundary p p p
units    metal
atom_style atomic''')

f.write('''
lattice     fcc {0}
'''.format(latticesize))

f.write('''
region      box block 0 ${xx} 0 ${yy} 0 ${zz}
create_box 1 box
create_atoms 1 box

pair_style eam/alloy''')

if elenum==0:
    f.write('''
pair_coeff  *  {0}.eam.alloy
{0}
'''.format(elements))
else:
    f.write('''
pair_coeff  *  {0}.eam.alloy'.format(elements))
    for i in range(elenum):
        if i==elenum-1:
            f.write(' {0}'.format(elelist[i]))
        else:
            f.write(' {0}'.format(elelist[i]))

f.write('''
#velocity all create 1 376847 loop geom
velocity all create 293 376847 loop geom

neighbor 1.0 bin
neigh_modify every 1 delay 5 check yes

timestep 0.001
#timestep 0.005
thermo 10

291
# stabilize pressure and temperature
fix       myfix all nve'''

f.write('fix

myfix all press/berendsen iso 1 1
{0}
.format(berendson))

f.write('''fix

mytfix all temp/berendsen 293 293 .1
run

5000
write_restart

293.equilibrium''')
f.close()

import os
os.system('mpirun -np 4 ../../lammps/src/lmp_openmpi<in.eq293

H.8 Building a FCC Structure (fccstruct.f)

subroutine fccstruct(latticeconstant)
imPLICIT none
c Program variables
  double precision latticeconstant
c Local Variables
  double precision perfect(500,3)
  integer i,j,k,l
c this code makes a perfect fcc crystal which is a 5x5x5 super cell. This super cell consists of 500 atoms.
C the output file is called fcc.xyz. The output structures first line contains the number of atoms in the structure file. The rest of the lines are atomic positions in x,y,z order.

  l=1
C this do loop creates a structure with an atom at the corner (i,j,k)*latticeconstant with one atom .5 latticeconstant in x dir, one atom .5 lattice constant in y dir and one atom .5 lattice constant in z dir.
    do 10 i=0,4
      do 20 j=0,4
        do 30 k=0,4
          perfect(l,1)=i*latticeconstant
          perfect(l,2)=j*latticeconstant
          perfect(l,3)=k*latticeconstant
          l=l+1
          perfect(l,1)=i*latticeconstant+
          5*latticeconstant
          perfect(l,2)=j*latticeconstant+
          5*latticeconstant
          perfect(l,3)=k*latticeconstant
          l=l+1
          perfect(l,1)=i*latticeconstant+
          5*latticeconstant
          292
5*latticeconstant
perfect(l,1)=i*latticeconstant
perfect(l,2)=j*latticeconstant+.5*latticeconstant
l=l+1
perfect(l,3)=k*latticeconstant+
5*latticeconstant

30 continue
20 continue
10 continue
l=0
C this do loop moves the atoms over to the left and back and down by 3 lattice units and writes the information into a file fcc.xyz
C
open (1,file='fcc.xyz')
c
write (*,*) 500
do 40 l=1,500
   perfect(l,1)=perfect(l,1)-2*latticeconstant
   perfect(l,2)=perfect(l,2)-2*latticeconstant
   perfect(l,3)=perfect(l,3)-2*latticeconstant
   if (abs(perfect(l,1)) .lt. 1e-14) perfect(l,1)=0
   if (abs(perfect(l,2)) .lt. 1e-14) perfect(l,2)=0
   if (abs(perfect(l,3)) .lt. 1e-14) perfect(l,3)=0
   write (1,*) perfect(l,1), perfect(l,2), perfect(l,3)
40 continue
C
close (1)
return
end

H.9 Finding the Fractional Part of a Number (fpart.f)

double precision function fpart(number)
C program variable
double precision number

C takes a number that is double precision and
C finds the fractional part of the number
C by using int operations
C
write (*,*) 'the number is', number
fpart=abs(number-int(number))
write (*,*) 'the fpart is', fpart
return
end
H.10 Stores the Potential in the Function Folder (functionfolderwriter.f)

subroutine functionfolderwriter
implicit none

C global Variables
integer a, b, c, m, n
common/parameters/a,b,c,m,n
integer elenum, fitmethod, potmethod
character*2 element(6)
common/control/elenum, element, fitmethod, potmethod

C local variables
character den*21, embed*23, pair*22, func*17
character shortden*6, shortembed*8, shortpair*7
double precision x,y
integer i

C write calculated functions in function folder  [is for single
element only]

C the functions in the function folder
func(1:12)=./functions/
func(13:14)=element(1)
func(15:15)=char(a+48)
func(16:16)=char(b+48)
func(17:17)=char(c+48)
den(1:17)=func
den(18:21)='.den'
embed(1:17)=func
embed(18:23)='.embed'
pair(1:17)=func
pair(18:22)='.pair'

C the functions in the main folder
shortden(1:2)=element(1)
shortden(3:6)='.den'
shortembed(1:2)=element(1)
shortembed(3:8)='.embed'
shortpair(1:2)=element(1)
shortpair(3:7)='.pair'

C open function files
open (20,file=den)
open (21,file=shortden)
open (22,file=embed)
open (23,file=shortembed)
open (24,file=pair)
open (25,file=shortpair)

do 10 i=1,3000
read (21,*) x,y
write (20,*) x,y
read (23,*) x,y
write (22,*) x,y
read (25,*) x,y
write (24,*) x,y

continue

close (20)
close (21)
close (22)
close (23)
close (24)
close (25)

return
end

H.11 Copies the Potential from the Function Folder to Main Folder
(functionwriter.f)

subroutine functionwriter
implicit none
C global Variables
integer a, b, c, m, n
common/parameters/a,b,c,m,n
integer elenum, fitmethod, potmethod
character*2 element(6)
common/control/elenum, element, fitmethod, potmethod
C local variables
character den*21 , embed*23 , pair*22, func*17
character shortden*6, shortembed*8, shortpair*7
double precision x,y
integer i

C write calculated functions in main folder [is for single element only]

C the functions in the function folder
func(1:12)= './functions/'
func(13:14)=element(1)
func(15:15)=char(a+48)
func(16:16)=char(b+48)
func(17:17)=char(c+48)
den(1:17)=func
den(18:21)= '.den'
embed(1:17)=func
embed(18:23)= '.embed'
pair(1:17)=func
pair(18:22)='.pair'

C the functions in the main folder
shortden(1:2)=element(1)
shortden(3:6)='.den'
shortembed(1:2)=element(1)
shortembed(3:8)='.embed'
shortpair(1:2)=element(1)
shortpair(3:7)='.pair'

C open function files
open (20,file=den)
open (21,file=shortden)
open (22,file=embed)
open (23,file=shortembed)
open (24,file=pair)
open (25,file=shortpair)

do 10 i=1,3000
   read (20,*) x,y
   write (21,*) x,y
   read (22,*) x,y
   write (23,*) x,y
   read (24,*) x,y
   write (25,*) x,y
10 continue

close (20)
close (21)
close (22)
close (23)
close (24)
close (25)

return
end

H.12 Calculates the Gradient for Stable Stacking Fault (gradsolve.f)

subroutine gradsolve(grad, const1, const2, m,crystal,sindex,atoms,energy0)
C global variables
   integer satoms, indexatoms(15), layers
   common/specialatomcount/satoms, indexatoms, layers
C program variable
   integer  m, atoms
   double precision const1(m), const2(m), grad(m), crystal(atoms,3), sindex(atoms)
C local variable
    integer i, j, k
    double precision dummy, sum, dummyconst(m)
    double precision newcrystal(atoms,3), surface(atoms,3)
    double precision energy, energy0

c y is the calculated values using const1
c exact is the actual values of the calculated values in y.
c the f calculated in this subroutine is a vector of sum of
errors squared.
C in Case 1 the f vector is calculated and the gradient is
calculated using the sum of error's squared in the f vector.
C also in Case1 the Sum of residuals is checked to see if the
termination condition has been met.
C if the termination condition has been met the subroutine is
exited.
C in Case 2 the sum of errors squared is calculated using the
parameters in const1.

c write (*,*) 'const1 is', const1
 c write (*,*) 'const2 is', const2
 c write (*,*) 'sindex is', sindex

C This loop makes a copy of crystal
    do 1 i=1,atoms
        do 2 j=1,3
            newcrystal(i,j)=crystal(i,j)
2      continue
1      continue

C Create newcrystal using const1 parameters
C Creates surface of newcrystal and than calculates the energy of
that surface
    call crystalcreation(const1,m,newcrystal,atoms)
    do 155 j=1,satoms
        do 160 k=1,3
            surface(j,k)=newcrystal(sindex(j),k)
160    continue
155    continue
    call periodicbcenergy(newcrystal,atoms,surface,satoms,energy0)
 c write (*,*) 'the original crystal energy is', energy0

C this loop makes a copy of the parameters const1
    do 15 j=1,m
        dummyconst(j)=const1(j)
15    continue
C this loop replaces Const1 with const2 in the j position and then calculates energy.
C using crystalcreation and periodicbcenergy subroutine.
C afterwards replaces Const1 value back into the j position and then goes back to beginning.
C of loop. Loop includes check to see if const2(j) is 0. In order for const2(j) to be 0,
C Const1(j) would have to be 0 due to how those are calculated.
So grad(j)=0 go to end of loop.

    do 20 j=1,m
      if (const2(j).eq. 0.d0) then
        grad(j)=0.d0
        goto 20
      else
        Const1(j)=const2(j)
      endif
    do 3 i=1,atoms
      do 4 k=1,3
        newcrystal(i,k)=crystal(i,k)
      4     continue
    3     continue
    call crystalcreation(const1,m,newcrystal,atoms)
    C create surface of newcrystal
    do 14 i=1,satoms
      do 12 k=1,3
        surface(i,k)=newcrystal(sindex(i),k)
      12     continue
    14     continue
    C calculate energy of surface
    call periodicbcenergy(newcrystal,atoms,surface,satoms,energy)
    Const1(j)=dummyconst(j)
    dummy=const2(j)-Const1(j)
    grad(j)=(energy-energy0)/(dummy)
    c        write (70,*) 'the new crystal energy is', energy, ' at step', j
    c        write (70,*) 'the new crystal energy is', energy, ' at step', j
    c        write (70,*) 'gradient is', grad(j)
    c        write (70,*) 'the gradient at this step is',
    grad(j)
    c        write (70,*)
    20     continue
    c        write (70,*) 'the gradient is', grad
    c        write (70,*) 'the gradient is', grad
35 return
H.13 Calculating the Hexagonal Close Pack Energy

(hexagonalclosepackenergy.f)

subroutine hexagonalclosepackenergy(hcpenergy)
implicit none
C global variables
  double precision a0,rcut
  common/pot_fitting/a0,rcut
  integer elenum, fitmethod, potmethod
  character*2 element(6)
  common/control/elenum, element, fitmethod, potmethod
C Program Variables
  double precision hcpenergy
C Local variables
  integer atoms
  double precision crystal(729,3),crystrow(3), r
  integer i,j,k,l
  integer lesspoints
  parameter (lesspoints=3000)
  double precision smoothrho(lesspoints),smoothd2rho(lesspoints)
  double precision smoothr(lesspoints),
  smoothepair(lesspoints), smoothd2epair(lesspoints)
  double precision smoothden(lesspoints),
  smoothembed(lesspoints), smoothd2embed(lesspoints)
  double precision epair, rho, rhobar, epairsum, F

  a0=3.61500000000000d0
  rcut=5.50679d0
  atoms=729
  write (*,201) a0
  call hexagonalclosepackstruct(a0)
C read in atom structure and store in crystal
  open (1,file='hcp.xyz')
  do 10 i=1,atoms
    read (1,*)(crystal(i,1), crystal(i,2), crystal(i,3)
  10 continue
  close (1)

C open potential tables
  open (9,file=element(1)//'.embed')
  open (10,file=element(1)//'.pair')
  open (11,file=element(1)//'.den')

  c read in potential table
do 15 i=1,lesspoints
  read (9,*) smoothden(i),smoothemb(i)
  read (10,*) smoothr(i), smoothpair(i)
  read (11,*) smoothr(i), smoothrho(i)
15  continue

C create splines
  call zspline(smoothden,smoothemb,lesspoints,
               1d32,1d32,smoothd2embed)
  call zspline(smoothr,smoothpair,lesspoints,
               1d32,1d32,smoothd2epair)
  call zspline(smoothr,smoothrho,lesspoints,
               1d32,1d32,smoothd2rho)

  epairsum=0d0
  rhobar=0d0
C calculate pair potential and electron density on 1 atom in the
hcp structure
  C the 1 atom is at 0,0,0 in normal x,y,z axis system
  do 40 j=1,atoms
    do 60 l=1,3
      crystrow(l)=crystal(j,l)
60     continue
    r=0d0
    do 80 k=1,3
      r=(crystrow(k))**2+r
80     continue
    r=sqrt(r)
    if (r.lt.rcut .and. r.gt.0d0) then
      C rhobar and epair are calculate in this section
      call zsplint(smoothr,smoothpair,smoothd2epair,lesspoints,r,epair)
      epairsum=epair+epairsum
      call zsplint(smoothr,smoothrho,smoothd2rho,lesspoints,r,rho)
      rhobar=rho+rhobar
    endif
    c    write (*,*) 'rhobar is', rhobar
    c    write (*,*) 'epairsum is', epairsum
40  continue

c Calculate embedding Energy
  call zsplint(smoothden,smoothemb,smoothd2emb,lesspoints,rhobar,F)
  hcpenergy=.5*epairsum+F
  c write (*,*) 'hcp energy is', hcpenergy
  close (9)
  close (10)
  close (11)
return
end

H.14 Building the Hexagonal Close Pack Structure

(hexagonalclosepackstruct.f)

subroutine hexagonalclosepackstruct(latticeconstant)
implicit none

c Progam variables
double precision latticeconstant

c Local Variables
double precision perfect(2197,3),r,fpart
integer i,j,k,l,n

C r is the radius of the atom
r=latticeconstant*sqrt(2d0)/4d0

C this code makes a hexagonal close packed structure.
l=1
do 10 k=0,8
   do 20 j=0,8
      do 30 i=0,8
         perfect(l,3)=r+k*sqrt(8d0)*r/sqrt(3d0)
         if (fpart(dble(k/2d0)) .ne. 0d0) then
            perfect(l,2)=r+j*sqrt(3d0)*r
            if (fpart(dble(j/2d0)) .ne. 0d0) then
               perfect(l,1)=2*i*r
               else
               perfect(l,1)=r+2*i*r
               endif
            endif
         else
         endif
      endif
   endif
10 l=1+l
enddo
continue

C this do loop moves the atoms over to the left and back and down by 3 lattice units and writes the information into a file

c fcc.xyz

open (1, file='hcp.xyz')

do 40 l=1,n
    perfect(l,1)=perfect(l,1)-8d0*r
    perfect(l,2)=perfect(l,2)-r-4d0*sqrt(3d0)*r
    perfect(l,3)=perfect(l,3)-r-4d0*sqrt(8d0)*r/sqrt(3d0)
    c
    if (abs(perfect(l,1)).lt.1e-14) perfect(l,1)=0
    if (abs(perfect(l,2)).lt.1e-14) perfect(l,2)=0
    if (abs(perfect(l,3)).lt.1e-14) perfect(l,3)=0

40 continue

write (1,*) perfect(l,1), perfect(l,2), perfect(l,3)

close (1)

return

end

H.15 Convert the Potential Style Used in the Program to that Used by LAMMPS (lammpspot.f)

subroutine lammpspot
implicit none

C global variables
  double precision a0, rcut
  COMMON/pot_fitting/a0,rcut
  double precision mass
  common/mdprop/mass
  integer elenum, fitmethod, potmethod
  character*2 element(6)
  common/control/elenum, element, fitmethod, potmethod

C local variables
  double precision dummy1, dummy2
  double precision epair, rho, F, deltarho, deltartar
  integer points, lesspoints
  parameter (points=10001, lesspoints=3000)
  double precision Flist(lesspoints), F2list(lesspoints), rhobarlist(lesspoints)
  double precision smootherho(lesspoints), smoothd2rho(lesspoints)
  double precision smoothr(lesspoints), smoothd2epair(lesspoints)
double precision  r, rmin, rhobarmax, x, psi, s
integer  i

C beginning of pure element code
if (elenum .eq. 1) then

C open the setfl file
open (11,file=element(1)//'.eam.alloy')

C Initial Comment lines
write (11,*) '#-> LAMMPS Potential File in DYNAMO 86 setfl Format <-'
write (11,*) 'Fast '//element(1)//' Eam Potential by Zachary Kraus'
write (11,*) 'Used Mishin Spline method to create potential'

c calculating Values in initial setup lines
deltarho=2d0/points
open (29, file=element(1)//'.den')
do 40 i=1,lesspoints
   read (29,*) smoothr(i), smoothrho(i)
40 continue

closing (29)
deltar=rcut/points

C Initial Setup Lines
write (11,*) 1, '//element(1)
write (11,*) points, deltarho, points, deltar, rcut
write (11,*) 1, mass, a0, ' FCC'

C Initiate Embedding Energy, Electron Density and Pair Potential Spline
open (29, file=element(1)//'.den')
onpe (30, file=element(1)//'.pair')
onpe (31, file=element(1)//'.embed')
do 15 i=1,lesspoints
   read (29,*) smoothr(i), smoothrho(i)
   read (30,*) smoothr(i), smoothpair(i)
   read (31,*) rhobarlist(i), Flist(i)
15 continue

call zspline(smoothr, smoothrho, lesspoints, 1d32, 1d32,
smooth2rho)
call zspline(smoothr, smoothpair, lesspoints, 1d32, 1d32,
smooth2pair)
call zspline(rhobarlist, Flist, lesspoints, 1d32, 1d32,
F2list)

C Insert Embedding Energy to file
write (11,14) 0d0
14 Format (F13.9)
do 10 i=1,points-1
   call zsplint(rhobarlist, Flist, F2list, lesspoints, i*deltarho, F)
   write (11,*) F
10 continue

C Insert Electron Density to file
   do 12 i=0,points-1
      call zsplint(smoothr, smoothrho, smoothd2rho, lesspoints, i*deltar, rho)
      write (11,*) rho
12 continue

C Insert (Pair Potential)*R to file
   write (11,14) 0d0
   do 13 i=1,points-1
      call zsplint(smoothr, smoothpair, smoothd2epair, lesspoints, i*deltar, epair)
      write (11,*) epair*i*deltar
13 continue
   close (11)
   close (29)
   close (30)
   close (31)
return

C end of pure element code
endif
end

H.16 Choose the Potential Which has the Better Test Sr (letterchoozer.f)

subroutine letterchoozer(letter,sr)
implicit none
c global variables
   integer a, b, c, m, n
   common/parameters/a,b,c,m,n
   integer aold,bold,cold
   common/oldies/aold,bold,cold
   double precision sratest,srbtest,srctest
   common/srlettertest/sratest,srbtest,srctest
   c program variables
   character letter*1
   double precision sr

C Depending on which letter is choozen
c different values for the sr and a,b or c will be found.
   if (letter .eq. 'a') then
      a=aold+1

sr=srtest
elseif (letter .eq. 'b') then
    b=bold+1
    sr=srbtest
elseif (letter .eq. 'c') then
    c=cold+1
    sr=srchtest
endif
return
end

H.17 Setups Molecular Dynamic Property Calculations by Equilibrating a Simulation Box at 293 K for Calculating Fitting Sr (mdsetupfit.f)

subroutine mdsetupfit
implicit none

c Global Variables
    integer a,b,c,m,n
    common/parameters/a,b,c,m,n
    integer elenum, fitmethod, potmethod
    character*2 element(6)
    common/control/elenum, element, fitmethod, potmethod

c local variables
    integer i, propnum, eqcount, eqmax
C the value n is the number of properties calculated in srsolvefit
C which has been defined in pot.f and located in the first line in the fit.txt file.
C The rest of the lines in the fit.txt file correspond to a property #.
C These property #'s are used to calculate the needed properties.
C In this subroutine,
C those property #'s are used to call the correct md equilibration routine if needed.
C The subroutine exits when all values in the file have been read or
C the maximum number of md equilibration routines as defined by eqmax has been called.
C The subroutine also calls another subroutine to change the software's potentials to
C Lammmps friendly potentials that can be used in MD calculations.

C Open file to read property # used for executing correct property calculations
    open (29,file='./controls/fit.txt')
C dummy read used to skip to the actual property #'s in the file
read (29,*) propnum

C begining of pure metal code
if (elenum .eq. 1) then

i=1
eqcount=0
eqmax=1

C while loop where properties and the Sum of Residuals Squared of the fit (srfit) are calculated
12 if (i .le. n) then
C reads in the prop # from fit.txt and executes the corresponding property calculation
read (29,*) propnum
C Execute 293eq.py when property # is 10
if (propnum .eq. 10) then
  if (eqcount .eq. 0) call lammpspot
  call system('python eq293.py')
eqcount=eqcount+1
  if (eqcount .eq. eqmax) goto 13
  i=i+1
  goto 12

C Execute 293eq.py when property # is 11
elseif (propnum .eq. 11) then
  if (eqcount .eq. 0) call lammpspot
  call system('python eq293.py')
eqcount=eqcount+1
  if (eqcount .eq. eqmax) goto 13
  i=i+1
  goto 12

C Return to begining of loop for all other property #'s
else
  i=i+1
  goto 12
endif
C end of while loop
def
13 close (29)
return
C end of pure metal code
def
end

H.18 Setsups Molecular Dynamic Property Calculations by Equilibrating a Simulation Box at 293 K for Calculating Testing Sr (mdsetuptest.f)
subroutine mdsetuptest(testnum)
implicit none
C program variables
integer testnum
C global variables
integer elenum, fitmethod, potmethod
character*2 element(6)
common/control/elenum, element, fitmethod, potmethod
c local variables
integer i, propnum, eqcount, eqmax
C the value n is the number of properties calculated in
srsolvefit
C which has been defined in pot.f and located in the first line
in the fit.txt file.
C The rest of the lines in the fit.txt file correspond to a
property #.
c These property #'s are used to calculate the needed properties.
C In this subroutine,
C those property #'s are used to call the correct md
equilibration routine if needed.
C The subroutine exits when all values in the file have been read
or
C the maximum number of md equilibration routines as defined by
eqmax has been called.
C The subroutine also calls another subroutine to change the
software's potentials to
C Lammps friendly potentials that can be used in MD calculations.
C Open file to read property # used for executing correct
property calculations
open (29, file='./controls/test.txt')
C dummy read used to skip to the actual property #'s in the file
read (29,*) propnum
C beginning of pure metal code
if (elenum.eq.1) then
  i=1
  eqcount=0
  eqmax=1
C while loop where properties and the Sum of Residuals Squared of
the fit (srfit) are calculated
12 if (i .le. testnum) then
C reads in the prop # from fit.txt and executes the corresponding
property calculation
  read (29,*) propnum
C Execute 293eq.py when property # is 10
if (propnum.eq.10) then
if (eqcount .eq. 0) call lammpspot
call system('python eq293.py')
eqcount=eqcount+1
if (eqcount .eq. eqmax) goto 13
goto 12

C Execute 293eq.py when property # is 11
elseif (propnum .eq. 11) then
  if (eqcount .eq. 0) call lammpspot
call system('python eq293.py')
eqcount=eqcount+1
if (eqcount .eq. eqmax) goto 13
  goto 12

C Return to begining of loop for all other property #'s
else
  i=i+1
goto 12
endif

C end of while loop
def __init__(self,x,y,func='x',p=[]):
  """func is a string with x as the independent variable and p[i] as the parameters
any other version of func will blow up."""
  self.y=y #known values
  self.x=x #independent variables (in the case where function is not defined these are the calculated values
# and p would be []

H.19 Function Template for Finding Temperature Gradient by Optimization
which is Used in Calculating Thermal Conductivity (objectivefunctions.py)
self.func=func  # function used to calculate the values
self.p=p  # parameters for the function.
def calculate(self):  # works
    p=self.p
    sum=0
    for i in range(len(self.y)):
        try: x=self.x[i]
        except IndexError: break
        try: calcval=eval(self.func)
        except NameError:
            print 'func must be in terms of x and p[i]'
            break
        sum=(self.y[i]-calcval)**2+sum
    return sum

class Sr2yw:
    def __init__(self,x,y,func='x',p=[]):
        """func is a string with x as the independent variable
        and p[i] as the parameters
        any other version of func will blow up."""
        self.y=y  # known values
        self.x=x  # independent variables (in the case where
        function is not defined these are the calculated values
        # and p would be []
        self.func=func  # function used to calculate the values
        self.p=p  # parameters for the function.
    def calculate(self):  # works
        p=self.p
        sum=0
        for i in range(len(self.y)):
            try: x=self.x[i]
            except IndexError: break
            try: calcval=eval(self.func)
            except NameError:
                print 'func must be in terms of x and p[i]'
                break
            sum=((self.y[i]-calcval)/self.y[i])**2+sum
        return sum

# ignore below rant it is entirely wrong. The real truth is
# assigning a variable to a list type variable
# Creates a linkage between the two variables such that a change
to an indexed value actually changes both variables
# but changing either variable entirely breaks the linkage. Also
# assigning to a new variable not part of the
# linkage will also break the linkage
# To avoid this problem, initially a list type variable must be assigned to an indexed value in the other list variable
# This method will never create a linkage between the two variables

# This linkage issue with lists can be taken advantage of. Unfortunately knowing when to take advantage of this situation in a controlled manner would be difficult and be even more difficult for people reading your code to understand.

# the end point here is do not assign list variables to another list variable directly; Use index assignments!

# This is a RANT about BUGS with Class Attributes in Python
# *********************************************************
# *********************************************************
# End result class attributes that are heavily changed externally should probably not be class attributes
# except for under certain circumstances
# These circumstances are when the class attributes can be used directly without being assigned to any variables
# in the module
# the second the attributes are assigned to variables in a module or even externally and than passed into the module
# Strange data linkages occur
# Why these data linkages do not occur in the module that is initiating the class is beyond my understanding.
# My best guess is the issue has to do with how python scopes class attributes assigned to a varible in sub modules of the class or the initiating module.
# End result learn from this and dont spend two days debugging a simple module that I allready know how to write.

**H.20 Optimization Used to Find Temperature Gradient for Calculating the Thermal Conductivity (optimizations.py)**

```python
import numpy

def steepestdescent(objfunc):
    # will probally want to add a bit more control into like the delta and alpha parameters
    """Takes in objfunc class, uses class methods to calculate derivate and modify the data in the class."
```
Exits when 1000 steps have been done or the derivative is near 0 or the sr is near 0

```python
p0 = objfunc.p
# print 'p0 is', p0
p1 = range(len(p0))  # used to initialise p1
dp = range(len(p0))  # used to initialise dp
pder = range(len(p0))  # used to initialise pder
pnew = range(len(p0))  # used to initialise pnew
# print 'p1 is', p1
# print 'dp is', dp
for i in range(1000):
    # print 'the step is', i+1
    sr0 = objfunc.calculate()
    # print 'sr0 is', sr0
    if sr0 < .00001: return 1

    # for setting up the other set of parameters
    for j in range(len(p0)):
        p1[j] = p0[j] * 1.01
        dp[j] = p1[j] - p0[j]  # difference in parameters at index j

    # print 'p1 is', p1
    # print 'dp is', dp

    # for calculating the partial derivatives
    for j in range(len(p0)):
        pnew[j] = p0[j]

    # print 'pnew is', pnew

    for j in range(len(p0)):
        pnew[j] = p1[j]  # replace current position with p1 parameter

    print 'pnew after replacing index', j, 'with', p1[j], 'is', pnew

    # print 'the derivative is', pder
    # print 'the magnitude of the derivates is',
    numpy.linalg.norm(pder)
    if numpy.linalg.norm(pder) < .0001: return 1

    # for producing new values of p0
    for j in range(len(p0)):
```
\[ p_0[j] = p_0[j] - p_{der}[j] \times 0.7 \] #producing a new set of parameters with known derivatives

objcfunc.p = p0

# print 'p0 for the next step is', p0

return 0

H.21 Optimizes the Potential by Minimizing Least Square Error Between Experimental and Calculated Properties (optimize.f)

subroutine optimize(sr)
implicit none
C global Variables

integer a, b, c, m, n
common/parameters/a,b,c,m,n

double precision alpha, beta, gamma, delta
common/controls/alpha,beta, gamma, delta

integer switch
common/roseswitch/switch

integer endflag
common/finalfit/endflag

integer elenum, fitmethod, potmethod
character*2 element(6)
common/control/elenum, element, fitmethod, potmethod

c program variables

double precision sr

C local variables

integer best, secondworse, worse, pworse

double precision const1(m), constmat(m+1,m), srvect(m+1)

double precision centroid(m)

double precision reflection(m)

double precision srreflection

CHARACTER FILENAME*36

integer i,j,k

C Code for Simplex optimization method [Should work for both alloys and pure metals]

if (fitmethod .eq. 1) then
write (*,*) 'Neldermead algorithm choosen for optimization'
write (28,*) 'Neldermead algorithm choosen for optimization'

alpha controls reflection

alpha = 1d0

beta controls contraction

beta = 0.5d0

gamma controls expansion

gamma = 2d0

delta controls shrinkage
delta=.5d0
C switch controls whether srsolve will run rose et al. (set to 1 for rose et al to run)
switch=1
C Controls when srsolvefit should write down the final fitted properties. [1 = write]
endflag=0
C number of parameters to minimize
c m=16
c number of points for pair potential, electron density and then embedding energy
c a=9
c b=7
c c=5
c number of properties calculated
c n=4

C code to open the right initialconstants file in the database folder
FILENAME(1:29) = './database//'
element(1)//'initialconstants'
FILENAME(30:30) = char(a+48)
FILENAME(31:31) = char(b+48)
FILENAME(32:32) = char(c+48)
FILENAME(33:36) = '.txt'

open (40,file=FILENAME,status='old')
read (40,*) const1
close (40)
c open (50,file='./nelder.txt')
c write (50,*) 'const1 is', const1

C Initiates the simplex using right angle rules. the first set of parameters is the initial paramaters.
c All other paramaters are at right angles to the initial condition.
C Also a mutliplicative rather than additive condition is used to keep other parameters close to initial condition.
do 10 i=1,m+1
do 20 j=1,m
  if (i-1 .eq. j)then
    constmat(i,j)=const1(j)*(1.1d0)
  else
    constmat(i,j)=const1(j)
end
c write (50,*) 'constant at row',i , 'column', j, 'is', constmat(i,j)
endif
20 continue 10 continue c write (50,*) 'constmat is', constmat

C Calculate The Srs for each set of parameters in Constmat. Place the Srs in Srvect.
write (*,*) 'initial calculations for neldermead'
do 15 i=1,m+1
   do 25 j=1,m
      const1(j)=constmat(i,j)
   continue
call splinemethod(const1,sr)
srvect(i)=sr
15 continue write (*,*) 'initial calculations finished'

C Begin Nelder Mead Algorithm. max iterations set as 1000.
do 30 i=1,1000
c write (50,*)
c write (50,*) 'beginning of step', i
write (*,*) 'begining of step', i
C Finding the index of the Best, Second Worse, and Worse SRs
worse=1
secondworse=1
best=1
c write (50,*) 'srvect is', srvect
do 35 j=2,m+1
   if (srvect(j) .le. srvect(best)) then
      pworse=best
      best=j
   else
      pworse=j
   endif
   if (srvect(pworse) .ge. srvect(worse)) then
      secondworse=worse
      worse=pworse
   elseif (srvect(pworse) .ge. srvect(secondworse))
      then
         secondworse=pworse
   endif
35 continue
c write (50,*) 'best is', best
c write (50,*) 'secondworse is', secondworse
c write (50,*) 'worse is', worse
if (i .eq. 1) write (50,*) 'beginning sr is', srvect(worse)
C Calculate the Centroid of the best side
   do 40 k=1,m
      centroid(k)=0d0
   do 45 j=1,m+1
      if (j .ne. worse) centroid(k)=centroid(k)+constmat(j,k)
   45   continue
   centroid(k)=centroid(k)/m
   40   continue
C Calculate Reflection Parameters. Calculate Sr Reflection
c If Sr reflection less than Sr second worse but greater or equal to Sr best, Keep Reflection Parameters.
c place Reflection parameters and Sr into Worse Slot.
   do 50 j=1,m
      reflection(j)=centroid(j)+alpha*(centroid(j)-constmat(worse,j))
   50   continue
C Calculate Expansion parameters and Sr Expansion
C If Shrinking is needed, Also done in this subroutine.
c write (50,*) 'reflection is', reflection
write (*,*) 'reflection calculated'
c call splinemethod(reflection,srreflection)
c write (50,*) 'srreflection is', srreflection
write (50,*) 'sr second worse is', srvect(secondworse)
c write (50,*) 'sr best is', srvect(best)
c if (srreflection.lt.srvect(secondworse).and.srreflection.ge.srvect(best)) then
   do 55 j=1,m
      constmat(worse,j)=reflection(j)
   55   continue
   srvect(worse)=srreflection
   write (50,*) 'srreflection bewteen second worse sr and best sr'
c write (50,*) 'reflection choosen'
c write (50,*) '1'
c elseif (srreflection.lt.srvect(best)) then
C Calculate Contraction Parameters and Sr Contraction.
c write (50,*)'srreflection g.t. sr secondworse calculate contraction'
call expand(srvect,constmat,centroid,reflection,srreflection,worse)
else
C Calculate Contraction Parameters and Sr Contraction.
c write (50,*) 'srreflection g.t. sr secondworse calculate contraction'
call contract(srvect, constmat, centroid, reflection, srreflection, worse, best)

C Termination Tests: function value convergence
C For Worse: Sr should be less than .01
if (srvect(worse).lt. .00001) then
  write (50,*) 'worse srvect is below .00001.'
  do 2 j=1,m
    const1(j)=constmat(worse,j)
  continue
C this step ensures the spline table is for the converged value.
  endflag=1
  call splinemethod(const1,sr)
  goto 1
endif
C For Entire Simplex: Sr should be within .5% of Sr(best)
k=0
  do 3 j=1,m+1
    if (j .ne. best) then
      sr=srvect(best)
      if (sr*1.005.ge. srvect(j) .and. sr*.995.le. srvect(j)) then
        k=k+1
        write (50,*) 'Sr,,srvect(j), ,at,, j, ' is within 0.1% of best Sr,,srvect(best)
      write (50,*) 'k is', k
    endif
  continue
  if (k .eq. m) then
    write (50,*) 'all parameters within 0.1% of best Sr'
    write (50,*) 'k is', k
    do 4 j=1,m
      const1(j)=constmat(best,j)
  continue
C This step ensures the spline table is for the converged value.
  final spline table'
  endflag=1
  call splinemethod(const1,sr)
  goto 1
endif
C End of one loop of nelder mead Algorithm
write (50,*) 'end of step', i
write (*,*) 'end of step', i
write (*,*)
write (50,*) 'minimized parameters after longrange search are', const1
write (50,*) 'minimized sr after long range search is', sr
C save the constants from the end of the long neldermead algorithm
1 open (40, file='nextconstants.txt')
write (40,*) const1
close (40)
write (28,*) 'the neldermead algorithm took', i, ' steps to optimize.'
write (*,*) 'the neldermead algorithm took', i, ' steps to optimize.'
c write (50,*)
c write (50,*) 'now beginning shortrange search'
c write (50,*) 'finished sr from long search is', sr
c close (50)
return
C End of Simplex optimization method
endif
end

C If Sr Expansion less than Sr Reflection, Keep Expansion Parameters.
c Otherwise keep Reflection Parameters.
C this is the greedy minimization approach.
C original nelder-mead paper used greedy expansion to keep simplex as large as possible to avoid premature terminations.
C if Function is terminating to early replace if else block starting on line 107 with do loop 65 and the sr replacement line after
c to change algorithm to greedy expansion.
subroutine expand(srvec, constmat, centroid, reflection, srreflection, worse)
implicit none
C global Variables
integer a,b,c,m,n
common/Parameters/a,b,c,m,n
double precision alpha, beta, gamma, delta
common/controls/alpha,beta, gamma, delta
C routine Variables
integer worse
double precision srvec(m+1), constmat(m+1,m), centroid(m)
double precision reflection(m), srreflection
C local Variables
double precision expansion(m), srexpansion
integer i,j,k
do 60 j=1,m
   expansion(j)=centroid(j)+gamma*(reflection(j)-
   centroid(j))
60 continue

C Contraction can either be on the inside (Sr reflection greater
or equal to Sr worse)
C or on the outside (Sr Reflection less than Sr worse).
C In inside case Contraction parameters accepted if Sr
contraction less than Sr worse.
C In outside case Contraction Parameters accepted if Sr
contraction less than Sr reflection.

C subroutine contract(srvect,constmat,centroid,reflection,srreflection,worse,best)
   implicit none
C global Variables
   integer a,b,c,m,n
   common/Parameters/a,b,c,m,n
   double precision alpha, beta, gamma, delta
   common/controls/alpha,beta,gamma,delta
C routine Variables
   integer worse,best
   double precision srvect(m+1), constmat(m+1,m), centroid(m)
   double precision Reflection(m), srreflection
C local Variables
  double precision contraction(m), srcontraction
  integer i, j, k

  write (50,*) 'srreflection is', srreflection
  write (50,*) 'worse sr is', srvect(worse)
    if (srreflection .ge. srvect(worse)) then
      write (50,*) 'srreflection is g.t. worse sr'
      write (50,*) 'inside contraction choozen'
      do 75 j=1,m
         contraction(j)=centroid(j)
         +beta*(constmat(worse,j)-centroid(j))
      continue
    end if
      write (50,*) 'contraction is', contraction
      write (*,*), 'inner contraction calculated'
      call splinemethod(contraction,srcontraction)
    else
      write (50,*) 'srcontraction is', srcontraction
      write (50,*) 'worse sr is', srvect(worse)
      if (srcontraction .lt. srvect(worse)) then
        write (50,*) 'srcontraction l.t. worse sr'
        write (50,*) 'contraction choozen'
        write (50,*) '3'
      end if
        do 80 j=1,m
           constmat(worse,j)=contraction(j)
        continue
        write (*,*), 'outer contraction calculated'
        call splinemethod(contraction,srcontraction)
      end if
    end else

  write (50,*) 'srreflection is l.t. worse sr'
  write (50,*) 'outside contraction choozen'
  do 85 j=1,m
     contraction(j)=centroid(j)+beta*(reflection(j)-
    centroid(j))
  continue
  write (50,*) 'contraction is', contraction
  write (*,*), 'outer contraction calculated'
  call splinemethod(contraction,srcontraction)
do 90 j=1,m
   constmat(worse,j)=contraction(j)
90       continue
else
   srvect(worse)=srccontraction
C Calculate Shrink Parameters and new Srs
   write (50,*), 'sr reflection l.t. sr contraction calculate and choose shrink'
   call shrink(srvect,constmat,best)
endif
C End if corresponds to the end of the contraction calculation block above
endif
return
end

subroutine shrink(srvect,constmat,best)
implicit none
C global Variables
integer a,b,c,m,n
common/Parameters/a,b,c,m,n
double precision alpha, beta, gamma, delta
common/controls/alpha,beta,gamma,delta
C routine Variables
integer worse,best
double precision srvect(m+1), constmat(m+1,m)
C local Variables
integer i,j,k
double precision const1(m),sr

write (50,*), 'best is', best
write (*,*), 'shrink calculated'
c write (50,*), '4'
do 10 j=1,m+1
   if (j .ne. best) then
      do 5 k=1,m
         constmat(j,k)=constmat(best,k)+delta*(constmat(j,k)-constmat(best,k))
         const1(k)=constmat(j,k)
5       continue
c write (50,*), 'new constants on row', j, 'are', const1
   call splinemethod(const1,sr)
   srvect(j)=sr
   write (50,*), 'new sr on row', j, 'is', srvect(j)
   endif
10 continue
return
end

320
H.22 Calculating the Energy of Atoms in a Box with Periodic Boundaries

(periodicboundaryconditionenergy.f)

subroutine periodicbcenergy(crystal, atoms, atomint, intnum, energy)
implicit none
C global variables
double precision a0,rcut
common/pot_fitting/a0,rcut
double precision xmin, xmax, zmin, zmax, ymax, ymin
common/boundarycondition/xmin,xmax,zmin,zmax,ymax,ymin
integer max(3), min(3)
common/pbcon/max,min
integer elenum, fitmethod, potmethod
character*2 element(6)
common/control/elenum, element, fitmethod, potmethod
C Program Variables
integer atoms, intnum
double precision crystal(atoms,3), atomint(atoms,3), energy,
atomenergy
C local variables
double precision r, delta(3)
double precision introw(3), crystrow(3), image(3)
integer i,j,k,l,noimagecount
integer lesspoints
parameter (lesspoints=3000)
double precision smoothrho(lesspoints),smoothd2rho(lesspoints)
double precision smoothr(lesspoints),
smoothpair(lesspoints), smoothd2pair(lesspoints)
double precision smoothden(lesspoints),
smoothembed(lesspoints), smoothd2embed(lesspoints)
double precision epair, rho, rhobar, epairsum, F
C I have assumed that the minimum image criterion holds
C begining of pure metal code [added in case code wants to be
expanded for alloys later.]
if (elenum .eq. 1) then
C open potential tables
open (9, file=element(1)///'.embed')
open (10, file=element(1)///'.pair')
open (11, file=element(1)///'.den')
c open (12, file='energy.txt')
c open (13, file='pbc.txt')
c open (14, file='pbccrystal.xyz')
c open (15, file='pbcint.xyz')
c read in potential table
do 10 i=1,lesspoints
    read (9,*) smoothden(i), smoothembed(i)
    read (10,*) smoothr(i), smoothpair(i)
    read (11,*) smoothr(i), smoothrho(i)
10   continue

c   write (*,*) 'max is', max
   write (*,*) 'min is', min
C create splines
   call zspline(smoothden, smoothembed, lesspoints,
1d32,1d32, smooth2embed)
   call zspline(smoothr, smoothpair, lesspoints,
1d32,1d32, smooth2pair)
   call zspline(smoothr, smoothrho, lesspoints,
1d32,1d32, smooth2rho)

C Length of box in x,y,z direction; 1=x, 2=y, 3=z.
   if (max(1).eq.1.or.min(1).eq.1) delta(1)=xmax-xmin
   if (max(2).eq.1.or.min(2).eq.1) delta(2)=ymax-ymin
   if (max(3).eq.1.or.min(3).eq.1) delta(3)=zmax-zmin

c   write (*,*) 'delta x is', delta(1)
   write (*,*) 'delta y is', delta(2)
   write (*,*) 'delta z is', deltaz
C Move atoms in crystal and atom in that are past real box
boundary conditions back into the box
   do 20 i=1,atoms
      if (crystal(i,1).lt.xmin) then
         if (min(1).eq.1) crystal(i,1)=crystal(i,1)+
delta(1)+delta(1)*int(abs((crystal(i,1)-xmin)/delta(1)))
      elseif (crystal(i,1).ge.xmax) then
         if (max(1).eq.1) crystal(i,1)=crystal(i,1)-
delta(1)-delta(1)*int(abs((crystal(i,1)-xmax)/delta(1)))
      endif
      if (crystal(i,2).lt.ymin) then
         if (min(2).eq.1) crystal(i,2)=crystal(i,2)+
delta(2)+delta(2)*int(abs((crystal(i,2)-ymin)/delta(2)))
      elseif (crystal(i,2).ge.ymax) then
         if (max(2).eq.1) crystal(i,2)=crystal(i,2)-
delta(2)-delta(2)*int(abs((crystal(i,2)-ymax)/delta(2)))
      endif
      if (crystal(i,3).lt.zmin) then
         if (min(3).eq.1) crystal(i,3)=crystal(i,3)+
delta(3)+delta(3)*int(abs((crystal(i,3)-zmin)/delta(3)))
      elseif (crystal(i,3).ge.zmax) then
         if (max(3).eq.1) crystal(i,3)=crystal(i,3)-
delta(3)-delta(3)*int(abs((crystal(i,3)-zmax)/delta(3)))
      endif
   c   write (14,*) crystal(i,1), crystal(i,2), crystal(i,3)

322
20 continue
   do 25 i=1,intnum
      if (atomint(i,1) .lt. xmin) then
         if (min(1) .eq. 1) atomint(i,1)=atomint(i,1)+delta(1)+delta(1)*int(abs((atomint(i,1)-xmin)/delta(1)))
         elseif (atomint(i,1) .ge. xmax) then
            if (max(1) .eq. 1) atomint(i,1)=atomint(i,1)-delta(1)-delta(1)*int(abs((atomint(i,1)-xmax)/delta(1)))
         endif
      if (atomint(i,2) .lt. ymin) then
         if (min(2) .eq. 1) atomint(i,2)=atomint(i,2)+delta(2)+delta(2)*int(abs((atomint(i,2)-ymin)/delta(2)))
         elseif (atomint(i,2) .ge. ymax) then
            if (max(2) .eq. 1) atomint(i,2)=atomint(i,2)-delta(2)-delta(2)*int(abs((atomint(i,2)-ymax)/delta(2)))
         endif
      if (atomint(i,3) .lt. zmin) then
         if (min(3) .eq. 1) atomint(i,3)=atomint(i,3)+delta(3)+delta(3)*int(abs((atomint(i,3)-zmin)/delta(3)))
         elseif (atomint(i,3) .ge. zmax) then
            if (max(3) .eq. 1) atomint(i,3)=atomint(i,3)-delta(3)-delta(3)*int(abs((atomint(i,3)-zmax)/delta(3)))
         endif
      if (atomint(i,1) .eq. intnum) continue
      endif
   endif
25 continue
   c !
   c energy=0d0
   c loop through interested atoms
   do 30 i=1,intnum
      do 50 k=1,3
         introw(k)=atomint(i,k)
      50 continue
      epairsum=0d0
      rhobar=0d0
   C loop through crystal atoms
   do 40 j=1,atoms
      do 60 l=1,3
         crystrow(l)=crystal(j,l)
      60 continue
      write (13,*) 'interested atom', i, ' is', introw
      write (13,*) 'crystal atom', j, 'is', crystrow
   C calculate Energy inside real box
   r=0d0
   do 80 k=1,3
      r=(crystrow(k)-introw(k))**2+r
   80 continue
      r=sqrt(r)
   c write (13,*) 'distance in the box', r
   if (r .lt. rcut .and. r .gt. .001d0) then
      !
   endif
323
C rhobar and epair are calculate in this section
   call zsplint(smoothr,smoothrepair,smoothd2epair,lesspoints,r,epair)
   epairsum=epair+epairsum
   call zsplint(smoothr,smoothrho,smoothd2rho,lesspoints,r,rho)
   rhobar=rho+rhobar
   write (13,*) 'rho in the box is', rho
   write (13,*) 'epair in the box is', epair
C interested atom and crystal atom are on top of each other move
to end of crystal loop
  elseif (r .lt. .001d0) then
    goto 5
  endif
  write(13,*) 'atoms overlap proceed to next atom in the crystal'
C Calculate needed image of crystal atoms and calculate the
energy of the crystal atoms.
  else
    noimagecount=0
    do 70 k=1,3
      r=crystrow(k)-introw(k)
      if (r .lt. -rcut) then
        if (max(k) .eq. 1) then
          image(k)=crystrow(k)+delta(k)
        else
          image(k)=crystrow(k)
          noimagecount=1+noimagecount
        endif
      elseif (r .gt. rcut) then
        if (min(k) .eq. 1) then
          image(k)=crystrow(k)-delta(k)
        else
          image(k)=crystrow(k)
          noimagecount=1+noimagecount
        endif
      elseif neither condition is true, image(k) will just be crystrow(K)
      else
        image(k)=crystrow(k)
        noimagecount=1+noimagecount
      endif
    70  continue
  endif
  write (13,*) 'image atom is', image
  if (noimagecount .eq. 3) then
    write (13,*) 'no image occured'
    goto 5
  endif
C calculate energy of crystalimage
  r=0
do 90 k=1,3
    r=(image(k)-introw(k))*r+r
90    continue
    r=sqrt(r)
c    write (13,*) 'distance for the image is', r
    if (r .lt. rcut .and. r .gt. 0.001d0) then
        call zsplint(smoothr,smoothpair,smoothd2pair,lesspoints,r,epair)
        epairsum=epair+epairsum
        call zsplint(smoothr,sMOOTHrho,smoothd2rho,lesspoints,r,rho)
        rhobar=rho+rhobar
        c    write (13,*) 'rho in the image box is', rho
        c    write (13,*) 'epair in the image box is', epair
        endif
      5  endif
      c    write (13,*) 'rhobar after calculation of crystal atom',j,' is', rhobar
      c    write (13,*) 'epair sum after calculation of crystal atom',j,' is', epairsum
      c    write (13,*) 'end of crystal atom'
c    write (13,*)
C end of crystal loop
40    continue
      c    write (12,*) 'data for atom', i
      c    write (12,*) 'epairsum is', epairsum
Calculate embedding Energy
    call zsplint(smoothden,smoothembed,smoothd2embed,lesspoints,rhobar,F)
      c    write (12,*) 'F is', F
      atomenergy=.5*epairsum+F
      c    write (12,*) 'atomenergy is', atomenergy
C sum energy of atom
    energy=energy+atomenergy
      c    write (12,*) 'total energy after step', i,' is', energy
C end of interested atom loop
      c    write (12,*)
      c    write (13,*) 'final data for intersted atom', i
      c    write (13,*) 'epair sum is', epairsum
      c    write (13,*) 'rhobar is', rhobar
      c    write (13,*) 'F is', F
      c    write (13,*) 'atomenergy is', atomenergy
      c    write (13,*) 'total energy after step', i,' is', energy
      c    write (13,*)
30 continue
C write (*.*) 'totalenergy is', energy
C at this point totalenergy is known exit
    close (9)
    close (10)
    close (11)
C close (12)
C close (13)
C close (14)
C close (15)
return
C end of pure metal code
endif
end

H.23 Vacancy Stress Calculation (potential_vacancy_stress_new.f)

C -------------------------------------------------------
C CREATION DATE :
C LAST MODIFIED : Sun Oct 27 19:04:29 1996
C MODULE : potential_cu_mishin
C AUTHOR : LI JU
C
C CHANGE LOG
C V1.0 : FIRST VERSION
C -------------------------------------------------------

SUBROUTINE vEVALFORCE_COPPER(X00,Y00,Z00,SUMPOTE,FX,FY,FZ)
C
C EVALUATE FORCES ON ATOMS USING PAIRWISE ADDITIVE LENNARD-JONES (6-12) INTERMOLECULAR POTENTIAL
C
IMPLICIT DOUBLE PRECISION (A-H,O-Z)

PARAMETER ( NPD = 499, NDOF = NPD*3+9)

PARAMETER ( EPSI = 120., SIGMA = 3.405)
PARAMETER ( BOLZ = 1.38054E-23)

PARAMETER(NATOM_O = 8,NATOM_Ar = 18)
PARAMETER(UE_IN_GUASSIAN = 3.79470135954879)
PARAMETER(EV_IN_J=1.60217733E-19)

C COMMON/POS/
X0(NPD),Y0(NPD),Z0(NPD),ATOMTYPE(NPD),NBOUNDARY(NPD)
C COMMON/FOR/FX(NPD),FY(NPD),FZ(NPD)
DIMENSION X00(NPD),Y00(NPD),Z00(NPD),NATOMTYPE0(NPD)
DIMENSION FX(NPD),FY(NPD),FZ(NPD)
COMMON/NABLSTA/RLIST,NTIMES,KSQRT
COMMON/NABLSTB/LIST(NPD*400),NABORS(NPD)
COMMON/NABLSTC/LISTBIG(NPD*400),NABORSBIG(NPD)
COMMON/STRESS/S(NPD,6),VONMISES(NPD),N_STRESSFLAG

DIMENSION DEXX(NPD),DEYY(NPD),DEVZ(NPD),X(NPD)

DIMENSION SX00(NPD),SY00(NPD),SZ00(NPD)

dimension pair1(3000),pair2(3000),pair_der2(3000)
dimension den1(3000),den2(3000),den_der2(3000)
dimension embed1(3000),embed2(3000),embed_der2(3000)
common/sp_pair/pair1,pair2,pair_der2
common/sp_den/den1,den2,den_der2
common/sp_embed/embed1,embed2,embed_der2

COMMON/POS_RED1/H(3,3),Hinv(3,3)

double precision rcut, a0
COMMON/pot_fitting/a0,rcut

NP = NPD
C thickness = 15.3371 A
C ABCH_a0 = 3.61500008917548
   ABCH_a0 = a0
C XPBC = 5*ABCH_a0*SQRT(2.0)/2.

RCUT = 5.506786
RLIST = 1.5*RCUT
RLISTBIG = 2.*RCUT

FACTOR = EV_IN_J*1E20
FACTOR = 1.0

SUMPOT = 0.
K = 0
KBIG = 0

n = 3000

USTRESS_IN_GPA = ((1.60217733e-19)/1e-30*1e-9)

V0 = (ABCH_a0**3)/4

DO 10 I=1,NP
   DEXX(I) = 0.
   DEYY(I) = 0.
   DEVZ(I) = 0.
   X(I) = 0.

327
DO 11 J=1,6
S(I,J) = 0
         CONTINUE
11 CONTINUE
10 CONTINUE
DO 1 I = 1, NPD
   SX00(I) = Hinv(1,1)*X00(I) + Hinv(2,1)*Y00(I) +
             Hinv(3,1)*Z00(I)
   SY00(I) = Hinv(1,2)*X00(I) + Hinv(2,2)*Y00(I) +
             Hinv(3,2)*Z00(I)
   SZ00(I) = Hinv(1,3)*X00(I) + Hinv(2,3)*Y00(I) +
             Hinv(3,3)*Z00(I)
      CONTINUE
C
C   LISTCHK=1 CHECK BOTH LIST
C   LISTCHK=2 CHECK FROM BIGBALL
C   LISTCHK=3 NO CHECK
C
LISTCHK = 3
IF(MOD(NTIMES, 100).EQ.0) THEN
   LISTCHK = 1
ELSE
   IF(MOD(NTIMES, 10).EQ.0) THEN
      LISTCHK = 2
   ENDIF
ENDIF
NTIMES = NTIMES + 1
WRITE(*,*) 'LISTCHK', LISTCHK
DO 20 I = 1, NP-1
   IF(LISTCHK.EQ.1) THEN
      NABORS(I) = K+1
      NABORSBIG(I) = KBIG+1
      JBEGIN = I+1
      JEND = NP
   ELSE
      IF(LISTCHK.EQ.2) THEN
         NABORS(I) = K+1
         JBEGIN = NABORSBIG(I)
         JEND = NABORSBIG(I+1)-1
         IF(JBEGIN.GT.JEND) THEN
            WRITE(*,*) 'BIG'
         ENDIF
      ELSE
         JBEGIN = NABORS(I)
         JEND = NABORS(I+1)-1
      ENDIF
   ENDIF
20 CONTINUE
IF(JBEGIN.GT.JEND) THEN
  WRITE(*,*) 'BIG',JBEGIN,JEND
ENDIF
ENDIF

DO 30 JX=JBEGIN,JEND
  J = JX
  IF(LISTCHK.EQ.2) J=LISTBIG(JX)
  IF(LISTCHK.EQ.3) J=LIST(JX)

  SXIJ = SX00(I) - SX00(J)
  SYIJ = SY00(I) - SY00(J)
  SZIJ = SZ00(I) - SZ00(J)
  IF(SXIJ.GT.0.5D0) SXIJ = SXIJ-1.D0
  IF(SXIJ.LT.-0.5D0) SXIJ = SXIJ+1.D0
  IF(SYIJ.GT.0.5D0) SYIJ = SYIJ-1.D0
  IF(SYIJ.LT.-0.5D0) SYIJ = SYIJ+1.D0
  IF(SZIJ.GT.0.5D0) SZIJ = SZIJ-1.D0
  IF(SZIJ.LT.-0.5D0) SZIJ = SZIJ+1.D0

  RX = H(1,1)*SXIJ + H(2,1)*SYIJ + H(3,1)*SZIJ
  RY = H(1,2)*SXIJ + H(2,2)*SYIJ + H(3,2)*SZIJ
  RZ = H(1,3)*SXIJ + H(2,3)*SYIJ + H(3,3)*SZIJ

  R = DSQRT( RX**2 + RY**2 + RZ**2 )

  SXR = ABS(SX00(I)-SX00(J))
  IF(SXR.GT.0.5) THEN
    IF(ABS(SX00(I)-(SX00(J)+1)).LT. ABS(SX00(I)-(SX00(J)-1)))
      SX00J = SX00(J)+1.
    ELSE
      SX00J = SX00(J)-1.
    ENDIF
  ELSE
    SX00J = SX00(J)
  ENDIF

  SYR = ABS(SY00(I)-SY00(J))
  IF(SYR.GT.0.5) THEN
    IF(ABS(SY00(I)-(SY00(J)+1)).LT. ABS(SY00(I)-(SY00(J)-1)))
      SY00J = SY00(J)+1.
    ELSE
      SY00J = SY00(J)
  ENDIF

329
SY00J = SY00(J-1).
ENDIF
ELSE
SY00J = SY00(J)
ENDIF
SYR = ABS(SY00(I)-SY00J)
SZR = ABS(SZ00(I)-SZ00(J))
IF(SZR.GT.0.5) THEN
IF(ABS(SZ00(I)-(SZ00(J)+1)).LT.
&       ABS(SZ00(I)-(SZ00(J)-1)))
THEN
SZ00J = SZ00(J)+1.
ELSE
SZ00J = SZ00(J)-1.
ENDIF
ELSE
SZ00J = SZ00(J)
ENDIF
SZR = ABS(SZ00(I)-SZ00J)
RX = H(1,1)*SXR + H(2,1)*SYR + H(3,1)*SZR
RY = H(1,2)*SXR + H(2,2)*SYR + H(3,2)*SZR
RZ = H(1,3)*SXR + H(2,3)*SYR + H(3,3)*SZR
IF(LISTCHK.EQ.1) THEN
IF(R.LT.RLIST) THEN
K = K + 1
LIST(K) = J
ENDIF
IF(R.LT.RLISTBIG) THEN
KBIG = KBIG + 1
LISTBIG(KBIG) = J
ENDIF
ENDIF
IF(LISTCHK.EQ.2) THEN
IF(R.LT.RLIST) THEN
K = K + 1
LIST(K) = J
ENDIF
ENDIF
if (r.lt.rcut) then
call vsplint(den1,den2,den_der2,n,r,rhij)
X(I) = X(I) + rhij
X(J) = X(J) + rhij
endif
30 CONTINUE

20 CONTINUE

IF(LISTCHK .EQ. 1) THEN
  NABORS(NP) = K+1
  NABTOT = K
  NABORSBIG(NP) = KBIG+1
  NABTOTBIG = KBIG
  c            write(*,*)  'Num of atoms in neighbor
  list',NABTOT,NABTOTBIG
ENDIF

IF(LISTCHK .EQ. 2) THEN
  NABORS(NP) = K+1
  NABTOT = K
  c            write(*,*)  'Updated num of atoms in neighbor
  list',NABTOT
ENDIF

DO 40 I =1, NP-1

  JBEGIN = NABORS(I)
  JEND = NABORS(I+1)-1

  call vsplint(embed1,embed2,embed_der2,n,x(i),u)
  call vsplintder1(embed1,embed2,embed_der2,n,x(i),dudxi)
  C          call vuu(X(i),u,dudxi,d2func)
  SUMPOTE = SUMPOTE + u

DO 50 JX=JBEGIN, JEND

  J = LIST(JX)

  SXIJ = SX00(I) - SX00(J)
  SYIJ = SY00(I) - SY00(J)
  SZIJ = SZ00(I) - SZ00(J)
  SX00J = SX00(J)
  SY00J = SY00(J)
  SZ00J = SZ00(J)

  IF(SXIJ.GT.0.5D0) THEN
    SXIJ = SXIJ-1.D0
    SX00J = SX00(J) + 1
  ENDIF
  IF(SXIJ.LT.-0.5D0) THEN
    SXIJ = SXIJ+1.D0
ENDIF
SX00J = SX00(J) - 1
ENDIF
IF(SYIJ.GT.0.5D0) THEN
    SYIJ = SYIJ - 1.D0
    SY00J = SY00(J) + 1
ENDIF
IF(SYIJ.LT.-0.5D0) THEN
    SYIJ = SYIJ + 1.D0
    SY00J = SY00(J) - 1
ENDIF
IF(SZIJ.GT.0.5D0) THEN
    SZIJ = SZIJ - 1.D0
    SZ00J = SZ00(J) + 1
ENDIF
IF(SZIJ.LT.-0.5D0) THEN
    SZIJ = SZIJ + 1.D0
    SZ00J = SZ00(J) - 1
ENDIF
RX = H(1,1)*SXIJ + H(2,1)*SYIJ + H(3,1)*SZIJ
RY = H(1,2)*SXIJ + H(2,2)*SYIJ + H(3,2)*SZIJ
RZ = H(1,3)*SXIJ + H(2,3)*SYIJ + H(3,3)*SZIJ
R = DSQRT( RX**2 + RY**2 + RZ**2 )
X00J = H(1,1)*SX00J + H(2,1)*SY00J + H(3,1)*SZ00J
Y00J = H(1,2)*SX00J + H(2,2)*SY00J + H(3,2)*SZ00J
Z00J = H(1,3)*SX00J + H(2,3)*SY00J + H(3,3)*SZ00J
C
SXR = ABS(SX00(I)-SX00(J))
C   IF(SXR.GT.0.5) THEN
C &   IF((ABS(SX00(I)-(SX00(J)+1)).LT.
C & &   ABS(SX00(I)-(SX00(J)-1)))
C THEN
C   SX00J = SX00(J)+1.
C ELSE
C   SX00J = SX00(J)-1.
C ENDIF
C ELSE
C   SX00J = SX00(J)
C ENDIF
C SXR = ABS(SX00(I)-SX00J)
C SYR = ABS(SY00(I)-SY00(J))
C   IF(SYR.GT.0.5) THEN
C &   IF((ABS(SY00(I)-(SY00(J)+1)).LT.
C & &   ABS(SY00(I)-(SY00(J)-1)))
C THEN
SY00J = SY00(J)+1.
ELSE
SY00J = SY00(J)-1.
ENDIF
ELSE
SY00J = SY00(J)
ENDIF
SYR = ABS(SY00(I)-SY00J)
SZR = ABS(SZ00(I)-SZ00(J))
IF(SZR.GT.0.5) THEN
IF(ABS(SZ00(I)-(SZ00(J)+1)).LT."
&
ABS(SZ00(I)-(SZ00(J)-1))) THEN
SZ00J = SZ00(J)+1.
ELSE
SZ00J = SZ00(J)-1.
ENDIF
ELSE
SZ00J = SZ00(J)
ENDIF
SZR = ABS(SZ00(I)-SZ00J)
RX = H(1,1)*SXR + H(2,1)*SYR + H(3,1)*SZR
RY = H(1,2)*SXR + H(2,2)*SYR + H(3,2)*SZR
RZ = H(1,3)*SXR + H(2,3)*SYR + H(3,3)*SZR

call
vsplintder1(embed1,embed2,embed_der2,n,x(j),dudxj)
call uu(X(j),func,dudxj,d2func)
if(r.lt.rcut) then
call vsplint(pair1,pair2,pair_der2,n,r,v)
call vsplintder1(pair1,pair2,pair_der2,n,r,dvdr)
call v2(R,v,dvdr,d2func)
call vsplintder1(den1,den2,den_der2,n,r,drhdr)
call rh(R,func,drhdr,d2func)
SUMPOTE = SUMPOTE + v
POTDER = dvdr+drhdr*(dudxi+dudxj)

DEDX(I)=DEDX(I) - POTDER*(X00(I)-X00J)/R
DEDY(I)=DEDY(I) - POTDER*(Y00(I)-Y00J)/R
DEDZ(I)=DEDZ(I) - POTDER*(Z00(I)-Z00J)/R

DEDX(J)=DEDX(J) + POTDER*(X00(I)-X00J)/R
DEDY(J)=DEDY(J) + POTDER*(Y00(I)-Y00J)/R
DEDZ(J)=DEDZ(J) + POTDER*(Z00(I)-Z00J)/R
**Atomic stress_1**

```fortran
C         IF(N_STRESSFLAG.EQ.1) THEN
    S(I, 1) = S(I, 1) + POTDER*(X00(I) - X00(J))*(X00(I) - X00(J))/R
    S(I, 2) = S(I, 2) + POTDER*(Y00(I) - Y00(J))*(Y00(I) - Y00(J))/R
    S(I, 3) = S(I, 3) + POTDER*(Z00(I) - Z00(J))*(Z00(I) - Z00(J))/R
    S(I, 4) = S(I, 4) + POTDER*(Y00(I) - Y00(J))*(Z00(I) - Z00(J))/R
    S(I, 5) = S(I, 5) + POTDER*(X00(I) - X00(J))*(Z00(I) - Z00(J))/R
    S(I, 6) = S(I, 6) + POTDER*(X00(I) - X00(J))*(Y00(I) - Y00(J))/R
    S(J, 1) = S(J, 1) + POTDER*(X00(I) - X00(J))*(X00(I) - X00(J))/R
    S(J, 2) = S(J, 2) + POTDER*(Y00(I) - Y00(J))*(Y00(I) - Y00(J))/R
    S(J, 3) = S(J, 3) + POTDER*(Z00(I) - Z00(J))*(Z00(I) - Z00(J))/R
    S(J, 4) = S(J, 4) + POTDER*(Y00(I) - Y00(J))*(Z00(I) - Z00(J))/R
    S(J, 5) = S(J, 5) + POTDER*(X00(I) - X00(J))*(Z00(I) - Z00(J))/R
    S(J, 6) = S(J, 6) + POTDER*(X00(I) - X00(J))*(Y00(I) - Y00(J))/R
C        ENDIF
```

**End atomic stress**

```fortran
endif
50       CONTINUE
FX(I) = DEDX(I)*FACTOR
FY(I) = DEDY(I)*FACTOR
FZ(I) = DEDZ(I)*FACTOR
C*** force in eV/A
```

**Atomic stress_2**

```fortran
C         IF(N_STRESSFLAG.EQ.1) THEN
    S(I, 1) = S(I, 1)/(2*V0)*USTRESS_IN_GPA
    S(I, 2) = S(I, 2)/(2*V0)*USTRESS_IN_GPA
    S(I, 3) = S(I, 3)/(2*V0)*USTRESS_IN_GPA
    S(I, 4) = S(I, 4)/(2*V0)*USTRESS_IN_GPA
    S(I, 5) = S(I, 5)/(2*V0)*USTRESS_IN_GPA
    S(I, 6) = S(I, 6)/(2*V0)*USTRESS_IN_GPA
```

**STRESS**

```fortran
TRACE = (S(I, 1) + S(I, 2) + S(I, 3))/3
DEVS11 = S(I, 1) - TRACE
DEVS22 = S(I, 2) - TRACE
DEVS33 = S(I, 3) - TRACE
```
EQSTRESS = DEVS11*DEVS11 + 
& DEVS22*DEVS22 + 
& DEVS33*DEVS33 + 
& 2*S(I,4)*S(I,4) + 
& 2*S(I,5)*S(I,5) + 
& 2*S(I,6)*S(I,6)

VONMISES(I) = DSQRT(3./2.*EQSTRESS)

C       ENDIF

CCC***** End atomic stress *****

40 CONTINUE

FX(NP) = DEDX(NP)*FACTOR
FY(NP) = DEDY(NP)*FACTOR
FZ(NP) = DEDZ(NP)*FACTOR

CCC****** Atomic stress 3 **************

C       IF(N_STRESSFLAG.EQ.1) THEN

S(NP,1) = S(NP,1)/(2*V0)*USTRESS_IN_GPA
S(NP,2) = S(NP,2)/(2*V0)*USTRESS_IN_GPA
S(NP,3) = S(NP,3)/(2*V0)*USTRESS_IN_GPA
S(NP,4) = S(NP,4)/(2*V0)*USTRESS_IN_GPA
S(NP,5) = S(NP,5)/(2*V0)*USTRESS_IN_GPA
S(NP,6) = S(NP,6)/(2*V0)*USTRESS_IN_GPA

TRACE = (S(NP,1) + S(NP,2) + S(NP,3))/3
DEVS11 = S(NP,1) – TRACE
DEVS22 = S(NP,2) – TRACE
DEVS33 = S(NP,3) – TRACE

EQSTRESS = DEVS11*DEVS11 + 
& DEVS22*DEVS22 + 
& DEVS33*DEVS33 + 
& 2*S(NP,4)*S(NP,4) + 
& 2*S(NP,5)*S(NP,5) + 
& 2*S(NP,6)*S(NP,6)

VONMISES(NP) = DSQRT(3./2.*EQSTRESS)

C       ENDIF

CCC***** End atomic stress *****
call vsplint(embed1, embed2, embed_der2, n, x(np), u)
C call uu(X(NP), u, dfunc, d2func)

SUMPOTE = SUMPOTE + u
SUMPOTE = SUMPOTE*FACTOR
c write(*,*), 'SUMPOTE=', SUMPOTE
C*** energy in eV
C write(*,*), 'x001', X00(1), Y00(1), Z00(1)
C write(*,*), 'x002', X00(2), Y00(2), Z00(2)
C write(*,*), 'atomic stress=', S(1,5), S(2,5)

RETURN
END

subroutine vloadtable

IMPLICIT DOUBLE PRECISION (A-H,O-Z)

dimension pair1(3000), pair2(3000), pair_der2(3000)
dimension den1(3000), den2(3000), den_der2(3000)
dimension embed1(3000), embed2(3000), embed_der2(3000)
common/sp_pair/pair1, pair2, pair_der2
common/sp_den/den1, den2, den_der2
common/sp_embed/embed1, embed2, embed_der2

integer elenum, fitmethod, potmethod
character*2 element(6)
common/control/elenum, element, fitmethod, potmethod

open(11, file=element(1)//'.pair')
open(12, file=element(1)//'.den')
open(13, file=element(1)//'.embed')

n = 3000
do 5 i=1, n
   read(11,*) pair1(i), pair2(i)
   read(12,*) den1(i), den2(i)
   read(13,*) embed1(i), embed2(i)
5 continue
yp1 = 1e32
ypn = 1e32
C x = 0.5
call vspline(pair1, pair2, n, yp1, ypn, pair_der2)
call vspline(den1, den2, n, yp1, ypn, den_der2)
call vspline(embed1, embed2, n, yp1, ypn, embed_der2)
C call vsplint(pair1, pair2, pair_der2, n, x, y)
C call vsplintder1(pair1, pair2, pair_der2, n, x, yder1)
c write(*,*), y, yder1
close(11)
SUBROUTINE vspline(x,y,n,yp1,ypn,y2)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
INTEGER n,NMAX
DOUBLE PRECISION yp1,ypn,x(n),y(n),y2(n)
PARAMETER (NMAX=5000)
INTEGER i,k
DOUBLE PRECISION p,qn,sig,un,u(NMAX)
if (yp1.gt.99e30) then
  y2(1)=0.
  u(1)=0.
else
  y2(1)=-0.5
  u(1)=(3./(x(2)-x(1)))*((y(2)-y(1))/(x(2)-x(1))-yp1)
endif
do 11 i=2,n-1
  sig=(x(i)-x(i-1))/(x(i+1)-x(i-1))
  p=sig*y2(i-1)+2.
  y2(i)=(sig-1.)/p
  u(i)=(6.*((y(i+1)-y(i))/(x(i+1)-x(i))-x(i)**1)-x(i))/(x(i)-y(i-1))/(x(i)-x(i-1))/(x(i+1)-x(i-1))-sig*y2(i-1))/p
11  continue
if (ypn.gt.99e30) then
  qn=0.
  un=0.
else
  qn=0.5
  un=(3./(x(n)-x(n-1)))*(ypn-(y(n)-y(n-1))/(x(n)-x(n-1)))
endif
y2(n)=(un-qn*u(n-1))/(qn*y2(n-1)+1.)
do 12 k=n-1,1,-1
  y2(k)=y2(k)*y2(k+1)+u(k)
12  continue
return
END

!!----   Subroutine splint is the Numerical Recipes sister routine
!! for the cubic spline routine, spline. It takes the data
!! points, xa and ya vectors of length n, and the second
!! derivatives computed by spline, y2a, and it computes the
!! y value for the specified x value.
subroutine vsplint(xa, ya, y2a, n, x, y)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
dimension xa(n), ya(n), y2a(n)
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)+
  & ((-3*a**2/h+1./h)*y2a(klo)+(3*b**2/h-1./h)*y2a(khi))*(h**2)/6.
return
end

subroutine vsplintder1(xa, ya, y2a, n, x, yder1)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
dimension xa(n), ya(n), y2a(n)
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
yder1=ya(klo)/h+ya(khi)/h+
  & ((-3*a**2/h+1./h)*y2a(klo)+(3*b**2/h-1./h)*y2a(khi))*(h**2)/6.
return
end
H.24 Reads in Material Property Information (propreader.f)

subroutine propreader(propnum,exact,weight)

C program variables
  double precision  exact(3), weight
  integer  propnum

C global variables
  integer  elenum, fitmethod, potmethod
  character*2 element(6)
  common/control/elenum, element, fitmethod, potmethod

c local variables
  integer i, dummy
  character dumb*12

C begin of pure element code
  if (elenum .eq. 1) then

C Open File to read in known Property Values used for calculating srfit
  open (25, file='./database/'//element(1)//'knownvalues.txt')

C Open file to read known weights used for calculating srfit
  open (26, file='./database/'//element(1)//'weightvalues.txt')

C Skipping over the two line header of the file 25 and 26
  read (25,*) dumb
  read (25,*) dumb
  read (26,*) dumb
  read (26,*) dumb

C do loop to go through files to extract the correct values
  do 10 i=1,propnum
    if (i .eq. 1) then
      read (25,*) dummy, exact(1), exact(2), exact(3)
      read (25,*) dumb
    elseif (i .eq. 7) then
      read (25,*) dummy
      read (25,*) dumb
    else
      read (25,*) dummy, exact(1)
      read (25,*) dumb
    endif
    read (26,*) dummy, weight
    read (26,*) dumb
  10 continue

  if (propnum .eq. 1) then
    write (*,*) 'exact values', exact(1), exact(2), exact(3)
  elseif (propnum .eq. 7) then
else
    write(*,*) 'exact value',exact(1)
endif

close (25)
close (26)
return
C end of pure element code
endif
end

H.25 Calculates the Rose Equation of State (roseeos.f)

subroutine roseeos (F,rose)
implicit none
C global variables
  double precision rcut, a0
  COMMON/pot_fitting/a0,rcut
double precision ecoh,bulk,omega
common/matprop/ecoh,bulk,omega
integer elenum, fitmethod, potmethod
character*2 element(6)
common/control/elenum, element, fitmethod, potmethod
C Program variables
  double precision F(24), rose(24)
c local variables
  double precision eqpos(500,3),pos(500,3)
double precision maxlat, minlat, step
double precision epair, rhobar, EU
double precision a, ecohj, latmult
double precision epair2(3000),rho2(3000),F2(3000),dist1(3000)
double precision epair1(3000), rho1(3000), F1(3000),
dist2(3000)
double precision rho(3000),rhobarv(3000), epairv(3000),
Fv(3000)
double precision edummy, rhodummy, Fdummy, r
integer i,k,j,steps,atoms

C # of times Rose Equation of State will be calculated
steps=24

C # of atoms
atoms=500

C reading atoms from file for equilibrium position
open (1,file='./fcc.xyz',status='old')
doi=1,atoms
read (1,*) eqpos(i,1), eqpos(i,2), eqpos(i,3)
10 continue
close (1)

ecohj=-1*ecoh*1.60217733d-19
omega=a0**3/4*1d-30

open (9,file=element(1)//'.embed')
open (10,file=element(1)//'.pair')
open (11,file=element(1)//'.den')
c read in embed, pair and density tables and set up as a spline
  do 20 i=1,3000
    read (10,*) dist1(i),epairv(i)
    read (11,*) dist2(i),rho(i)
    read (9,*) rhobarv(i), Fv(i)
  20 continue
  call zspline(dist1, epairv, 3000, 1d32, 1d32, epair2)
  call zspline(dist2, rho, 3000, 1d32, 1d32, rho2)
  call zspline(rhobarv, Fv, 3000, 1d32, 1d32, F2)

C setting up the parameters to calculate the Embedding Energy F(i) and
C Rose Et al EOS, rose(i)
  maxlat =1.4d0
  minlat=.8d0
  step=(maxlat-minlat)/(steps-1)
  latmult=maxlat

  do 70 j=1,steps
    do 90 k=1,atoms
      pos(k,1)=latmult*eqpos(k,1)
      pos(k,2)=latmult*eqpos(k,2)
      pos(k,3)=latmult*eqpos(k,3)
  90 continue
  epair=0d0
  rhobar=0d0

C loop to calculate the pair potential energy, rhobar and embedded energy
  do 80 i=1,atoms
    r=sqrt(pos(i,1)**2+pos(i,2)**2+pos(i,3)**2)
    if (r .ge. rcut) then
      epair=epair
rhobar=rhobar
C When r is equal to 0 keep epair and rhobar the same

    elseif (r .eq. 0) then
      epair=epair
      rhobar=rhobar

    else
      C in this section is where robar and epair are calculated.
      call zsplint(dist1, epairv, epair2, 3000, r, edummy)
      epair=epair+edummy

      call zsplint(dist2, rho, rho2, 3000, r, rhodummy)
      rhobar=rhobar+rhodummy

    endif

80     continue
C calculates the rose et al energy, rose(i), c the embedding energy should the material follow the Universal Eos
    a=(latmult-1d0)/sqrt(ecohj/(9*bulk*omega))
    Eu=ecoh*(1+a)*exp(-a)
    rose(j)=EU-.5d0*epair
C calculates the actual Emebedding Energy at the given rhobar.
    call zsplint(rhobarv, Fv, F2, 3000, rhobar, Fdummy)
    F(j)=Fdummy
    latmult=latmult-step

70     continue
C    write (*,*) 'F is', F

      close (10)
      close (11)
      close (9)

      return
end

H.26 Calculating Cubic Splines (splinehandler.f)
C    Cubic Spline function from Numerical Recipe
C
SUBROUTINE zspline(x, y, n, yp1, ypn, y2)
IMPLICIT REAL*8 (A-H,O-Z)
INTEGER n,NMAX
REAL*8 yp1,ypn,x(n),y(n),y2(n)
PARAMETER (NMAX=5000)
INTEGER i,k
REAL*8 p,qn,sig,un,u(NMAX)
if (yp1.gt..99e30) then
  y2(1)=0.
  u(1)=0.
else
  y2(1)=-0.5
  u(1)=(3./(x(2)-x(1)))*((y(2)-y(1))/(x(2)-x(1))-yp1)
endif
do 11 i=2,n-1
  sig=(x(i)-x(i-1))/(x(i+1)-x(i-1))
  p=sig*y2(i-1)+2.
  y2(i)=(sig-1.)/p
  u(i)=(6.*((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
if (ypn.gt..99e30) then
  qn=0.
  un=0.
else
  qn=0.5
  un=(3./(x(n)-x(n-1)))*(ypn-(y(n)-y(n-1))/(x(n)-x(n-1)))
endif
y2(n)=(un-qn*u(n-1))/(qn*y2(n-1)+1.)
do 12 k=n-1,1,-1
  y2(k)=y2(k)*y2(k+1)+u(k)
12 continue
return
END

C Cubic Spline Function for calculating Cubic Splines which have
  a set first and second
c derivative at a specific point along the spline.
C the routine breaks the spline in two sections one before the
  known derivative and
c one after the known derivative.
C The function takes the x and y values (xlist and ylist); number
  of x and y values (n);
c the point which has the known derivatives (known); and the
  known derivatives(der, der2),
c and calculates the 2nd derivatives (y2list) and a list of the
  first derivatives (b).
Subroutine specialspline(xlist, ylist, n, known, der, der2,
y2list,b)

implicit none
INTEGER n, known
double precision xlist(n), ylist(n), der, der2, y2list(n)
C local variables
double precision b(n), c(n), d(n), h(n)
integer i
C This is the section after known
if (known .eq. n) goto 1
do 10 i=known,n
  if (i .eq. known) then
    b(i)=der
    c(i)=der2/2d0
    y2list(i)=der2
    h(i)=xlist(i+1)-xlist(i)
    b(i+1)=3/h(i)*(ylist(i+1)-ylist(i))-2*b(i)-
    c(i)*h(i)
    d(i)=(2*(ylist(i)-ylist(i+1))+b(i+1)*h(i)
+2*b(i)*h(i))/h(i)**2
  elseif (i .eq. n) then
    y2list(i)=2*c(i-1)+6*d(i-1)*h(i-1)
  else
    c(i)=(2*c(i-1)+6*d(i-1)*h(i-1))/2d0
    y2list(i)=2*c(i)
    h(i)=xlist(i+1)-xlist(i)
    b(i+1)=3/h(i)*(ylist(i+1)-ylist(i))-2*b(i)-
    c(i)*h(i)
    d(i)=(2*(ylist(i)-ylist(i+1))+b(i+1)*h(i)
+2*b(i)*h(i))/h(i)**2
  endif
10 continue
C This is the section before known
if (known .eq. 1) goto 2
1 do 20 i=known,1,-1
  if (i .eq. known) then
    y2list(i)=der2
    b(i)=der
    c(i)=der2/2d0
  else
    h(i)=xlist(i+1)-xlist(i)
    d(i)=(2*c(i+1)-2/h(i)*b(i+1)+2/h(i)**2*(ylist(i
+1)-ylist(i)))/(2*h(i))
    c(i)=(-2*d(i)*h(i)**2+b(i+1)+1/h(i)*(ylist(i-
    ylist(i+1)))/h(i)
    b(i)=(ylist(i+1)-ylist(i)-c(i)*h(i)**2-
    d(i)*h(i)**3)/h(i)
    y2list(i)=2*c(i)
  endif
20 continue
subroutine specialsplint(xlist,ylist,b,n,x,y)
imPLICIT none
INTEGER n
double precision xlist(n), ylist(n), x, y, b(n)
C local variables
double precision c, d, h
integer klo, khi, k, i
klo=1
khi=n
1 if (khi-klo.gt.1) then
  k=(khi+klo)/2
  if(xlist(k).gt.x) then
    khi=k
  else
    klo=k
  endif
  goto 1
endif
i=klo
h=xlist(i+1)-xlist(i)
c=(3/h*(ylist(i+1)-ylist(i))-2*b(i)-b(i+1))/h

d=(2*(ylist(i)-ylist(i+1))+b(i+1)*h+b(i)*h)/h**3
y = ylist(i)+b(i)*(x-xlist(i))+c*(x-xlist(i))**2+d*(x-xlist(i))**3
return
end

subroutine specialsplintder1(xlist,ylist,b,n,x,yder)
INTEGER n
double precision xlist(n), ylist(n), x, yder, b(n)
C local variables
double precision c, d
integer klo, khi, k
klo=1
khi=n
1 if (khi-klo.gt.1) then
  k=(khi+klo)/2
  if(xlist(k).gt.x) then
    khi=k
  else
    klo=k
  endif
  goto 1
endif

i=klo
h=xlist(i+1)-xlist(i)
c=(3/h*(ylist(i+1)-ylist(i))-2*b(i)-b(i+1))/h
\( d=(2*(ylist(i)-ylist(i+1))+b(i+1)*h+b(i)*h)/h^3 \)
y2der = b(i) + 2*c*(x-xlist(i)) + 3*d*(x-xlist(i))^2
return
end

subroutine specialsplintder2(xlist,ylist,b,n,x,y2der)
INTEGER n
double precision xlist(n), ylist(n), x,y2der, b(n)
C local variables

double precision c, d
integer klo,khi,k
klo=1
khi=n

if (khi-klo.gt.1) then
  k=(khi+klo)/2
  if(xlist(k).gt.x)then
  khi=k
  else
  klo=k
  endif
  goto 1
endif

i=klo
h=xlist(i+1)-xlist(i)
c=(3/h*(ylist(i+1)-ylist(i))-2*b(i)-b(i+1))/h
\( d=(2*(ylist(i)-ylist(i+1))+b(i+1)*h+b(i)*h)/h^3 \)
y2der = 2*c + 6*d*(x-xlist(i))
return
end

---

Subroutine splint is the Numerical Recipes sister routine
for the cubic spline routine, spline. It takes the data
points, xa and ya vectors of length n, and the second
derivatives computed by spline, y2a, and it computes the
y value for the specified x value.

subroutine zsplint(xa, ya, y2a, n, x, y)
IMPLICIT REAL*8 (A-H,O-Z)
dimension xa(n),ya(n),y2a(n)
klo=1
khi=n

if (khi-klo.gt.1) then
k=(khi+klo)/2
if(xa(k).gt.x)then
  khi=k
else
  Klo=k
endif
goto 1
dendif

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.
return
dend

subroutine zsplintder1(xa,ya,y2a,n,x,yder)
IMPLICIT REAL*8 (A-H,O-Z)
dimension xa(n),ya(n),y2a(n)
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
dendif

h=xa(khi)-xa(klo)
if (h.eq.0.) pause 'bad xa input.'
a=(xa(khi)-x)/h
b=(x-xa(klo))/h
yder1=ya(klo)/h+ya(khi)/h+
& ( (-3*a**2/h+1./h)*y2a(klo)+(3*b**2/h-1./h)*y2a(khi))*(h**2)/6.
return
dend

subroutine zsplintder2(xa,ya,y2a,n,x,yder)
IMPLICIT REAL*8 (A-H,O-Z)
dimension xa(n),ya(n),y2a(n)
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
yder2= ((6*a/h**2)*y2a(klo)+(6*b/h**2)*y2a(khi))*(h**2)/6.
return
end

H.27 Calculating the Potential Using Knots and Cubic Splines

(splinemethod.f)

subroutine splinemethod (const1,sr)
implicit none
c global variables
  integer a, b, c, m, n
  common/parameters/a,b,c,m,n
double precision a0, rcut
  COMMON/pot_fitting/a0,rcut
double precision ecoh, bulk, omega
  common/matprop/eco,bulk,omega
  integer elenum, fitmethod, potmethod
  character*2 element(6)
  common/control/elenum, element, fitmethod, potmethod
c program variables
  double precision const1(m), sr
  c local variables
  double precision eqpos(500,3),pos(500,3)
double precision epair, rho, rhobar, epairsum, F, step,
  latmult, delta
double precision depair, drho, d2epair, d2rho, dF, d2F
double precision Flist(c),derivF(c), F2list(c), rhobarlist(c)
double precision rholist(b), rho2list(b), rrho(b)
double precision epairlist(a), epair2list(a), repair(a)
double precision known(2),scale1(a-2), scale2(b-2),
  scale3(c-2), scale(a+b+c-6), eqscale
  integer points,lesspoints
parameter (points=3000, lesspoints=3000)
double precision specrho(points),specd2rho(points)
double precision specr(points)
double precision smoothrho(lesspoints), smoothd2rho(lesspoints)
  double precision smoothr(lesspoints), smoothd2rho(lesspoints)
  double precision smoothpair(lesspoints), smoothd2pair(lesspoints)
  double precision r, rmin, rhobarmax, x, psi, s
  double precision depairsum, d2epairsum, drhosum, d2rhosum
CHARACTER   FILENAME*25
integer i,k,j, fslot

C code for mishin's potential creation method
if (potmethod .eq. 1) then
C code for pure element
  if (elenum .eq. 1) then

C check to insure paramaters(const1) are possible.
C An impossible set of parameters would have a negative rcut
  [const1(m)] or
C negative rho [const1(a-1:a+b-4)
C If they are not possible create ludicrously high sr and skip to
  end of this program.
C High sr should cause minimizer to not choose that value.

  do 85 k=a-1,a+b-4
     if (const1(k) .le. 0d0) then
       write (*,*) 'bad rho parameter set high sr value'
       sr=1d32
       goto 65
  endif
  continue
  if (const1(m) .le. 0d0) then
  write (*,*) 'bad rcut parameter'
  sr=1d32
  goto 65
  endif

C this file is read to get the equilibrium atomic
positions.
  open (1,file='./fcc.xyz',status='old')
  do 10 i=1,500
     read (1,*) eqpos(i,1), eqpos(i,2), eqpos(i,3)
     write (*,*) eqpos(i,1), eqpos(i,2), eqpos(i,3)
  10 continue
c open (12,file='pair.txt')
c open (13,file='den.txt')
c open (14,file='embed.txt')
c opening files to store final cubic spline information
  open (9,file=element(1)/'.embed')
  open (10,file=element(1)/'.pair')
  open (11,file=element(1)/'.den')
  open (15,file=./database/element(1)/'knownconst.txt',
       status='old')
    read (15,*) known

C calculating omega needed for calculating first and 2nd
derivative of embedding energy
    omega=a*0**3/4*1d-30
C do i really need omega in m^3

C needed parameters for creating the x lists
    rmin=1d0
    rcut=const1(m)
    rhobarmax=2d0

C code to open the right scale file in the database folder
    FILENAME(1:18) = './database/element(1)/scale'
    FILENAME(19:19) = char(a+48)
    FILENAME(20:20) = char(b+48)
    FILENAME(21:21) = char(c+48)
    FILENAME(22:25) = '.txt'
    open (45,file=FILENAME,status='old')
    read (45,*) scale
    close (45)

    do 100 i=1,a-2
       scale1(i)=scale(i)
     100 continue

    c write (*,*) 'scale1 is', scale1

    do 110 i=1,b-2
       scale2(i)=scale(a+i-2)
     110 continue

    c write (*,*) 'scale2 is', scale2

    do 120 i=1,c-2
       scale3(i)=scale(a+b+i-4)
     120 continue

    c write (*,*) 'scale 3 is', scale3
C Creating a list of the x values for the pair energy and the electron density
    delta=r-cut-rmin
    repair(1)=rmin
do 40 i=2,a
        if (i .eq. a) then
            repair(i)=rcut
        else
            repair(i)=rmin+scale1(i-1)*delta
        endif
40 continue
    rrho(1)=rmin
    do 50 i=2,b
        if (i .eq. b) then
            rrho(i)=rcut
        else
            rrho(i)=rmin+scale2(i-1)*delta
        endif
50 continue
C Creating a list of the y values for the pair energy and the electron density
do 20 i=1,a
    if (i .eq. a) then
        epairlist(i)=0d0
    elseif (i .eq. 1) then
        epairlist(i)=known(1)
    else
        epairlist(i)=const1(i-1)
    endif
    write (12,*) repair(i),epairlist(i)
20 continue
    do 30 i=1,b
        if (i .eq. b) then
            rholist(i)=0d0
        elseif (i .eq. 1) then
            rholist(i)=known(2)
        else
            rholist(i)=const1(i+a-3)
        endif
    c    write (13,*) rrho(i),rholist(i)
30 continue
C Create Regular cubic splines for the pair potential and electron density
    call zspline(repair,epairlist,a,1d32,1d32,epair2list)
call zspline(rrho,rholist,b,1d32,1d32,rho2list)
c write (*,*) 'The 2nd derivatives for epair are', epair2list
c write (*,*) 'the 2nd derivatives for rho are', rho2list

C Calculate 1st derivative of pair potential and electron density at rcut
  call zsplintder1(repair,epairlist,epair2list,a,rcut,depair)
call zsplintder1(rrho,rholist,rho2list,b,rcut,drho)
c write (*,*) 'depair at rcut is', depair
c write (*,*) 'drho at rcut is', drho

C Zero 1st and 2nd derivative of pair potential and electron Density.
  step=(rcut-rmin)/(lesspoints-1)
r=rmin
do 95 i=1,lesspoints
  x=r-rcut
  psi=x/(1+2**4*x**4)
  psi=0d0
  if (i .eq. lesspoints) then
    smoothr(i)=rcut
    smoothepair(i)=0d0
  else
    call zsplint(repair,epairlist, epair2list,a,r,epair)
    smoothr(i)=r
    smoothepair(i)=epair-depair*psi
  endif
  write (10,200) smoothr(i), smoothepair(i)
c write (10,*) smoothr(i), smoothepair(i)
  if (i .eq. lesspoints) then
    smoothr(i)=rcut
    smoothrho(i)=0d0
  else
    call zsplint(rrho,rholist,rho2list,b,r,rho)
    smoothr(i)=r
    smoothrho(i)=rho-drho*psi
  endif
  write (11,200) smoothr(i), smoothrho(i)
c write (11,*) smoothr(i),smoothrho(i)
200 format (D30.23,D30.23)
r=r+step
95 continue

C Create zeroed 1st and 2nd derivative splines for pair potential and electron density with x points.
call zspline(smoothr,smoothepair,lesspoints, 1d32,1d32,smoothd2epair)
call zspline(smoothr,smoothrho,lesspoints, 
1d32,1d32,smoothd2rho)
c write (*,*) 'The 2nd derivatives for epair are', 
smoothd2epair(1)
c write (*,*) 'the 2nd derivatives for rho are', smoothd2rho(1)
c write (*,*) 'The 2nd derivatives for epair are', 
smoothd2epair(3000)
c write (*,*) 'the 2nd derivatives for rho are', 
smoothd2rho(3000)

C Setting up atomic positions of Equilibrium Structure
latmult=1d0
!
do 5 k=1,500
  pos(k,1)=latmult*eqpos(k,1)
  pos(k,2)=latmult*eqpos(k,2)
  pos(k,3)=latmult*eqpos(k,3)
5 continue

C Setting up loop to calculate Pair Potential Energy, Rhobar, 
C and derivatives of Potential Energy
epairsum=0d0
rhold=0d0
depairsum=0d0
d2pairsum=0d0

c loop to calculate the pair potential energy, rhobar and 
embedded energy and derivatives of potential energy.
do 15 i=1,500
  r=sqrt(pos(i,1)**2+pos(i,2)**2+pos(i,3)**2)
  
c if r is greater or equal to the cutoff than the pair energy and 
rhobar do not change
  if (r .ge. rcut) then
    epairsum=epairsum
    rhobar=rhobar
    depairsum=depairsum
    d2pairsum=d2pairsum
  c
  write (18,*) 'd2pairsum', d2pairsum
  c
drhosum=drhosum
  c
d2rhosum=d2rhosum
  c
  write (18,*) 'd2rhosum',d2rhosum
  c
  When r is equal to 0,
  C Keeps rhobar pair potential and derivatives of pair potential 
energy from blowing up.
  elseif (r .eq. 0) then
    epairsum=epairsum
    rhobar=rhobar
depairsum=depairsum
d2epairsum=d2epairsum
c          write (18,*) 'd2epairsum', d2epairsum
c          drhosum=drhosum
c          d2rhosum=d2rhosum
c          write (18,*) 'd2rhosum', d2rhosum
else
C rhobar, epair, d2epair, and depair are calculated in this section
c          write (18,*) 'r is', r
          call
          zsplint(smoothr, smoothepair, smoothd2epair, lesspoints, r, epair)
          epairsum=epair+epairsum
          call
          zsplint(smoothr, smoothrho, smoothd2rho, lesspoints, r, rho)
          rhobar=rho+rhobar
          call
          zsplintder1(smoothr, smoothepair, smoothd2epair, lesspoints, r, depair)
          depairsum = depair*r + depairsum
          call
          zsplintder2(smoothr, smoothepair, smoothd2epair, lesspoints, r, d2epair)
          d2epairsum = d2epair*r**2 + d2epairsum
          endif

15 continue

c Check to see if rhobar is less than 0. If so give the calculated values sufficiently high values
and goto end of program
c          if (rhobar .le. 0d0) then
          write (*,*) rhobar
c          c11=1d6
c          c12=1d6
c          c44=1d6
c          e_vacancy=1d3
c          goto 65
c          endif

C Calculating F
          F=ecoh-.5d0*epairsum
          write (*,*) 'rhobar is', rhobar
          write (*,*) 'F is', F
C calculating s for rho*s transform
   if (rhobar.eq. 0) then
      s=1
   else
      s=1d0/rhobar
   endif
write (*,*) 's is', s

C if rhobar lt 0 close open files, sr set to 1d32 and exit routine.
   if (rhobar .lt. 0d0) then
      sr=1d32
      write(*,*) 'bad rhobar'
C closing all of the open files
   close (10)
   close (11)
   close (9)
c   close (12)
c   close (13)
c   close (14)
c   close (1)
c   close (15)
goto 65
   endif

C creates 3000 point data table for rho
C **** rho*s transform not implemented
C Later use data for calculations of drhobar and d2rhobar.
   step=(rcut-rmin)/(points-1)
   r=rmin
   do 90 i=1,points
      if (i .eq. points) then
         specr(i)=rcut
         specrho(i)=0d0
      else
         call zsplint(smoothr,smoothrho,smoothd2rho,lesspoints,r,rho)
         rho=rho*s
         specr(i)=r
         specrho(i)=rho
      endif
   c   write (11,200) specr(i), specrho(i)
   write (11,*) specr(i),specrho(i)
   r=r+step
90 continue

C make spline of new rho values for calculation of 1st and 2nd derivatives of rho.
C Setting up loop to calculate derivatives of electron density
drhosum=0d0
d2rhosum=0d0

C Calculate derivative of rho from new rho values
do 75 i=1,500  
r=sqrt(pos(i,1)**2+pos(i,2)**2+pos(i,3)**2)

C if r is greater or equal to the cutoff than the pair energy and rhobar do not change
    if (r .ge. rcut) then
        drhosum=drhosum
        d2rhosum=d2rhosum
        write (18,*) 'd2rhosum',d2rhosum
    endif

C When r is equal to 0 to keep the paire and rhobar from blowing up paire and rhobar do not change
    elseif (r .eq. 0) then
        drhosum=drhosum
        d2rhosum=d2rhosum
        write (18,*) 'd2rhosum',d2rhosum
    else
        write (18,*) 'r is', r
        call zsplintder1(specr, specrho, specd2rho, 
        points, r, drho)
        drhosum = drho*r + drhosum

        call zsplintder2(specr, specrho, specd2rho, 
        points, r, d2rho)
        write (18,*) 'r is', r
        write (18,*) 'r**2 is', r**2
        write (18,*) 'd2rho is', d2rho
        d2rhosum = d2rho*r**2 + d2rhosum
        write (18,*) 'd2rhosum',d2rhosum
    endif

    75 continue

    if (drhosum .eq. 0) then
        C dF and d2F in case drhosum is 0
        dF=0
        d2F=0
else
C Calculating dF
   dF = -.5*depairsum/drhosum
C Calculating d2F
   d2F = (9*bulk*omega*6.24150648E+018 -.5*d2epairsum -
   dF*d2rhosum)/(drhosum**2)
endif

write (*,*) 'dF is', dF
write (*,*) 'd2F is', d2F
write (*,*) epairsum,depairsum,d2epairsum,drhosum,d2rhosum

C Create 3000 point data table for pair potential
C ***** V + 2*dF*rho/s transform not implemented
C ***** transform is divided by s to get the pretransformed rho
C ***** g=df in above transform
r=rmin
do 80 i=1,points
   if (i .eq. points) then
      r=rcut
      epair=0
   else
      call zsplint(smoothr,smoothepair,smoothd2epair,lesspoints,r,epair)
      call zsplint(specr,specrho,specd2rho,points, r, rho)
      epair=epair+2*dF*rho/s
   endif
   write (10,*) r, epair
write (10,200) r,epair
r=r+step
80 continue

c rhobar at equilibrium has been set to 1 through the s transform.
c rhobarlist and scale 3 already have the s transform built in.
c F and rhobar at equilibrium are placed in the corresponding
Flist and rhobarlist
c at int(c/2)+1. The rhobar is set into the correct slot by
placing .5 in the right
c position in scale3.

C Creating a list of the y values for the embedding energy
Flist(1)=0d0
do 25 i=1,c-1
   if (i .eq. int(c/2)) then
      Flist(i+1)=F
   elseif (i .lt. int(c/2)) then
      Flist(i+1)=const1(i+a+b-4)
   else

Flist(i+1)=const1(i+a+b-5)

endif

25 continue

C Creating a list of the x values for the embedding energy
delta=2
rhoparlist(1)=0d0
c write (14,*) rhobarlist(1),Flist(1)
do 35 i=2,c
  if (i .eq. c) then
    rhobarlist(i)=rhobarmax
  else
    rhobarlist(i)=scale3(i-1)*delta
  endif
c write (14,*) rhobarlist(i),Flist(i)
c write (14,200) rhobarlist(i),Flist(i)
35 continue

C Calculate cubic spline for embedding energy with F0, dF and d2F
included in the spline.
C Then create a 3000 point data table using the splines.
c ***** F=F-dF*rhobar/s transformation not implemented
C ***** transform is divided by s to get the pretransformed rho
C ***** g=df in above transform

    call specialspline(rhobarlist,Flist,c,int(c/2)+1,dF,d2F,F2list,derivF)
c write (*,*) 'the 1st derivatives of F are', derivF
c write (*,*) 'the 2nd derivatives of F are', F2list
    step=(rhobarmax-0)/(points-1)
rhobar=0
do 55 i=1,points
    call specialsplint(rhobarlist,Flist,derivF,c,rhobar,F)
c    call zsplint(rhobarlist,Flist,F2list,c,rhobar,F)
c    F = F - dF*rhobar/s
    write (9,*) rhobar, F
c write (9,200) rhobar,F
rhopar=rhopar+step
55 continue

C closing all of the open files
    close (10)
close (11)
close (9)
c close (12)
c close (13)
c close (14)
close (1)
close (15)

358
This section calls a program to calculate the sum of residuals for the calculated properties. The properties that are calculated are C11, C12, C44.

call srsolvefit(sr)

return

C end of code for pure element
endif

C end of code for Mishin's potential creation method
endif
end

H.28 Solving the Sr for the Fitting Properties (srsolvefit.f)

subroutine srsolvefit(sr)
implicit none
C program variables
double precision sr
C Global Variables
integer a,b,c,m,n
common/parameters/a,b,c,m,n
double precision c11,c12,c44,e_vacancy
common/fit/c11,c12,c44,e_vacancy
double precision mass, at
common/mdprop/mass,at
double precision ecoh, bulk, omega
common/matprop/ecoh,bulk,omega
integer switch
common/roseswitch/switch
integer endflag
common/finalswitch/endflag
integer elenum, fitmethod, potmethod
character*2 element(6)
common/control/elenum, element, fitmethod, potmethod
C local variables
integer i, steps, propnum, j, jmax
double precision dummy, sum, y(3), exact(3), weight, F(24), rose(24)
double precision surf100, surf110, surf111, hcpenergy, stable, unstable
double precision texp, tcond
C the value n is the number of properties calculated in srsolvefit
C which has been defined in pot.f and located in the first line in the fit.txt file.
C The rest of the lines in the fit.txt file correspond to a property #.
c These property #'s are used to calculate the needed properties.
C The only exception is the Rose et al. eos. which requires both a property # (7)
c and switch value (1) to execute.
C Switch is currently set to 1 during the longneldermead calculation
C and 0 during the shortneldermead calculation.

C write down final values from fitting procedure
    if (endflag .eq. 1) write (95,*) 'final fitting calculations
    for a=',a,', b=', b,', c=', c

C setups equilibrium properties and lammps friendly potential if needed.
call mdsetupfit

C Open file to read property # used for executing correct
property calculations
    open (29, file='./controls/fit.txt')
C dummy read used to skip to the actual property #s in the file
    read (29,*) propnum

    i=1
    sum=0d0

C start of pure element code
    if (elenum .eq. 1) then

C while loop where properties and the Sum of Residuals Squared of
the fit (srfit) are calculated
    12 if (i .le. n) then
C reads in the prop # from fit.txt and executes the corresponding
property calculation
        read (29,*) propnum
C Execute Elastic Property Calculations when property # is 1
    if (propnum .eq. 1) then
        call elastic

        write (*,*) 'c11,c12,c44'
        write (*,*) c11,c12,c44

        if (endflag .eq. 1) then
            write (95,*) 'c11,c12,c44'
            write (95,*) c11,c12,c44
        endif

        call propreader(propnum, exact, weight)

    if (endflag .eq. 1) then
write (95,*) 'exact values', exact(1),
exact(2), exact(3)
endif
y(1)=c11
y(2)=c12
y(3)=c44
jmax=3
C call to calculate sr for this property because it has multiple properties
goto 13

C Execute Vacancy Formation Energy Calculation when property # is 2
elseif (propnum .eq. 2) then
call vacancy(ecoh)

write (*,*) 'vacancy energy'
write (*,*) e_vacancy

if (endflag .eq. 1) then
    write (95,*) 'vacancy energy'
    write (95,*) e_vacancy
endif

call propreader(propnum, exact, weight)

if (endflag .eq. 1) then
    write (95,*) 'exact value',exact(1)
endif
y(1)=e_vacancy
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C Execute 100 Surface Energy when property # is 3
elseif (propnum .eq. 3) then
call surface100(surf100)

write (*,*) '100 surface energy'
write (*,*) surf100

if (endflag .eq. 1) then
    write (95,*) '100 surface energy'
    write (95,*) surf100
endif

call propreader(propnum, exact, weight)
if (endflag .eq. 1) then
  write (95,*) 'exact value',exact(1)
endif
y(1)=surf100
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C Execute 110 Surface Energy when property # is 4
elseif (propnum .eq. 4) then
call surface110(surf110)

write (*,*) '110 surface energy'
write (*,*) surf110

if (endflag .eq. 1) then
  write (95,*) '110 surface energy'
  write (95,*) surf110
endif
call propreader(propnum, exact, weight)

if (endflag .eq. 1) then
  write (95,*) 'exact value',exact(1)
endif
y(1)=surf110
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C Execute 111 Surface Energy when property # is 5
elseif (propnum .eq. 5) then
call surface111(surf111)

write (*,*) '111 surface energy'
write (*,*) surf111

if (endflag .eq. 1) then
  write (95,*) '111 surface energy'
  write (95,*) surf111
endif
call propreader(propnum, exact, weight)
if (endflag .eq. 1) then
    write (95,*) 'exact value', exact(1)
endif

y(1)=surf111
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C Execute Hexagonal Close Pack Energy when property # is 6
elseif (propnum .eq. 6) then
    call hexagonalclosepackenergy(hcpenergy)

    write (*,*) 'hexagonal close pack energy'
    write (*,*) hcpenergy

    if (endflag .eq. 1) then
        write (95,*) 'hexagonal close pack energy'
        write (95,*) hcpenergy
    endif

call propreader(propnum, exact, weight)

    if (endflag .eq. 1) then
        write (95,*) 'exact value', exact(1)
    endif

    y(1)=hcpenergy
    dummy=(y(1)-exact(1))/exact(1)
    sum = weight*dummy**2 + sum

    i=i+1
goto 12

C Execute Universal EOS of Rose et. al when property # is 7 and switch is 1
elseif (propnum .eq. 7) then
    if (switch .eq. 1) then
        call roseeos(F, rose)
        write (*,*) 'roseeos finished'
        call propreader(propnum, exact, weight)

C set up number of steps for RoseEOS Sr calculation.
    steps=24
C calculate Sr of Universal EOS of Rose et al and add to previous SR.
    do 30 j=1,steps
        dummy=(F(j)-rose(j))/(rose(j))
        sum=weight*dummy**2+sum
    30 continue
continue
i=i+1
goto 12
else
i=i+1
goto 12
endif

C Execute Stable Stacking Fault when property # is 8
elseif (propnum .eq. 8) then
call stablefault(stable)
write (*,*) 'Stable Stacking Fault'
write (*,*) stable
if (endflag .eq. 1) then
write (95,*) 'Stable Stacking Fault'
write (95,*) stable
endif
call propreader(propnum, exact, weight)
if (endflag .eq. 1) then
write (95,*) 'exact value',exact(1)
endif

y(1)=stable
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C Execute Unstable Stacking Fault when property # is 9
elseif (propnum .eq. 9) then
call unstablefault(unstable)
write (*,*) 'unstable Stacking Fault'
write (*,*) unstable
if (endflag .eq. 1) then
write (95,*) 'unstable Stacking Fault'
write (95,*) unstable
endif
call propreader(propnum, exact, weight)
if (endflag .eq. 1) then
write (95,*) 'exact value',exact(1)
endif
y(1)=unstable
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C Execute Thermal Expansion MD calculation when property # is 10
elseif (propnum .eq. 10) then
call system('python texp.py')
open (99, file='ans.txt')
read (99,*) texp, y(2)
close (99)

write (*,*) 'thermal expansion, room temperature lattice constant'
write (*,*) texp, y(2)

if (endflag .eq. 1) then
write (95,*) 'thermal expansion, room temperature lattice constant'
write (95,*) texp, y(2)
endif

call propreader(propnum, exact, weight)

C propreader only reads values from knownvalues.txt and weightvalues.txt
C the room temperature lattice constant is located in a different file
C so exact(2) will be defined here rather than in propreader
exact(2)=at
if (endflag .eq. 1) then
write (95,*) 'exact values', exact(1),

exact(2)
endif
y(1)=texp

C y(2) allready defined when ans.txt was read
jmax=2

C call to calculate sr for this property because it has multiple properties
goto 13

C Execute Thermal Conductivity MD calculation when property # is 11
elseif (propnum .eq. 11) then
call system('python tcond.py')
open (99, file='ans.txt')
read (99,*) tcond
close (99)

write (*,*) 'thermal conductivity'
write (*,*) tcond

if (endflag .eq. 1) then
    write (95,*) 'thermal conductivity'
    write (95,*) tcond
endif

call propreader(propnum, exact, weight)

if (endflag .eq. 1) then
    write (95,*) 'exact value', exact(1)
endif
y(1)=tcond
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12
endif
C end of while loop
endif

C Final Sr returned by this subroutine is equal to the sum of the individual pieces in the while loop.
sr=sum
C skips the do loop and goto statement below
goto 14
C this do loop calculates the sum of errors squared for individual properties that have more than one calculated and exact value.
13 do 25 j=1,jmax
    write (*,*) 'calculated value', y(j)
    write (*,*) 'exact value', exact(j)
    write (*,*) 'weight', weight(j)
    dummy=(y(j)-exact(j))/exact(j)
    sum = weight*dummy**2 + sum
25 continue
    i=i+1
C goes back to the beginning of the while loop above
    goto 12
14 write (*,*) 'srfit is', sr
    write (*,*)
    if (endflag .eq. 1) then
write (95,*) 'srfit is', sr
write (95,*)
endif
close (29)
return
C end of pure element code
endif
dend

H.29 Solving the Sr for the Testing Properties (srsovltest.f)

subroutine srsolvetest(sr,testnum)
implicit none
c program variables
double precision sr
integer testnum
c Global Variables
integer a,b,c,m,n
common/parameters/a,b,c,m,n
double precision c11,c12,c44,e_vacancy
common/fit/c11,c12,c44,e_vacancy
double precision mass, at
common/mdprop/mass,at
double precision ecoh, bulk, omega
common/matprop/ecohe,bulk,omega
integer elenum, fitmethod, potmethod
character*2 element(6)
common/control/elenum, element, fitmethod, potmethod
local variables
integer i, steps, propnum, j, jmax
double precision dummy, sum, y(3), exact(3),weight, F(24),
rose(24)
double precision surf100, surf110, surf111, hcpenergy,
stable, unstable
double precision texp, tcond
C the value testnum is the number of properties calculated in
srsovlfit
C which has been defined in pot.f and located in the first line
in the test.txt file.
C The rest of the lines in the test.txt file correspond to a
property #.
c These property #s are used to calculate the needed properties.
C file value 27 corresponds to srsolvtest.txt
write (27,*) 'begining Sr Solve Test for a=',a,', b=', b,', c=', c
C setups equilibrium properties and lammps friendly potential if needed. [not implemented]
    call mdsetuptest(testnum)

C Open file to read property # used for executing correct property calculations
    open (29,file='./controls/test.txt')
C dummy read used to skip to the actual property #'s in the file
    read (29,*) propnum

    i=1
    sum=0d0

C begining of pure element code
    if (elenum .eq. 1) then

C while loop where properties and the Sum of Residuals Squared of the fit (srfit) are calculated
    12 if (i .le. testnum) then
C reads in the prop # from fit.txt and executes the corresponding property calculation
    read (29,*) propnum
C Execute Elastic Property Calculations when property # is
    if (propnum .eq. 1) then
        call elastic

        write (*,*) 'c11,c12,c44'
        write (*,*) c11,c12,c44

        write (27,*) 'c11,c12,c44'
        write (27,*) c11,c12,c44

        call propreader(propnum, exact, weight)

        write (27,*) 'exact values', exact(1), exact(2),
        exact(3)
        y(1)=c11
        y(2)=c12
        y(3)=c44
        jmax=3

C call to calculate sr for this property because it has multiple properties
        goto 13

C Execute Vacancy Formation Energy Calculation when property # is
    elseif (propnum .eq. 2) then
        call vacancy(ecoh)
write (*,*) 'vacancy energy'
write (*,*) e_vacancy

write (27,*) 'vacancy energy'
write (27,*) e_vacancy

call propropreader(propnum, exact, weight)

write (27,*) 'exact value',exact(1)
y(1)=e_vacancy
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C Execute 100 Surface Energy when property # is 3
elseif (propnum .eq. 3) then
    call surface100(surf100)

write (*,*) '100 surface energy'
write (*,*) surf100

write (27,*) '100 surface energy'
write (27,*) surf100

call propropreader(propnum, exact, weight)

write (27,*) 'exact value',exact(1)
y(1)=surf100
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C Execute 110 Surface Energy when property # is 4
elseif (propnum .eq. 4) then
    call surface110(surf110)

write (*,*) '110 surface energy'
write (*,*) surf110

write (27,*) '110 surface energy'
write (27,*) surf110

call propropreader(propnum, exact, weight)

write (27,*) 'exact value',exact(1)
y(1)=surf110
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C Execute 111 Surface Energy when property # is 5
elseif (propnum .eq. 5) then
call surface111(surf111)

write (*,*) '111 surface energy'
write (*,*) surf111

write (27,*), surf111
write (27,*), surf111

call proprecader(propnum, exact, weight)

write (27,*), 'exact value', exact(1)
y(1)=surf111
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C Execute Hexagonal Close Pack Energy when property # is 6
elseif (propnum .eq. 6) then
call hexagonalclosepackenergy(hcpenergy)

write (*,*) 'hexagonal close pack energy'
write (*,*) hcpenergy

write (27,*), hcpenergy
write (27,*), hcpenergy

call proprecader(propnum, exact, weight)

write (27,*), 'exact value', exact(1)
y(1)=hcpenergy
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C Execute Universal EOS of Rose et. al when property # is 7
elseif (propnum .eq. 7) then
call roseeos(F,rose)
write (*,*) 'roseeos finished'

write (27,*) 'roseeos finished'

call propreader(propnum, exact, weight)

C set up number of steps for RoseEOS Sr calculation.
steps=24

C calculate Sr of Universal EOS of Rose et al and add to previous SR.
do 30 j=1,steps
   dummy=(F(j)-rose(j))/(rose(j))
   sum=weight*dummy**2+sum
30  continue
   i=i+1
goto 12

C Execute Stable Stacking Fault when property # is 8
elseif (propnum .eq. 8) then
call stablefault(stable)
write (*,*) 'Stable Stacking Fault'
write (*,*) stable
write (27,*) 'Stable Stacking Fault'
write (27,*) stable

call propreader(propnum, exact, weight)
write (27,*) 'exact value',exact(1)
y(1)=stable
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

   i=i+1
goto 12

C Execute Unstable Stacking Fault when property # is 9
elseif (propnum .eq. 9) then
call unstablefault(unstable)
write (*,*) 'unstable Stacking Fault'
write (*,*) unstable
write (27,*) 'unstable Stacking Fault'
write (27,*) unstable

call propreader(propnum, exact, weight)
write (27,*) 'exact value', exact(1)
y(1)=unstable
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C Execute Thermal Expansion MD calculation when property # is 10
elseif (propnum .eq. 10) then
    call system('python texp.py')
    open (99, file='ans.txt')
    read (99,*) texp, y(2)
    close (99)

    write (27,*) 'thermal expansion, room temperature lattice constant'
    write (27,*) texp, y(2)

    call propreader(propnum, exact, weight)

    C propreader only reads values from knownvalues.txt and weightvalues.txt
    C the room temperature lattice constant is located in a different file
    C so exact(2) will be defined here rather than in propreader
    exact(2)=at
    write (27,*) 'exact values', exact(1), exact(2)
    y(1)=texp

    C y(2) allready defined when ans.txt was read
    jmax=2

    C call to calculate sr for this property because it has multiple properties
    goto 13

C Execute Thermal Conductivity MD calculation when property # is 11
elseif (propnum .eq. 11) then
    call system('python tcond.py')
    open (99, file='ans.txt')
    read (99,*) tcond
    close (99)

    write (27,*) 'thermal conductivity'
    write (27,*) tcond

    call propreader(propnum, exact, weight)
 overwrite (27,*) 'exact value', exact(1)
y(1)=tcond
dummy=(y(1)-exact(1))/exact(1)
sum = weight*dummy**2 + sum

i=i+1
goto 12

C end of while loop
endif

c C Final Sr returned by this subroutine is equal to the sum of the
individual pieces in the while loop.
sr=sum

c C skips the do loop and goto statement below
goto 14

c C this do loop calculates the sum of errors squared
for individual properties that have more than one calculated
and exact value.
13 do 25 j=1,jmax
    write (*,*) 'calculated value', y(j)
    write (*,*) 'exact value', exact(j)
    write (*,*) 'weight', weight(j)
dummy=(y(j)-exact(j))/exact(j)
sum = weight*dummy**2 + sum
25 continue
i=i+1
C goes back to the beginning of the while loop above
goto 12

14 write (*,*) 'sr of test is', sr
write (*,*)
write (27,*) 'sr of test is', sr
write (27,*)
close (25)
close (26)
close (29)
return

C End of pure element code
endif
end

H.30 Calculating the Stable Stacking Fault (stablefault.f)

subroutine stablefault(stable)
implicit none
C global variables
double precision a0,rcut


common/pot_fitting/a0,rcut
double precision mass, at
common/mdprop/mass,at
double precision ecoh, bulk, omega
common/matprop/ecoh,bulk,omega
double precision xmin, xmax, zmin, zmax, ymax, ymin
common/boundarycondition/xmin,xmax,zmin,zmax,ymax,ymin
integer max(3), min(3)
common/pbcon/max,min
integer satoms,indexatoms(15),layers
common/specialatomcount/satoms,indexatoms,layers
double precision index(15,150)
common/indexing/index

C program variables
double precision stable
C local variables
integer atoms, atoms2
double precision surface(288,3), crystal(288,3), axis111(288,3), sindex(288)
double precision index2(288), box2_1(288,3), box2(288,3)
double precision row(3), rotoatom(3), rotomat(3,3), rototrans(3,3), rotomat2(3,3), rotomat1(3,3)
double precision bottomsurf(250,3), topsurf(250,3)
double precision zvalues(15), currentz
double precision gammaenergy, surfen0, surfen, energy,
prevgamma
double precision maxx, minx, maxy, miny, nextminy, nextminx,
nextnextminy
double precision height, base, area, area1, area2
integer ymaxloc, yminloc, botsindex, topsindex
double precision step, vectmagnitude, magnitude, h, alpha
double precision const1(10), const2(10), dconst(10), grad(10)
integer i, j, k, l, m, n

c a0=3.6150000000000d0
c rcut=5.50679d0
atoms=288
c e_coh=-3.54d0
c write (*,201) a0
c call struct111(a0)
C read in atom structure and store in crystal
open (1, file='(111).xyz')
do 10 i=1,atoms
    read (1,*) crystal(i,1), crystal(i,2), crystal(i,3)
10 continue
close (1)
c write (*,201) crystal
C rotation matrix1 for rotating by 30 degrees around the z axis
rotomat1(1,1)=sqrt(3d0)/2d0
rotomat1(1,2)=-.5d0
rotomat1(1,3)=0
rotomat1(2,1)=.5d0
rotomat1(2,2)=sqrt(3d0)/2d0
rotomat1(2,3)=0d0
rotomat1(3,1)=0D0
rotomat1(3,2)=0D0
rotomat1(3,3)=1D0
c write (*,*) 'rotomat1 is', rotomat1
c write (*,201) rotomat1
c WRITE (*,*)

C rotation matrix2 for converting axis system to (111) plane
rotomat2(1,1)=-1/sqrt(6d0)
rotomat2(1,2)=-1/sqrt(6d0)
rotomat2(1,3)=2/sqrt(6d0)
rotomat2(2,1)=1/sqrt(2d0)
rotomat2(2,2)=-1/sqrt(2d0)
rotomat2(2,3)=0d0
rotomat2(3,1)=1/sqrt(3d0)
rotomat2(3,2)=1/sqrt(3d0)
rotomat2(3,3)=1/sqrt(3d0)
c write (*,*) 'rotomat2 is', rotomat2
c write (*,201) rotomat2
c write (*,*)

c calculate rotation matrix (rotomat) which is rotomat1*rotomat2
  DO 2 I = 1, 3
    DO 2 J = 1, 3
      rotomat(I,J) = 0D0
      DO 1 K = 1, 3
        rotomat(I,J) = rotomat(I,J) + rotomat1(I,K) * rotomat2(K,J)
    1 CONTINUE
  2 CONTINUE

C Converts numbers that should be zero to zero
  do 3 i=1,3
    do 4 j=1,3
      if (abs(rotomat(i,j)).lt. 1d-10) then
        rotomat(i,j)=0d0
      4 continue
  3 continue

c write (*,*) 'rotomat is', rotomat
c write (*,201) rotomat
c 201 format (D25.18)
C Define Transpose of Rotation matrix
rototrans(1,1)=rotomat(1,1)
rototrans(1,2)=rotomat(2,1)
rototrans(1,3)=rotomat(3,1)
rototrans(2,1)=rotomat(1,2)
rototrans(2,2)=rotomat(2,2)
rototrans(2,3)=rotomat(3,2)
rototrans(3,1)=rotomat(1,3)
rototrans(3,2)=rotomat(2,3)
rototrans(3,3)=rotomat(3,3)

C Rotate atoms in crystal so 2 axis are parallel with (111) surface.
C Store results in axis111.
C Also keep Crystal in original axis system.
c!
open (1,file='(111)rotate.xyz')
do 20 i=1,atoms
   do 30 j=1,3
      row(j)=crystal(i,j)
   30  continue!
c!
write (*,*) row
!
call dmvcalc(3,3,rotomat,row,rotoatom)
c!
write (*,*) rotoatom
!
do 40 j=1,3
   axis111(i,j)=rotoatom(j)
   if (abs(axis111(i,j)) .lt. 1d-8) axis111(i,j)=0
40  continue!
c!
write (1,*) axis111(i,1), axis111(i,2), axis111(i,3)
20  continue!
c!
write (1,*) axis111(i,1), axis111(i,2), axis111(i,3)
c!
close(1)

C go through the atoms in axis111
C Find the unique Z values of the atoms and place them into zvalues.
C Assign the number of unique zvalues as layers
C Order zvalues from greatest to least
C Assign atoms to an index matrix where the row corresponds to the Zlayers in Zvalues
C Assign the number of atoms in each row of the index matrix to the corresponding row in indexatoms
j=1
C stores first atom z value in zvalues
   zvalues(j)=axis111(1,3)
   j=j+1
   do 50 i=2,atoms
      do 51 k=1,j-1
if (real(axis111(i,3)) .eq. real(zvalues(k)))
goto 50
C Breaks do loop and skips Z value assignment if z crystal value is equal to zvalue.
51    continue
    zvalues(j)=axis111(i,3)
    j=j+1
50    continue
C Zeroing indexatoms
indexatoms(i)=0
    do 57  j=i-1,1,-1
        zvalues(j+1)=zvalues(j)
        if (currentz .lt. zvalues(j)) then
            zvalues(j+1)=currentz
            goto 56
        elseif (j .eq. 1) then
            zvalues(j)=currentz
        endif
    enddo
57    continue
56    continue
C Organized zvalues from greatest to least
    do 58  i=1,atoms
        do 59  j=1,layers
            if (real(axis111(i,3)) .eq. real(zvalues(j))) then
                indexatoms(j)=indexatoms(j)+1
                index(j,indexatoms(j))=i
            endif
        enddo
    enddo
59    continue
58    continue
C Create bottom and top surface using axis111 matrix
k=1
l=1
c    open (1,file='bottomsurface.xyz')
OPEN (2, FILE='topsurface.xyz')
DO 25 I=1, ATOMS
   IF (AXIS111(I,3) .EQ. 0) THEN
      DO 35 J=1,3
         BOTTOMSURF(K,J)=AXIS111(I,J)
      CONTINUE
   ENDIF
   WRITE (1,*) BOTTOMSURF(K,1), BOTTOMSURF(K,2), BOTTOMSURF(K,3)
   K=K+1
35 CONTINUE
   IF (AXIS111(I,3) .GT. 0 .AND. AXIS111(I,3) .LE. 3) THEN
      DO 45 J=1,3
         TOPSURF(L,J)=AXIS111(I,J)
      CONTINUE
      WRITE (2,*) TOPSURF(L,1), TOPSURF(L,2), TOPSURF(L,3)
      L=L+1
45 CONTINUE
25 CONTINUE
C 202 FORMAT (D23.16, D25.18, D25.18)
   BOTSINDEX=K-1
   TOPSINDEX=L-1
   WRITE (*,*) 'BOTTOM SURFACE HAS THIS NUMBER OF ATOMS', BOTSINDEX
   WRITE (*,*) 'TOP SURFACE HAS THIS NUMBER OF ATOMS', TOPSINDEX
   CLOSE(1)
   CLOSE(2)
C FIND MAXIMUM AND MINIMUM Y VALUES AS WELL AS THEIR POSITION OF THE BOTTOMSURFACE
   YM统治=1
   YMmaxloc=1
   MAXY=BOTTOMSURF(1,2)
   NEXTMINY=BOTTOMSURF(1,2)
   MINY=BOTTOMSURF(1,2)
   DO 55 I=2, BOTSINDEX
      IF (BOTTOMSURF(I,2) .LT. 1.01*MINY) THEN
         IF (MINY .LT. 1.01*NEXTMINY .AND. NEXTMINY .LT. 1.01*NEXTNEXTMINY) THEN
            NEXTNEXTMINY=NEXTMINY
            NEXTMINY=MINY
         ELSEIF (MINY .LT. NEXTMINY) THEN
            NEXTMINY=MINY
         ENDIF
         MINY=BOTTOMSURF(I,2)
         YM统治LOC=1
      ELSEIF (BOTTOMSURF(I,2) .LT. 1.01*NEXTMINY .AND. BOTTOMSURF(I,2) .GT. .99*MINY) THEN
         ...
if (nextminy .lt. 1.01*nextnextminy)
  nextnextminy=nextminy
  nextminy=bottomsurf(i,2)
else if (bottomsurf(i,2) .lt. 1.01*nextnextminy .and. bottomsurf(i,2) .gt. .99*nextminy)
  nextnextminy=bottomsurf(i,2)
else if (bottomsurf(i,2) .gt. maxy)
  maxy=bottomsurf(i,2)
  ymaxloc=i
endif
continue
write (*,201) nextnextminy
write (*,201) nextminy
write (*,201) miny
write (*,*) 'max y position is', ymaxloc
write (*,*) 'max y is', maxy
write (*,*) 'min y position is', yminloc
write (*,*) 'min y is', miny
write (*,*) 'nextminy is', nextminy
write (*,*) 'nextnextminy is', nextnextminy

C set min and max y as the periodic boundary conditions
  ymin=miny
  ymax=miny+(nextnextminy-miny)*3
  ymax=maxy+(nextminy-miny)

C calculate surface area of bottom surface (shape=parrallelogram)
area=base*height
  height=maxy-miny
  height=ymax-ymin
write (*,*) 'height is', height

C find x positions of the corners of the ymin side.
  minx=0
  nextminx=0
  maxx=0
  do 65 i=1,botsindex
    if (bottomsurf(i,2) .ge. (1.01*miny) .and. bottomsurf(i,2) .le. (.99*miny)) then
      write (*,*) 'location of miny is', i
      if (minx .lt. nextminx) then
        if (minx .lt. nextminx) nextminx=minx
        minx=bottomsurf(i,1)
      elseif (bottomsurf(i,1) .lt. nextminx .and.
        bottomsurf(i,1) .gt. minx) then
        nextminx=bottomsurf(i,1)
      elseif (bottomsurf(i,1) .gt. maxx) then
        maxx=bottomsurf(i,1)
    endif
  endif
C set min and max x as the periodic boundary condition
x min = min x
x max = min x + 6 * (next min x - min x)
C calculate the base and than calculate the area of the bottom surface
base = max x - min x
area = base * height - (next min x - min x) * (next min y - min y)
C convert area from angstrom^2 to meter^2
area = 1d-20 * area
C Boundary Conditions of axis111
max(1) = 1
max(2) = 1
max(3) = 0
min(1) = 1
min(2) = 1
C Use Surface atoms to calculate the energy of the surface of the bottombox and top box
C inside the material
satoms = 4 * botsindex
energy = satoms * ecoh
C Sets which boundary conditions are periodic a 1 is on a 0 is off
C Is in order of x,y,z boundary condition
C convert from ev to mj
   energy=energy*1.6021773d-16
c write (*,*) 'the energy in mJ is', energy

C calculate ground surface energy
   surfen=energy/area
c write (*,*) 'the surface energy is', surfen
   surfen0=surfen
   gammaenergy=surfen-surfen0
c open (1,file='gammasurface.txt')
c write (1,*) gammaenergy
c write (*,*) 'the gammaenergy is', gammaenergy

C setup previous gamma energy=gamma energy
   prevgamma=gammaenergy

C for testing only
C goto 161

C go through the rotated atoms
C assign atoms (row of axis111) that have z<=0 into index1
    (corresponds bottombox) [dont need anymore]
C assign atoms (row of axis111) that have z>0 into index2
    (corresponds to topbox)
C assign atoms (row of axis111) that have z>=-3 and z<=5 into
    sindex (corresponds to surfaces)
k = 1
   do 60 i=1,atoms
      if (axis111(i,3).gt. 0) then
         index2(k)=i
         k=k+1
      endif
60     continue
   atoms2=k-1

   open (3,file='surfaceatoms.xyz')
   l = 1
   do 53 i=3,6
      do 52 j=1,indexatoms(i)
         do 261 n=1,3
            surface(l,n)=axis111(index(i,j),n)
            sindex(l)=index(i,j)
261       continue
   c write (3,*) surface(l,1), surface(l,2),
   c surface(l,3)
      l=l+1
52     continue
53     continue
   c close (3)
satoms=l-1
C write (*,*) '# surface atoms are', satoms
C write (*,*) sindex

C assign atoms in axis111 to top box keeping the axis111 frame of reference
   do 240 j=1,atoms2
      do 245 k=1,3
       box2_1(j,k)=axis111(index2(j),k)
       245    continue
   240 continue

C convert top box from axis111 frame of reference to original crystal frame of reference
   do 250 k=1,atoms2
      do 255 j=1,3
       row(j)=box2_1(k,j)
       255    continue
   250    continue
C write (*,*) row
   call dmvcalc(3,3,rototrans,row,rotoatom)
C write (*,*) rotoatom
   do 260 j=1,3
       box2(k,j)=rotoatom(j)
       if (abs(box2(k,j)) .lt. 1e-14) box2(k,j)=0
       260    continue
   260 continue

C begin finding the stable stacking fault energy by moving topbox in step*(-2, 1, 1) direction
   step=at/6d0
C this shifts topbox atoms a total of 1/6*at<-2 1 1> direction in the regular crystal system
C write (*,*)
C write (*,*) 'beginining calculation of stable stacking fault'
C write (70,* )'beginining calculation of stable stacking fault'
   do 105 j=1,atoms2
      box2(j,1)=box2(j,1)+step*-2d0
      box2(j,2)=box2(j,2)+step*1d0
      box2(j,3)=box2(j,3)+step*1d0
   105    continue

C Convert topbox to the axis111 frame of reference.
   do 165 k=1,atoms2
      do 170 j=1,3
       row(j)=box2(k,j)
      170    continue
c       write (*,*) row
        call dmvcalc(3,3,rotomat,row,rotoatom)
c       write (*,*) rotoatom
        do 175  j=1,3
               box2_1(k,j)=rotoatom(j)
               if (abs(box2_1(k,j)) .lt. 1e-14) box2_1(k,j)=0
        175              continue
        165          continue

C Assign atoms from topbox to their corresponding atoms in axis111.
C Keep the axis 111 frame of reference
        do 12  j=1,atoms2
               do 13  k=1,3
                      axis111(index2(j),k)=box2_1(j,k)
               13        continue
        12          continue

C assign atoms from axis111 to surface
        open (3,file='surfacemodatoms.xyz')
        do 11  j=1,satoms
               do 14  k=1,3
                      surface(j,k)=axis111(sindex(j),k)
               14             continue
        11          continue
        close (3)

c        open (70,file='stackingfault.txt')
c Calculate Original Energy of the Surface in the axis111 system
        call periodicbcenergy(axis111,atoms,surface,satoms,energy)
c        write (*,*) 'the energy of the surface before optimization is', energy
c        write (70,*) 'the energy of the surface before optimization is', energy
c        write (*,*)

C for testing only
        goto 161

C minimize energy by moving layers of atoms in (0,0,1) direction
C this direction is according to axis111 frame of reference. In terms of the crystal framework
C this direction would be (1,1,1) direction.
        alpha=.002
C sets alpha which is used in the update formula of the steepest decent method
        h=1d-1
C sets h which is used in creating the other set of constants

C number of constants to minimize
m = layers

C Assign const1 (Assign Positive numbers for the top box and negative numbers for the bottom box)
c
const1(1)=0d0
do 16 j=1,4
    const1(j)=.033d0
16    continue
do 17 j=5,m
    const1(j)=-.033d0
17    continue
c
const1(m)=0d0

C Assign const2
    do 110 j=1,m
        const2(j)=const1(j)*(1+h)
110    continue

C This loop iteratively searches for the constant values which minimize the sum of errors squared
    do 115 j=1,16
    C Calculates the gradient and than the length of the gradient
        write (*,*) 'step', j, ' of optimization'
        write (70,*) 'step', j, ' of optimization'
        call gradsolve(grad,const1,const2,m,axis111,sindex,atoms,energy)
        magnitude=vecmagnitude(grad,m)
        write (70,*) 'the original crystal energy is', energy
        write (70,*) 'after step', j, ' length of gradient is', magnitude
        write (70,*)
        write (*,*) 'after step', j, ' length of gradient is', magnitude
        write (*,*)
    C New parameters are calculated using the steepest descent method
        do 125 k=1,m
            dconst(k)=-1*grad(k)
            const1(k)=const1(k)+dconst(k)*alpha
        125    continue
        do 120 k=1,m
    C Calculate other set of parameters (const2) used for calculating the gradient
        const2(k) = const1(k) *(1d0+h)
C The next step is to see if a local minimum has been found for the parameters
C if the length of the gradient is less than .05 than terminate minimization

\[ \text{if (magnitude .lt. 1.75d0) goto 130} \]

C end of minimization

C the final parameters and the final length of the gradient
C write (*,*) 'optimization finished'
C write (*,*) 'the final length of gradient is', magnitude
C write (70,*) 'optimization finished'
C write (70,*) 'the final length of gradient is', magnitude
C write (*,*) 'the final const1 is', const1

C create final crystal using final optimised parameters
C the created Crystal, and surface are in the axis111 frame of reference

\[ \text{call crystalcreation(const1,m,axis111,atoms)} \]

C create new surface in the axis111 frame of reference
\[ \text{do 155 j=1,satoms} \]
\[ \text{do 160 k=1,3} \]
\[ \text{surface(j,k)=axis111(sindex(j),k)} \]

C calculate energy of the surface inside the crystal.
C call periodicbcenergy(axis111,atoms,surface,satoms,energy)
C write (*,*) 'the final energy is', energy
C write (70,*) 'the final energy is', energy
\[ \text{energy=energy*1.6021773d-16} \]
C write (*,*) 'the energy in mj is', energy

C calculate gamma energy
\[ \text{surfен=energy/area} \]
C write (*,*) 'the surface energy is', surfен
C write (70,*) 'the surface energy is', surfен
\[ \text{gammaenergy=surfен-surfен0} \]
C write (1,*) gammaenergy
C write (*,*) 'the gamma energy is', gammaenergy
C write (70,*) 'the gamma energy is', gammaenergy
C write (70,*)
c 161  close (1)
c  close (70)
stable=anint(gammaenergy*10)/10
c write (*,*) 'stable stacking fault is', stable
end

H.31 Building a (110) surface (struct110.f)

subroutine struct110(latticeconstant)
implicit none
c Progam variables
double precision latticeconstant
c Local Variables
double precision perfect(800,3),fpart
integer i,j,k,l, index
c this code makes a perfect fcc crystal which is a 5x5x5 super
 cell. This super cell consists of 500 atoms.
C the output file is called fcc.xyz. The output structures first
line contains the number of atoms in the structure file. The rest
of the lines are atomic positions in x,y,z order.
l=1

C this do loop creates a structure with an atom at the corner
(i,j,k)*latticeconstant with one atom .5 lattice constant in x dir,
one atom .5 lattice constant in y dir and one atom .5 lattice
constant in z dir.
c open (12,file='exam.txt')
do 10 index=-20,20,5
   do 20 i=-20,25,5
      j=-i+index
      do 30 k=-20,20,10
         perfect(l,1)=i/10d0
         perfect(l,2)=j/10d0
         perfect(l,3)=k/10d0+fpart(dble(index/10d0))
         write (12,*) perfect(l,1), perfect(l,2), perfect(l,3)
      l=1+l
   30 continue
20 continue
10 continue

C this do loop moves the atoms over to the left and back and down
by 3 lattice units and writes the information into a file
C fcc.xyz
open (1,file='(110).xyz')
do 40 l=1,450
   if (abs(perfect(l,1)) .lt. 1e-14) perfect(l,1)=0
   if (abs(perfect(l,2)) .lt. 1e-14) perfect(l,2)=0
        continue
40 continue
if (abs(perfect(l, 3)) .lt. 1e-14) perfect(l, 3)=0
perfect(l, 1)=perfect(l, 1)*latticeconstant
perfect(l, 2)=perfect(l, 2)*latticeconstant
perfect(l, 3)=perfect(l, 3)*latticeconstant
write (1,*) perfect(l, 1), perfect(l, 2), perfect(l, 3)
40 continue
close (1)
return
der

H.32 Building a (111) surface (struct111.f)

subroutine struct111(latticeconstant)

implicit none

! Program variables
! double precision latticeconstant

! Local Variables
! double precision perfect(800,3)
integer i,j,k,l

! this code makes a perfect fcc crystal which is a 5x5x5 super
! cell. This super cell consists of 500 atoms.
C the output file is called fcc.xyz. The output structures first
line contains the number of atoms in the structure file. The rest
of the lines are atomic positions in x,y,z order.

l=1
! this do loop creates a structure with an atom at the corner
(i,j,k)*latticeconstant with one atom .5 latticeconstant in x dir,
one atom .5 lattice constant in y dir and one atom .5 lattice
constant in z dir.
c! open (12,file='exam.txt')
do 10 i=-1,1
do 20 j=-1,1
do 30 k=-3,4
perfect(l, 1)=i
perfect(l, 2)=j
perfect(l, 3)=k-perfect(l, 1)-perfect(l, 2)
c
perfect(l,3)
! l=1+l
perfect(l, 1)=i+.5
perfect(l, 2)=j+.5
perfect(l, 3)=k-perfect(l, 1)-perfect(l, 2)
c
perfect(l,3)
! l=l+1
perfect(l, 1)=i+.5
perfect(l, 2)=j
perfect(l, 3) = k - perfect(l, 1) - perfect(l, 2)
write (12, *) perfect(l, 1), perfect(l, 2),
perfect(l, 3)
l = l + 1
perfect(l, 1) = i
perfect(l, 2) = j + 0.5
perfect(l, 3) = k - perfect(l, 1) - perfect(l, 2)
write (12, *) perfect(l, 1), perfect(l, 2),
perfect(l, 3)
l = l + 1
30 continue
20 continue
10 continue
write (*, *) 'l is', l - 1
C this do loop moves the atoms over to the left and back and down
by 3 lattice units and writes the information into a file
C fcc.xyz
open (1, file='(111).xyz')
do 40 l = 1, 288
if (abs(perfect(l, 1)) .lt. 1e-14) perfect(l, 1) = 0
if (abs(perfect(l, 2)) .lt. 1e-14) perfect(l, 2) = 0
if (abs(perfect(l, 3)) .lt. 1e-14) perfect(l, 3) = 0
perfect(l, 1) = perfect(l, 1) * latticeconstant
perfect(l, 2) = perfect(l, 2) * latticeconstant
perfect(l, 3) = perfect(l, 3) * latticeconstant
write (1, *) perfect(l, 1), perfect(l, 2), perfect(l, 3)
40 continue
close (1)
return
end

H.33 Calculating the (100) Surface Energy (surface100.f)

subroutine surface100(surf100)
implicit none
C global variables
double precision a0, rcut
common/pot_fitting/a0, rcut
double precision mass, at
common/mdprop/mass, at
double precision ecoh, bulk, omega
common/matprop/ecoh, bulk, omega
double precision xmin, xmax, zmin, zmax, ymax, ymin
common/boundarycondition/xmin, xmax, zmin, zmax, ymax, ymin
integer max(3), min(3)
common/pbcon/max, min
integer satoms, indexatoms(15), layers
common/specialatomcount/satoms, indexatoms, layers
double precision index(15,150)
common/indexing/index
C program variables
double precision surf100
C local variables
integer atoms
double precision crystal(500,3), surface(500, 3),
sindex(500)
double precision bottomsurf(250,3), topsurf(250,3)
double precision gammaenergy, surfen0, surfen, energy
double precision maxx, minx, maxy, miny, nextminy, nextminx,
nextnextminy, maxz, minz
double precision height, base, area
integer ymaxloc, yminloc, botsindex, topsindex
double precision grad(10), const1(10), const2(10), dconst(10)
double precision alpha, h, magnitude, vectmagnitude
integer i, j, k, l, m, n

c a0=3.6150000000000d0
c rcut=5.50679d0
atoms=500
c e_coh=-3.54d0
c write (*,201) a0
c call fccstruct(a0)
C read in atom structure at ao and convert to at and store in
crystal
open (1, file='fcc.xyz')
do 10 i=1,atoms
   read (1,*) crystal(i,1), crystal(i,2), crystal(i,3)
do 11 j=1,3
   crystal(i,j)=crystal(i,j)/a0*at
11   continue
10 continue
   close (1)
c write (*,201) crystal
C go through the atoms in crystal
C Find the unique Z values of the atoms and place them into
zvalues.
C Assign the number of unique zvalues as layers
C Order zvalues from greatest to least
C Assign atoms to an index matrix where the row corresponds to
the Zlayers in Zvalues
C Assign the number of atoms in each row of the index matrix to
the corresponding row in indexatoms
j=1
C stores first atom z value in zvalues
zvalues(j)=crystal(1,3)
j=j+1
do 50 i=2,atoms
    do 51 k=1,j-1
        if (crystal(i,3) .eq. zvalues(k)) goto 50
    enddo
    continue
zvalues(j)=crystal(i,3)
j=j+1
50 continue
!
!
c write (*,*), 'zvalues is', zvalues
layers=j-1
c write (*,*), 'number of z layers is', layers
!
!
do 56 i=1,layers
    currentz=zvalues(i)
    indexatoms(i)=0
    do 57 j=i-1,1,-1
        zvalues(j+1)=zvalues(j)
        if (currentz .lt. zvalues(j)) then
            zvalues(j+1)=currentz
            goto 56
        elseif (j .eq. 1) then
            zvalues(j)=currentz
            goto 56
        endif
    enddo
57 continue
56 continue
!
c write (*,*), 'organized zvalues from greatest to least', zvalues
!
!
do 58 i=1,atoms
    do 59 j=1,layers
        if (crystal(i,3) .eq. zvalues(j)) then
            indexatoms(j)=indexatoms(j)+1
            index(j,indexatoms(j))=i
        endif
    enddo
59 continue
58 continue
!
c write (*,*), 'indexatoms is', indexatoms
!
c write (*,*), 'index is', index
!
!
C Create bottom Surface using crystal matrix
C Bottom Surface corresponds with Z value=0
C this surface is used to calculate the surface area.
k=1
open (1, file='bottomsurface.xyz')
open (2, file='topsurface.xyz')
do 25 i=1, atoms
   if (crystal(i, 3) .eq. 0) then
      do 35 j=1, 3
         bottomsurf(k, j) = crystal(i, j)
      continue
   end if
35 continue
write (1, *) bottomsurf(k, 1), bottomsurf(k, 2), bottomsurf(k, 3)
k = k + 1
25 continue
botsindex = k - 1
write (*, *) 'bottom surface has this number of atoms', botsindex
close (1)
close (2)

C Find maximum and minimum y values as well as their position of the bottom surface
C apparently there's a slight bug in the next do loop so
nextnextminy = nextminy
  yminloc = 1
  ymaxloc = 1
  maxy = bottomsurf(1, 2)
  nextminy = bottomsurf(1, 2)
  miny = bottomsurf(1, 2)
do 55 i = 2, botsindex
   if (bottomsurf(i, 2) .lt. 1.01 * miny) then
      if (miny .lt. 1.01 * nextminy .and. nextminy .lt. 1.01 * nextnextminy) then
         nextnextminy = nextminy
         nextminy = miny
      elseif (miny .lt. nextminy) then
         nextminy = miny
      endif
      miny = bottomsurf(i, 2)
      yminloc = i
   elseif (bottomsurf(i, 2) .lt. 1.01 * nextminy .and. bottomsurf(i, 2) .gt. .99 * miny) then
      if (nextminy .lt. 1.01 * nextnextminy) then
         nextnextminy = nextminy
         nextminy = bottomsurf(i, 2)
      elseif (bottomsurf(i, 2) .lt. 1.01 * nextnextminy .and. bottomsurf(i, 2) .gt. .99 * nextminy) then
         nextnextminy = bottomsurf(i, 2)
      elseif (bottomsurf(i, 2) .gt. maxy) then
         maxy = bottomsurf(i, 2)
      endif
   endif
55 continue
C set min and max y as the periodic boundary conditions
ymin=ymin
ymax=ymin+5*a0 alternative formula
ymax=maxy+(nextnextminy-miny)

C calculate surface area of bottom surface (shape=parallelogram)
C area=base*height-correction factor
height=maxy-miny+a0*sqrt(2d0)/4d0
height=ymax-ymin
write (*,*) 'height is', height

C find maximum and minimum x positions of the corners of the ymin side.
minx=0
nextminx=0
maxx=0
do 65 i=1,botsindex
bottomsurf(i,2) ge (1.01*miny) .and. (bottomsurf(i,2) le (.99*miny)) then
write (*,*) 'location of miny is', i
if (bottomsurf(i,1) lt. minx) then
if (minx lt. nextminx) nextminx=minx
minx=bottomsurf(i,1)
elseif (bottomsurf(i,1) lt. nextminx .and. bottomsurf(i,1) .gt. minx) then
nextminx=bottomsurf(i,1)
elseif (bottomsurf(i,1) .gt. maxx) then
maxx=bottomsurf(i,1)
endif
endif
65 continue
write (*,*) 'minx is', minx
write (*,*) 'nextminx is', nextminx
write (*,*) 'maxx is', maxx

C set min and max x as the periodic boundary condition
xmin=minx
c xmax=minx+5*a0 alternative way to calculate this
x_{\text{max}}=\text{max}x+(\text{next} \text{min}x-\text{min}x)

C calculate the base and than calculate the area of the bottom surface
c base=\text{max}x-\text{min}x+a0*\sqrt{2d0}/4d0
base=x_{\text{max}}-\text{xmin}
c area=base*height-(\text{next} \text{min}x-\text{min}x)\times(\text{next} \text{next} \text{min}y-\text{min}y)*2d0
area=base*height
c write (*,*) 'base is', base
c write (*,*) 'area is', area

C convert area from angstrom^2 to meter^2
area=1d-20*area
c write (*,*) 'total area in m^2', area

C Boundary Conditions of crystal box
c write (*,*) 'x max boundary condition is', xmax
c write (*,*) 'x min boundary condition is', xmin
c write (*,*) 'y max boundary condition is', ymax
c write (*,*) 'y min boundary condition is', ymin
c write (*,201) ymax
\text{c write (*,201) ymin}

C Sets which boundary conditions are periodic a 1 is on a 0 is off
C Is in order of x,y,z boundary condition
    \text{max}(1)=1
    \text{max}(2)=1
    \text{max}(3)=0
    \text{min}(1)=1
    \text{min}(2)=1
    \text{min}(3)=0

C Use Surface atoms to calculate energy of the surface inside the material
satoms=atoms
c write (*,*) '# surface atoms are', satoms
c \text{energy}=satoms*\text{ecoh}
c write(*,*) 'the energy is', energy

C convert from ev to mj
\text{energy}=\text{energy} \times 1.6021773d-16
c write (*,*) 'the energy in mJ is', energy

C calculate ground surface energy
surfens = energy / area / 2d0

write (*,*) 'the surface energy is', surfen
surfen0 = surfen
gammaenergy = surfen - surfen0

open (1, file = 'gammasurface.txt')
write (1, *) gammaenergy
write (*,*) 'the gammaenergy is', gammaenergy

C for testing purposes
go to 161

C go through the crystal atoms
C assign atoms (row of crystal) into surfaceatoms [surface]
C Assign the atoms crystal rows, which are placed into surfaceatoms, into the sindex vector
C Find the number of atoms in the surfaceatoms structure [satoms]
C i corresponds to the layer #, indexatoms(i) is the # atoms in the layer
l = 1
do 53 i = 1, layers
   do 52 j = 1, indexatoms(i)
      do 260 n = 1, 3
         surface(l, n) = crystal(index(i, j), n)
      sindex(l) = index(i, j)
   260 continue
write (3, *) surface(l, 1), surface(l, 2), surface(l, 3)
l = l + 1
52 continue
53 continue
satoms = l - 1
write (*,*) '# surface atoms are', satoms

call periodicbcenergy(crystal, atoms, surface, satoms, energy)
write (*,*) 'the original energy before optimization is', energy
write (*,*) 'the sindex is', sindex

C minimize energy by moving layers in (0,0,1) direction of the crystal system
open (70, file = 'stackingfault.txt')
alpha = .002
C sets alpha which is used in the update formula of the steepest decent method
h = 1d-1
C sets h which is used in creating the other set of constants
C number of constants to minimize
m = layers

C assign const1 (Assign negative numbers to the first 4 top layers and than 0 everywhere else)
    do 6 j = 1, 5
        const1(j) = -.05d0
    6 continue
    do 7 j = 6, layers
        const1(j) = .05d0
    7 continue

C Assign const2
    do 195 j = 1, m
        const2(j) = const1(j) * (1 + h)
    195 continue

C This loop iteratively searches for the constant values which minimize the sum of errors squared
    do 200 j = 1, 30

C Calculates the gradient and than the length of the gradient
    write (*, *) 'step', j, ' of optimization'
    write (70, *) 'step', j, ' of optimization'
    call gradsolve(grad, const1, const2, m, crystal, sindex, atoms, energy)
    magnitude = vectmagnitude(grad, m)
    write (70, *) 'the original crystal energy is', energy
    write (70, *) 'after step', j, ' length of gradient is', magnitude
    write (70, *)
    write (*, *) 'after step', j, ' length of gradient is', magnitude
    write (*, *)
    goto 161

C New parameters are calculated using the steepest descent method
    do 205 k = 1, m
        dconst(k) = -1 * grad(k)
        const1(k) = const1(k) + dconst(k) * alpha
    205 continue

C Calculate other set of parameters (const2) used for calculating the gradient
    const2(k) = const1(k) * (1d0 + h)
    210 continue

C The next step is to see if a local minimum has been found for the parameters
C if the length of the gradient is less than .05 than terminate minimization
    if (magnitude .lt. 1.6d0) goto 215
C end of minimization
200   continue
C the final parameters and the final length of the gradient
C write (*,*) 'optimization finished'
C write (*,*) 'the final length of gradient is', magnitude
C write (70,*) 'optimization finished'
C write (70,*) 'the final length of gradient is', magnitude
C write (*,*) 'the final const1 is', const1
C create final crystal using final optimised parameters
C the created Crystal, and surface are in the crystal frame of reference
215     call crystalcreation(const1,m,crystal,atoms)

C assign atoms from crystal to surface to calculate the optimised surface energy.
C open (3,file='surfaceatoms.xyz')
   do 220 i=1,satoms
       do 225 j=1,3
           surface(i,j)=crystal(sindex(i),j)
   225           continue
   220       continue
C close (3)

C Calculate Energy of the Surface in the axis111 system
    call periodicbcenergy(crystal,atoms,surface,satoms,energy)
C write(*,*) 'the energy is', energy

C convert from ev to mj
   energy=energy*1.6021773d-16
C write (*,*) 'the energy in mJ is', energy

C calculate ground surface energy
   surfen=energy/area/2d0
C write (*,*) 'the surface energy is', surfen
C surfen0=surfen
   gammaenergy=surfen-surfen0
C write (1,*) gammaenergy
C write (*,*) 'the gammaenergy is', gammaenergy
   surf100=anint(gammaenergy)
C write (*,*) 'the surface energy of the 100 plane is', surf100

C 161   close (1)
H.34 Calculating the (110) Surface Energy (surface110.f)

```fortran
subroutine surface110(surf110)
implicit none
C global variables
double precision a0,rcut
common/pot_fitting/a0,rcut
double precision ecoh, bulk, omega
common/matprop/ecoh,bulk,omega
double precision xmin, xmax, zmin, zmax, ymax, ymin
common/boundarycondition/xmin,xmax,zmin,zmax,ymax,ymin
integer max(3), min(3)
common/pbcon/max,min
integer satoms,indexatoms(15),layers
common/specialatomcount/satoms,indexatoms,layers
double precision index(15,150)
common/indexing/index
C program variables
double precision surf110
C local variables
integer atoms
double precision surface(450,3), crystal(450,3), axis111(450,3), sindex(450)
double precision row(3), rotoatom(3), rotomat(3,3), rototrans(3,3), rotomat2(3,3), roto
mat1(3,3)
double precision bottomsurf(150,3), topsurf(150,3)
double precision zvalues(15), currentz
double precision gammaenergy, surfen0, surfen, energy
double precision maxx,minx,maxy,miny,nextminy, nextminx, nextnextminy, maxz,minz
double precision height,base, area
integer ymaxloc,yminloc,botsindex, topsindex
double precision grad(10), const1(10), const2(10), dconst(10)
double precision alpha, h, magnitude, vectmagnitude
integer i, j, k, l, m, n
C even though this code is for 110 surface due to ease of coding,
c axis111 will not be converted to axis110. Make sure not to get
confused.

c a0=3.6150000000000d0
c rcut=5.50679d0
atoms=450
c e_coh=-3.54d0
c write (*,201) a0
```

397
c call struct110(a0)
C read in atom structure and store in crystal
  open (1,file='(110).xyz')
  do 10 i=1,atoms
    read (1,*) crystal(i,1), crystal(i,2), crystal(i,3)
 10 continue
  close (1)
c write (*,201) crystal

C rotation matrix1 for rotating by 0 degrees around the z axis
  rotomat1(1,1)=1d0
  rotomat1(1,2)=0d0
  rotomat1(1,3)=0d0
  rotomat1(2,1)=0d0
  rotomat1(2,2)=1d0
  rotomat1(2,3)=0d0
  rotomat1(3,1)=0d0
  rotomat1(3,2)=0d0
  rotomat1(3,3)=1d0
c write (*,*) 'rotomat1 is',rotomat1
c write (*,201) rotomat1
c WRITE (*,*)
C rotation matrix2 for converting axis system to (110) plane
  rotomat2(1,1)=1.0/sqrt(2d0)
  rotomat2(1,2)=1.0/sqrt(2d0)
  rotomat2(1,3)=0d0
  rotomat2(2,1)=0d0
  rotomat2(2,2)=0d0
  rotomat2(2,3)=1d0
  rotomat2(3,1)=1.0/sqrt(2d0)
  rotomat2(3,2)=1.0/sqrt(2d0)
  rotomat2(3,3)=0d0
c write (*,*) 'rotomat2 is',rotomat2
c write (*,201) rotomat2
c write (*,*)

C calculate rotation matrix (rotomat) which is rotomat1*rotomat2
  DO 2 i = 1, 3
    DO 2 j = 1, 3
      rotomat(i,j) = 0d0
    2     DO 1 k = 1, 3
      rotomat(i,j) = rotomat(i,j) + rotomat1(i,k) * 
                      rotomat2(k,j)
 1     CONTINUE
 2     CONTINUE

C Converts numbers that should be zero to zero
  do 3 i=1,3
do 4 j=1,3
   if (abs(rotomat(i,j)) .lt. 1d-10)
5   rotomat(i,j)=0d0
4      continue
3   continue

write (*,*) 'rotomat is', rotomat
write (*,201) rotomat
   format (D25.18)

C Define Transpose of Rotation matrix
rototrans(1,1)=rotomat(1,1)
rototrans(1,2)=rotomat(2,1)
rototrans(1,3)=rotomat(3,1)
rototrans(2,1)=rotomat(1,2)
rototrans(2,2)=rotomat(2,2)
rototrans(2,3)=rotomat(3,2)
rototrans(3,1)=rotomat(1,3)
rototrans(3,2)=rotomat(2,3)
rototrans(3,3)=rotomat(3,3)

C Rotate atoms in crystal so 2 axis are parallel with (110) surface.
C Store results in axis111.
C Also keep Crystal in original axis system.
c open (1,file='(110)rotate.xyz')
do 20 i=1,atoms
   do 30 j=1,3
      row(j)=crystal(i,j)
30      continue
write (*,*) row
   call dmvcalc(3,3,rotomat,row,rotoatom)
c write (*,*) rotoatom
do 40 j=1,3
   axis111(i,j)=rotoatom(j)
   if (abs(axis111(i,j)) .lt. 1d-8) axis111(i,j)=0
40      continue
write (1,*) axis111(i,1), axis111(i,2), axis111(i,3)
20      continue
close(1)

C go through the atoms in axis111
C Find the unique Z values of the atoms and place them into zvalues.
C Assign the number of unique zvalues as layers
C Order zvalues from greatest to least
C Assign atoms to an index matrix where the row corresponds to the Zlayers in Zvalues
C Assign the number of atoms in each row of the index matrix to the corresponding row in indexatoms
j = 1
C stores first atom z value in zvalues
zvalues(j) = axis111(1,3)
j = j + 1
do 50 i = 2, atoms
   do 51 k = 1, j - 1
      if (real(axis111(i, 3)) .eq. real(zvalues(k)))
         goto 50
   51 continue
   zvalues(j) = axis111(i, 3)
j = j + 1
50 continue
c write (*,*) 'zvalues is', zvalues
layers = j - 1
c write (*,*) 'number of z layers is', layers

do 56 i = 1, layers
   c write (*,*) 'zvalues is', zvalues, 'before step', i
   currentz = zvalues(i)
c zeroing indexatoms
   indexatoms(i) = 0
   do 57 j = i - 1, 1, -1
      zvalues(j + 1) = zvalues(j)
      if (currentz .lt. zvalues(j)) then
         zvalues(j + 1) = currentz
         goto 56
      elseif (j .eq. 1) then
         zvalues(j) = currentz
      endif
57 continue
56 continue
c write (*,*) 'organized zvalues from greatest to least', zvalues

do 58 i = 1, atoms
   do 59 j = 1, layers
      if (real(axis111(i, 3)) .eq. real(zvalues(j)))
         then
            indexatoms(j) = indexatoms(j) + 1
            index(j, indexatoms(j)) = i
58 continue
C Create bottom and top Surface using axis111 matrix
  k=1
  l=1
  open (1,file='bottomsurface.xyz')
  open (2,file='topsurface.xyz')
  do 25 i=1,atoms
    if (axis111(i,3).eq. 0) then
      do 35 j=1,3
        bottomsurf(k,j)=axis111(i,j)
      continue
      write (1,*) bottomsurf(k,1), bottomsurf(k,2), bottomsurf(k,3)
      k=k+1
    endif
    if (axis111(i,3).gt. 0 .and. axis111(i,3).le. 3) then
      do 45 j=1,3
        topsurf(l,j)=axis111(i,j)
      continue
      write (2,*) topsurf(l,1), topsurf(l,2), topsurf(l,3)
      l=l+1
    endif
  continue
  botsindex=k-1
  topsindex=l-1
  write (*,'bottom surface has this number of atoms',botsindex)
  write (*,'top surface has this number of atoms', topsindex
  close(1)
  close(2)

C Find maximum and minium y values as well as their position of the bottomsurface
C apparently theres a bug in this do loop so the nextnextminy is actually nextminy
  yminloc=1
  ymaxloc=1
  maxy=bottomsurf(1,2)
  nextminy=bottomsurf(1,2)
  miny=bottomsurf(1,2)
  do 55 i=2,botsindex
if (bottomsurf(i, 2) .lt. 1.01*miny) then
  if (miny .lt. 1.01*nextminy .and. nextminy .lt. 1.01*nextnextminy) then
    nextnextminy=nextminy
    nextminy=miny
  elseif (miny .lt. nextminy) then
    nextminy=miny
  endif
  miny=bottomsurf(i, 2)
  yminloc=i
elseif (bottomsurf(i, 2) .lt. 1.01*nextminy .and. bottomsurf(i, 2) .gt. .99*miny) then
  if (nextminy .lt. 1.01*nextnextminy) then
    nextnextminy=nextminy
    nextminy=bottomsurf(i, 2)
  elseif (bottomsurf(i, 2) .lt. 1.01*nextnextminy .and. bottomsurf(i, 2) .gt. .99*nextminy) then
    nextnextminy=bottomsurf(i, 2)
  elseif (bottomsurf(i, 2) .gt. maxy) then
    maxy=bottomsurf(i, 2)
    ymaxloc=i
  endif
  continue
continue
write (*,201) nextnextminy
write (*,201) nextminy
write (*,201) miny
write (*,*) 'max y position is', ymaxloc
write (*,*) 'max y is', maxy
write (*,*) 'min y position is', yminloc
write (*,*) 'min y is', miny
write (*,*) 'nextminy is', nextminy
write (*,*) 'nextnextminy is', nextnextminy

C set min and max y as the periodic boundary conditions
  ymin=miny
  ymax=(nextnextminy-miny)*5d0+miny alternative way of calculating ymax
  ymax=maxy+(nextnextminy-miny)

C calculate surface area of bottom surface (shape=parrallelogram)
C area=base*height
  height=maxy-miny+a0*sqrt(2d0)/4d0
  height=ymax-ymin
write (*,*) 'height is', height

C find x positions of the corners of the ymin side.
  minx=0
  nextminx=0
maxx=0
do 65 i=1,botsindex
  if (bottomsurf(i,2) .ge. (1.01*miny) .and.
  bottomsurf(i,2) .le. (.99*miny)) then
    write (*,*) 'location of miny is', i
    if (bottomsurf(i,1) .lt. minx) then
      if (minx .lt. nextminx) nextminx=minx
      minx=bottomsurf(i,1)
    elseif (bottomsurf(i,1) .lt. nextminx .and.
    bottomsurf(i,1) .gt. minx) then
      nextminx=bottomsurf(i,1)
    endif
  endif
  endif
65 continue
write (*,*) 'minx is', minx
write (*,*) 'nextminx is', nextminx
write (*,*) 'maxx is', maxx
C set min and max x as the periodic boundary condition
xmin=minx
xmax=(nextminx-minx)*10d0+minx alternative way of calculating xmax
xmax=maxx+(nextminx-minx)
C calculate the base and than calculate the area of the bottom surface
base=maxx-minx+a0*sqrt(2d0)/4d0
base=xmax-xmin
carea=base*height-(nextnextminy-miny)*(nextminx-
minx)*2d0
area=base*height
write (*,*) 'base is', base
write (*,*) 'area is', area
C convert area from angstrom^2 to meter^2
area=1d-20*area
write (*,*) 'total area in m^2', area
C Boundary Conditions of axis111
write (*,*) 'x max boundary condition is', xmax
write (*,*) 'x min boundary condition is', xmin
write (*,*) 'y max boundary condition is', ymax
write (*,*) 'y min boundary condition is', ymin
write (*,201) ymax
write (*,201) ymin
c Sets which boundary conditions are periodic a 1 is on a 0 is off
C Is in order of x,y,z boundary condition
  max(1)=1
  max(2)=1
  max(3)=0
  min(1)=1
  min(2)=1
  min(3)=0

C Use Surface atoms to calculate energy of the surface inside the material
  satoms=atoms
  c  write (*,*), 'surface atoms are', satoms
  energy=satoms*ecoh
  c  write(*,*), 'the energy is', energy

C convert from ev to mj
  energy=energy*1.6021773d-16
  c  write (*,*), 'the energy in mJ is', energy

C calculate ground surface energy
  surfen=energy/area/2d0
  c  write (*,*), 'the surface energy is', surfen
  surfen0=surfen
  gammaenergy=surfen-surfen0
  c  open (1,file='gammasurface.txt')
  c  write (1,*), gammaenergy
  c  write (*,*), 'the gammaenergy is', gammaenergy

C for testing only
  c  goto 161

C go through the rotated atoms
C assign atoms (row of axis111) into surfaceatoms [surface]
C Assign the atoms crystal rows, which are placed into surfaceatoms, into the sindex vector
C Find the number of atoms in the surfaceatoms structure [satoms]
C i corresponds to the layer #, indexatoms(i) is the # atoms in the layer
  l=1
  do 53 i=1,layers
     do 52 j=1,indexatoms(i)
        do 260 n=1,3
           surface(l,n)=axis111(index(i,j),n)
           sindex(l)=index(i,j)
        260     continue
     c  write (3,*), surface(l,1), surface(l,2),
        surface(l,3)
l=l+1
52      continue
53      continue
        satoms=l-1
        write (*,*),'# surface atoms are', satoms
        call periodicbcenergy(axis111,atoms,surface,satoms,energy)
        write (*,*),'the original energy before optimization is', energy
        write (*,*),

        c minimize energy by moving layers in (0,0,1) direction of the crystal system
        c open (70,file='stackingfault.txt')
        alpha=.002
        C sets alpha which is used in the update formula of the steepest decent method
        h=1d-1
        C sets h which is used in creating the other set of constants
        C number of constants to minimize
        m=layers

        C assign const1 (Assign negative numbers to the first 4 top layers and than 0 everywhere else)
        do 6 j=1,4
           const1(j)=-.05d0
        6      continue
        do 7 j=5,layers
           const1(j)=.05d0
        7      continue

        C Assign const2
        do 195 j=1,m
           const2(j)=const1(j)*(1+h)
        195     continue

        C This loop iteratively searches for the constant values which minimize
        c the sum of errors squared
        do 200 j=1,30
        C Calculates the gradient and than the legnth of the gradient
        c write (*,*),'step', j, ' of optimization'
        c write (70,*),'step', j, ' of optimization'
        call
        gradsolve(grad,const1,const2,m,axis111,sindex,atoms,energy)
        magnitude=vecmagnitude(grad,m)
        c write (70,*),'the original crystal energy is', energy
write (70,*) 'after step', j, ' length of gradient is', magnitude
write (70,*)
write (*,*) 'after step', j, ' length of gradient is', magnitude
write (*,*)
goto 161

c C New parameters are calculated using the steepest descent method
do 205 k=1,m
dconst(k)=-1*grad(k)
const1(k)=const1(k)+dconst(k)*alpha
205 continue
do 210 k=1,m
C Calculate other set of parameters (const2) used for calculating the gradient
const2(k) = const1(k)* (1d0+h)
210 continue

c C The next step is to see if a local minimum has been found for the parameters
C if the length of the gradient is less than .05 than terminate minimization
if (magnitude .lt. 1.6d0) goto 215

c end of minimization
200 continue
C the final parameters and the final length of the gradient
c write (*,*) 'optimization finished'
c write (*,*) 'the final length of gradient is', magnitude
c write (70,*) 'optimization finished'
c write (70,*) 'the final length of gradient is', magnitude
c write (*,*) 'the final const1 is', const1

c C create final crystal using final optimised parameters
C the created Crystal, and surface are in the axis110 frame of reference
215 call crystalcreation(const1,m,axis111,atoms)
c C assign atoms from crystal to surface to calculate the optimized surface energy.
c open (3,file='surfaceatoms.xyz')
do 220 i=1,satoms
do 225 j=1,3
   surface(i,j)=axis111(sindex(i),j)
225 continue
c     write (3,*) surface(i,1), surface(i,2), surface(i,3)
220   continue

C convert from ev to mj
   energy=energy*1.6021773d-16

C calculate ground surface energy
   surfen=energy/area/2

C calculate gamma energy
   gammaenergy=surfen-surfen0
   surf110=anint(gammaenergy)

end

H.35 Calculating the (111) Surface Energy (surface111.f)

   subroutine surface111(surf111)
   implicit none
   C global variables
   double precision a0,rcut
   common/pot_fitting/a0,rcut
   double precision ecoh, bulk, omega
   common/matprop/ecoh,bulk,omega
   double precision xmin, xmax, zmin, zmax, ymax, ymin
   common/boundarycondition/xmin,xmax,zmin,zmax,ymax,ymin
   integer max(3), min(3)
   common/pbcon/max,min
   integer satoms,indexatoms(15),layers
   common/specialatomcount/satoms,indexatoms,layers
   double precision index(15,150)
   common/indexing/index
   C program variables
   double precision surf111
   C local variables
   integer atoms
   double precision surface(288,3), crystal(288,3),
   axis111(288,3), sindex(288)
double precision row(3), rotoatom(3), rototrans(3,3), rotoatom2(3,3), rotoatom1(3,3)

double precision bottomsurf(150,3), topsurf(150,3)

double precision zvalues(15), currentz

double precision gammaenergy, surfen0, surfen, energy

double precision maxx, minx, maxy, miny, nextminy, nextminx,
nextnextminy, maxz, minz

double precision height, base, area

ingeger ymaxloc, yminloc, botsindex, topsindex

double precision grad(10), const1(10), const2(10), dconst(10)

double precision alpha, h, magnitude, vectmagnitude

integer i, j, k, l, m, n

c a0=3.6150000000000d0

c rcut=5.50679d0

atoms=288

c e_coh=-3.54d0

c write (*,201) a0

c call struct111(a0)

C read in atom structure and store in crystal

open (1, file='(111).xyz')
do 10 i=1,atoms
   read (1,*) crystal(i,1), crystal(i,2), crystal(i,3)
10 continue

close (1)
c write (*,201) crystal

C rotation matrix1 for rotating by 30 degrees around the z axis

rotoatom1(1,1)=sqrt(3d0)/2d0
rotoatom1(1,2)=-.5d0
rotoatom1(1,3)=0
rotoatom1(2,1)=.5d0
rotoatom1(2,2)=sqrt(3d0)/2d0
rotoatom1(2,3)=0d0
rotoatom1(3,1)=0d0
rotoatom1(3,2)=0d0
rotoatom1(3,3)=1d0

c write (*,*) 'rotatom1 is',_rotatom1

c write (*,201) rotoatom1

c WRITE (*,*)

C rotation matrix2 for converting axis system to (111) plane

rotoatom2(1,1)=-1/sqrt(6d0)
rotoatom2(1,2)=-1/sqrt(6d0)
rotoatom2(1,3)=2/sqrt(6d0)
rotoatom2(2,1)=1/sqrt(2d0)
rotoatom2(2,2)=-1/sqrt(2d0)
rotoatom2(2,3)=0d0
rotomat2(3,1)=1/sqrt(3d0)
rotomat2(3,2)=1/sqrt(3d0)
rotomat2(3,3)=1/sqrt(3d0)
c write (*,*) 'rotomat2 is', rotomat2
c write (*,201) rotomat2
c write (*,*)

C calculate rotation matrix (rotomat) which is rotomat1*rotomat2
DO 2 I = 1, 3
   DO 2 J = 1, 3
      rotomat(I,J) = 0D0
      DO 1 K = 1, 3
         rotomat(I,J) = rotomat(I,J) + rotomat1(I,K) *
            rotomat2(K,J)
   1 CONTINUE
2 CONTINUE

C Converts numbers that should be zero to zero
   do 3 i=1,3
      do 4 j=1,3
         if (abs(rotomat(i,j)) .lt. 1D-10) then
            rotomat(i,j)=0D0
         end if
   4 continue
3 continue

c write (*,*) 'rotomat is', rotomat
c write (*,201) rotomat

c 201 format (D25.18)

C Define Transpose of Rotation matrix
rototrans(1,1)=rotomat(1,1)
rototrans(1,2)=rotomat(2,1)
rototrans(1,3)=rotomat(3,1)
rototrans(2,1)=rotomat(1,2)
rototrans(2,2)=rotomat(2,2)
rototrans(2,3)=rotomat(3,2)
rototrans(3,1)=rotomat(1,3)
rototrans(3,2)=rotomat(2,3)
rototrans(3,3)=rotomat(3,3)

C Rotate atoms in crystal so 2 axis are parallel with (111) surface.
C Store results in axis111.
C Also keep Crystal in original axis system.
c open (1,file='(111)rotate.xyz')
do 20 i=1,atoms
   do 30 j=1,3
      row(j)=crystal(i,j)
30 continue
c write (*,*) row

  call dmvcalc(3,3,rotomat,row,rotoatom)
c write (*,*) rotoatom
do 40 j=1,3
     axis111(i,j)=rotoatom(j)
     if (abs(axis111(i,j)) .lt. 1d-8) axis111(i,j)=0
  40 continue
c write (1,*), axis111(i,1), axis111(i,2), axis111(i,3)
  20 continue
c close(1)

C go through the atoms in axis111
C Find the unique Z values of the atoms and place them into zvalues.
C Assign the number of unique zvalues as layers
C Order zvalues from greatest to least
C Assign atoms to an index matrix where the row corresponds to the Zlayers in Zvalues
C Assign the number of atoms in each row of the index matrix to the corresponding row in indexatoms
j=1
C stores first atom Z value in zvalues
  zvalues(j)=axis111(1,3)
  j=j+1
do 50 i=2,atoms
   do 51 k=1,j-1
      if (real(axis111(i,3)) .eq. real(zvalues(k)))
   goto 50
C Breaks do loop and skips Z value assignment if z crystal value is equal to zvalue.
   51 continue
   zvalues(j)=axis111(i,3)
   j=j+1
  50 continue
c write (*,*) 'zvalues is', zvalues, 'before step', i
C currentz = zvalues(i)
c write (*,*) 'number of z layers is', layers
  do 56 i=1,layers
  c write (*,*) 'zvalues is', zvalues, 'before step', i
C zeroing indexatoms
  indexatoms(i)=0
do 57 j=i-1,1,-1
     zvalues(j+1)=zvalues(j)
     if (currentz .lt. zvalues(j)) then
        zvalues(j+1)=currentz
  57 continue
goto 56
elseif (j .eq. 1) then
  zvalues(j)=currentz
C found right slot for current z to fit in so begin algorithm for next zvalue
  goto 56
endif
57 continue
56 continue
c  write (*,*), 'organized zvalues from greatest to least', zvalues
  do 58 i=1,atoms
    do 59 j=1,layers
      if (real(axis111(i,3)) .eq. real(zvalues(j)))
        indexatoms(j)=indexatoms(j)+1
        index(j,indexatoms(j))=i
      endif
    59 continue
  58 continue
c  write (*,*), 'indexatoms is', indexatoms
c  write (*,*), 'index is', index
C Create bottom and top Surface using axis111 matrix
  k=1
  l=1
c  open (1, file='bottomsurface.xyz')
c  open (2, file='topsurface.xyz')
  do 25 i=1,atoms
    if (axis111(i,3) .eq. 0) then
      do 35 j=1,3
        bottomsurf(k,j)=axis111(i,j)
      35 continue
      c  write (1,*), bottomsurf(k,1), bottomsurf(k,2), bottomsurf(k,3)
      k=k+1
    endif
    if (axis111(i,3) .gt. 0 .and. axis111(i,3) .le. 3) then
      do 45 j=1,3
        topsurf(l,j)=axis111(i,j)
      45 continue
      c  write (2,*), topsurf(l,1), topsurf(l,2), topsurf(l,3)
      l=l+1
    endif
  25 continue
c 202 format (D23.16, D25.18, D25.18)
botsindex = k - 1
topsindex = l - 1

c write (*, *) 'bottom surface has this number of atoms', botsindex
c write (*, *) 'top surface has this number of atoms', topsindex
c close(1)
c close(2)

C Find maximum and minimum y values as well as their position on the bottom surface
yminloc = 1
ymaxloc = 1
maxy = bottomsurf(1, 2)
nextminy = bottomsurf(1, 2)
miny = bottomsurf(1, 2)
do 55 i = 2, botsindex
   if (bottomsurf(i, 2) .lt. 1.01 * miny) then
      if (miny .lt. 1.01 * nextminy .and. nextminy .lt. 1.01 * nextnextminy) then
         nextnextminy = nextminy
         nextminy = miny
      elseif (miny .lt. nextminy) then
         nextminy = miny
      endif
      miny = bottomsurf(i, 2)
yminloc = i
   elseif (bottomsurf(i, 2) .lt. 1.01 * nextminy .and. bottomsurf(i, 2) .gt. 0.99 * miny) then
      if (nextminy .lt. 1.01 * nextnextminy) then
         nextnextminy = nextminy
         nextminy = bottomsurf(i, 2)
      elseif (bottomsurf(i, 2) .lt. 1.01 * nextnextminy .and. bottomsurf(i, 2) .gt. 0.99 * nextminy) then
         nextnextminy = bottomsurf(i, 2)
      elseif (bottomsurf(i, 2) .gt. maxy) then
         maxy = bottomsurf(i, 2)
ymaxloc = i
      endif
   endif
55 continue
c write (*, 201) nextnextminy
c write (*, 201) nextminy
c write (*, 201) miny
c write (*, *) 'max y position is', ymaxloc
c write (*, *) 'max y is', maxy
c write (*, *) 'min y position is', yminloc
c write (*, *) 'min y is', miny
c write (*, *) 'nextminy is', nextminy
c write (*, *) 'nextnextminy is', nextnextminy
C set min and max y as the periodic boundary conditions
  ymin=miny
C  ymax=miny+(nextnextminy-miny)*3 Another way of calculating
  the periodic boundary condition
  ymax=maxy+(nextminy-miny)

C calculate surface area of bottom surface (shape=parrallelogram)
C area=base*height-correction factor
C height uses the periodic boundary conditions because the energy
  is from the pbc area
  height=ymax-ymin
C write (*,*) 'height is', height
C find x positions of the corners of the ymin side.
  minx=0
  nextminx=0
  maxx=0
do 65 i=1,botsindex
  if (bottomsurf(i,2).ge.(1.01*miny).and.
      bottomsurf(i,2).le. (.99*miny)) then
    write (*,*) 'location of miny is', i
    if (bottomsurf(i,1).lt.minx) then
      if (minx.lt.nextminx) nextminx=minx
      minx=bottomsurf(i,1)
    elseif (bottomsurf(i,1).lt.nextminx .and.
      bottomsurf(i,1).gt. maxx) then
      maxx=bottomsurf(i,1)
    endif
  endif
  if (bottomsurf(i,2).ge.(1.01*miny).and.
      bottomsurf(i,2).le. (.99*miny)) then
    write (*,*) 'location of miny is', i
    if (bottomsurf(i,1).lt.minx) then
      if (minx.lt.nextminx) nextminx=minx
      minx=bottomsurf(i,1)
    elseif (bottomsurf(i,1).lt.nextminx .and.
      bottomsurf(i,1).gt. maxx) then
      maxx=bottomsurf(i,1)
    endif
  endif
65 continue
C write (*,*) 'minx is', minx
C write (*,*) 'nextminx is', nextminx
C write (*,*) 'maxx is', maxx

C set min and max x as the periodic boundary condition
  xmin=minx
C  xmax=minx+6*(nextminx-minx) Another way of calculating the
  periodic boundary condition
  xmax=maxx+(nextminy-miny)

C calculate the base and than calculate the area of the bottom
  surface
C Base uses the periodic boundary conditions because the energy
  is from the pbc area
  base=xmax-xmin
C area=base*height-(nextminx-minx)*(nextminy-miny)
  area=base*height
c write (*,*) 'base is', base

c write (*,*) 'area is', area

C convert area from angstrom^2 to meter^2
area=1d-20*area
c write (*,*) 'total area in m^2', area

C Boundary Conditions of axis111

c write (*,*) 'x max boundary condition is', xmax
c write (*,*) 'x min boundary condition is', xmin
c write (*,*) 'y max boundary condition is', ymax
c write (*,*) 'y min boundary condition is', ymin
c write (*,201) ymax
c write (*,201) ymin

C Sets which boundary conditions are periodic a 1 is on a 0 is off
C Is in order of x,y,z boundary condition
max(1)=1
max(2)=1
max(3)=0
min(1)=1
min(2)=1
min(3)=0

C Use Surface atoms to calculate energy of the surface inside the material
satoms=atoms
c write (*,*) '# surface atoms are', satoms
energy=satoms*ecoh
c write(*,*) 'the energy is', energy

C convert from ev to mj
energy=energy*1.6021773d-16
c write (*,*) 'the energy in mJ is', energy

C calculate ground surface energy
surfen=energy/area/2d0
c write (*,*) 'the surface energy is', surfen
surfennon=surfen
gammaenergy=surfen-surfennon

c open (1,file='gammasurface.txt')
c write (1,*) gammaenergy
c write (*,*) 'the gammaenergy is', gammaenergy

C for testing only
c goto 161
C go through the rotated atoms
C assign atoms (row of axis111) into surfaceatoms [surface]
C Assign the atoms crystal rows, which are placed into surfaceatoms, into the sindex vector
C Find the number of atoms in the surfaceatoms structure [satoms]
C i corresponds to the layer #, indexatoms(i) is the # atoms in the layer

l = 1
do 53 i = 1, layers
   do 52 j = 1, indexatoms(i)
      do 260 n = 1, 3
         surface(l, n) = axis111(index(i, j), n)
         sindex(l) = index(i, j)
      end do 260
   end do 52
   continue
end do 53
write (3, *) surface(l, 1), surface(l, 2), surface(l, 3)
l = l + 1
52 continue
53 continue
satoms = l - 1
write (*, *) '# surface atoms are', satoms

call periodicbcenergy(axis111, atoms, surface, satoms, energy)
write (*, *) 'the original energy before optimization is', energy
write (*, *)

C minimize energy by moving layers in (0,0,1) direction of the crystal system
open (70, file = 'stackingfault.txt')
alpha = .002
C sets alpha which is used in the update formula of the steepest decent method
h = 1d-1
C sets h which is used in creating the other set of constants

C number of constants to minimize
m = layers

C assign const1 (Assign negative numbers to the first 4 top layers and than 0 everywhere else)
do 6 j = 1, 4
   const1(j) = -.03d0
6 continue
do 7 j = 5, layers
   const1(j) = .03d0
7 continue
C Assign const2
    do 195 j=1,m
        const2(j)=const1(j)*(1+h)
    195    continue

C This loop iteratively searches for the constant values
which minimize
the sum of errors squared
    do 200 j=1,30
C Calculates the gradient and than the length of the gradient
    write (*,*) 'step', j, ' of optimization'
    call
    gradsolve(grad,const1,const2,m,axis111,sindex,atoms,energy)
    magnitude=vectmagnitude(grad,m)
    write (70,*) 'the original crystal energy is', energy
    write (70,*) 'after step', j, ' length of
    gradient is', magnitude
    c
    write (*,*) 'after step', j, ' length of
    gradient is', magnitude
    c
    write (*,*)
    goto 161

C New parameters are calculated using the steepest descent method
    do 205 k=1,m
        dconst(k)=-1*grad(k)
        const1(k)=const1(k)+dconst(k)*alpha
    205    continue
    do 210 k=1,m
C Calculate other set of parameters (const2) used for calculating
the gradient
    const2(k) = const1(k)*(1d0+h)
    210    continue

C The next step is to see if a local minimum has been found for
the parameters
C if the length of the gradient is less than .05 than terminate
minimization
    if (magnitude .lt. .6d0) goto 215

c end of minimization
    200    continue
C the final parameters and the final length of the gradient
    write (*,*) 'optimization finished'
    write (*,*) 'the final length of gradient is', magnitude
    write (70,*) 'optimization finished'
write (70,*) 'the final length of gradient is', magnitude
write (*,*) 'the final const1 is', const1

C create final crystal using final optimised parameters
C the created Crystal, and surface are in the axis110 frame of reference
   call crystalcreation(const1,m,axis111,atoms)

C assign atoms from crystal to surface to calculate the optimized surface energy.
   open (3, file='surfaceatoms.xyz')
   do 220 i=1,satoms
      do 225 j=1,3
         surface(i,j)=axis111(sindex(i),j)
      225 continue
   220 continue
   call periodicbcenergy(axis111,atoms,surface,satoms,energy)
   write (*,*) 'the energy is', energy

C convert from ev to mJ
   energy=energy*1.6021773d-16
   write (*,*) 'the energy in mJ is', energy

C calculate ground surface energy
   surfen=energy/area/2d0
   write (*,*) 'the surface energy is', surfen
   surfen0=surfen
gammaenergy=surfen-surfen0
   write (1,*) gammaenergy
   write (*,*) 'the gammaenergy is', gammaenergy
   surf111=anint(gammaenergy)
c write (*,*) 'the surface energy of the 111 plane is', surf111

! Calculating the Thermal Conductivity (tcond.py)

#!/usr/bin/env python
#
# tcond.py
#
# read in important information from ./controls/control.txt
# write a file named in.tcond
# Run Lammmps using in.tcond
# Read in Temperature Profile from temp.txt
# use optimizations and objectivefunctions module to fit temperature profile to a linear equation
# read in log.tcond and find the total energy transfered
# use the total energy transfered and slope of the linear equation to calculate the thermal conductivity
# write the thermal conductivity to ans.txt

f=open('controls/control.txt','r')
elenum=int(f.readline().rstrip('
'))
elements=''
elelist=[]
for i in range(elenum):
    elelist.append(f.readline().rstrip('
'))
    elements += elelist[i]
f.close()

f=open('in.tcond','w')
f.write('''# bulk Cu lattice
read_restart 293.equilibrium
reset_timestep 0

pair_style eam/alloy''')

if elenum==0:
    f.write('

pair_coeff * * {0}.eam.alloy
{0}
'.format(elements))
else:
    f.write('

pair_coeff * * {0}.eam.alloy'.format(elements))
    for i in range(elenum):
        if i==elenum-1:
            f.write(' {0}
'.format(elelist[i]))
        else:
            f.write(' {0}'.format(elelist[i]))

f.write(''
neighbor 1.0 bin
neigh_modify every 1 delay 5 check yes

timestep 0.001
thermo 50
compute ke all ke/atom
variable temp atom c_ke/1.5
fix 3 all ave/spatial 10 100 1000 x lower .1 v_temp
file temp.text units reduced ave running
fix 1 all thermal/conductivity 50 x 10

''
fix 2 all nve
#fix 2 all nvt temp 293 293 10
thermo_style custom step temp f_1
log log.tcond
run 15000''
f.close()

import os
os.system('mpirun -np 4 ../../lammps/src/lmp_openmpi<in.tcond')

f=open('temp.text','r')
for line in f:
    row=line.split()
    try: int(row[0]) # Checks to make sure line in file is a data line and not a text line
    except ValueError: continue
    else: #acquires important data
        if len(row)==2:
            #initializes both data lists because this is the begining of a new set of data
            knowntemp=[]
            scaledx=[]
        else: #store data in the above data lists
            knowntemp.append(row[3])
            scaledx.append(row[1])
f.close()

print 'the knowntemp is', knowntemp
print 'the scaled x is', scaledx

# keep the first half + 1 of the data in the lists
knowntemp=knowntemp[:len(knowntemp)/2+1]
scaledx=scaledx[:len(scaledx)/2+1]

# Gets length from log.lammps file
f=open('log.lammps','r')
for line in f:
    row=line.split()
    try: row[0]
    except IndexError: continue
    else:
        if row[0]=='orthogonal':
            minx=float(row[4])
            maxx=float(row[8])
            length=maxx-minx
            break
        else: continue
f.close()
# convert knowntemp to the actual temperature in kelvin and produce new list x corresponding
# to the real distance along the lattice and convert the values into float
kb=0.000086173324
x=[]
for i in range(len(knowntemp)):
    knowntemp[i]=float(knowntemp[i])/kb
    x.append(float(scaledx[i])*length)
print 'the knowntemp is', knowntemp
print 'x is', x

# assign parameters and a function to p and func
# p is a list of the parameters and func is a string with only x and p[i]
# for this specific problem func is linear and there are only 2 parameters
func='p[0]*x+p[1]'
p=range(2)
p[0]=.1 # initial guess for the slope which is good for metals.
p[1]=knowntemp[0] # initial guess for the y intercept which is a descent guess since knowntemp[0] is the
                    # closest to the x axis
print 'the initial parameters are', p

import optimizations, objectivefunctions
sr=objectivefunctions.Sr2yw(x,knowntemp,func,p)
print 'the initial sr^2 is', sr.calculate()
# optimization loop runs steepest descent a maximum of 10 times unless the function meets its minimum criteria
# or the objective function increases
for i in range(10):
    flag=optimizations.steepestdescent(sr)
    srnew=sr.calculate()
    print 'the calculated sr for step', i+1, 'is', srnew
    if flag==1: break
    if i!=0:
        srdiff=srnew-srold
        if srdiff>0: break
    srold=srnew
print 'the final parameters are', sr.p

f=open('log.tcond','r')
for line in f:
    row=line.split()
    try:
        int(row[0]) # Checks to make sure line in file is a data line and not a text line
except ValueError:
    if row[0] == 'Loop': break  # ends data acquisition from file
else: continue  # otherwise continues reading the file
else:  # acquires important data
    heat = float(row[2])
f.close()
print 'the heat transferred is', heat

# calculating the thermal conductivity and setting up/calculating any final parameters needed
perangstrom = 0.00000000001  # converts angstroms to meters
perelectronvolt = 1.602E-19  # converts electron volts to joules
time = 1.5E-11  # if for some reason the in.tcond file gets changed by adding more steps, this value will need changing.
heat = heat * perelectronvolt
tempgrad = sr.p[0] / perangstrom
length = length * perangstrom
tcond = heat / 2.0 / length ** 2 / tempgrad / time

print 'the thermal conductivity is', tcond
f = open('ans.txt', 'w')
f.write(str(tcond))
f.close()

H.37 Calculating the Thermal Expansion (texp.py)

#!/usr/bin/env python
#
# texp.py
#
# read in important information from ./controls/control.txt
# reads in important property information from ./database/’[element names]’materialconst.txt
# writes a file named in.texp
# Run Lammps using in.texp
# Use two different log files to calculate thermal expansion coefficient
# write the thermal expansion coefficient to ans.txt in this folder

f = open('controls/control.txt', 'r')
elemnum = int(f.readline().rstrip('
'))
elements = '
elelist = []
for i in range(elemnum):
    elelist.append(f.readline().rstrip('
'))
elements += elelist[i]
f.close()

propfile='database/'+elements+'materialconst.txt'
f=open(propfile,'r')
a=f.read()
a=a.split(','
) latticesize=float(a[1])
# calculate the berendson pressure control parameter using the bulk modulus
bulkmod=a[4]
bulkmod=bulkmod.rstrip('
')
bulkmod=bulkmod.replace('d','e')
bulkmod=float(bulkmod)
cbulk=1.383e11 #copper bulk modulus
cberendson=100 #copper berendson pressure control parameter
berendson=bulkmod*cberendson/cbulk
f.close()

f=open('in.texp','w')
f.write('''# bulk Cu lattice

read_restart 293.equilibrium
#read_restart 300.equilibrium

pair_style eam/alloy''')

if elenum==0:
    f.write('\npair_coeff * * {0}.eam.alloy {0}\n'.format(elements))
else:
    f.write('\npair_coeff * * {0}.eam.alloy'.format(elements))
    for i in range(elenum):
        if i==elenum-1:
            f.write(' {0}\n'.format(elelist[i]))
        else:
            f.write(' {0}'.format(elelist[i]))

f.write('''
neighbor 1.0 bin
neigh_modify every 1 delay 5 check yes

timestep 0.001
thermo 10

fix 1 all npt temp 293 293 .1 iso 1 1 50 drag 2

thermo_style custom step c_1_temp c_1_press vol
log log.293
run 10000
# stabilize pressure and temperature at an increased temperature
clear

variable x index 1
variable y index 1
variable z index 1

variable xx equal 20*$x
variable yy equal 20*$y
variable zz equal 20*$z

dimension 3
boundary p p p
units metal
atom_style atomic''')

f.write('
lattice fcc {0}\n'.format(latticesize))

f.write('''region box block 0 ${xx} 0 ${yy} 0 ${zz}
create_box 1 box
create_atoms 1 box

pair_style eam/alloy''')

if elenum==0:
    f.write('\pair_coeff * * {0}.eam.alloy
{0}\n'.format(elements))
else:
    f.write('\pair_coeff * * {0}.eam.alloy'.format(elements))
    for i in range(elenum):
        if i==elenum-1:
            f.write(' {0}\n'.format(elelist[i]))
        else:
            f.write(' {0}'.format(elelist[i]))

f.write('''
variable t index 300
velocity all create $t 376847 loop geom

neighbor 1.0 bin
neigh_modify every 1 delay 5 check yes

timestep 0.001
#timestep 0.005
 thermo 10

fix myfix all nve
fix mytfixed all temp/berendsen $t $t .1''')

423
```python
f.write('\\nfix

\n!
!
mypfix all press/berendsen iso 1 1
\{0\}\n'.format(berendson))
f.write('''run

5000

# keep temparture and pressure the same and record thermo data in
new log for the purpose of calculating thermal expansion.
unfix
unfix
unfix
unfix
fix
thermo_style
log
run
''')
```

```
import os
os.system('mpirun -np 4 ../../../lammps/src/lmp_openmpi<in.texp')

Temp293=[]
#initialize temperature list at 293k
Vol293=[]
# initialize volume list at 293k
f=open('log.293','r')
for line in f:
    row=line.split()
    try:
        int(row[0])
        # Checks to make sure line in file is a
data line and not a text line
    except ValueError:
        if row[0]=='Loop': break
        #ends data acquisition from
        file
    else: continue
        # otherwise continues
    reading the file
else: #acquires important data
    Temp293.append(float(row[1]))
    Vol293.append(float(row[3]))
    # calculates the average temperature and volume at 293k
averagetemp293=sum(Temp293)/len(Temp293)
averagevol293=sum(Vol293)/len(Vol293)
f.close()
print 'the average temperature for the 293 run is', averagetemp293
print 'the average volume for the 293 run is', averagevol293
```

```
Temp300=[]
#initialize temperature list at 300k
Vol300=[]
#initialize volume list at 300k
f=open('log.300')
```
int(row[0]) # checks to make sure line in file is a
data line and not a text line
except ValueError:
    if row[0]=='Loop': break #ends data aquisition from
file
else: continue # otherwise continues
reading the file
else: #acquires the important data
    Temp300.append(float(row[1]))
    Vol300.append(float(row[3]))
#calculate the average temperature and volume at 293k
averagetemp300=sum(Temp300)/len(Temp300)
averagevol300=sum(Vol300)/len(Vol300)
f.close()
p
int for the 300 run is',
averagetemp300
print 'the average volume for the 300 run is', averagevol300

#calculate the linear thermal expansion coefficient
ltec=(averagevol300-averagevol293)/(averagetemp300-
averagevol293/3.0/10.0**6
print 'the linear thermal expansion coefficient is', ltec

#calculate the room temperature lattice constant
at=(averagevol293)**(1.0/3.0)/20.0
print 'the room temperature lattice constant is at', at

f=open('ans.txt','w')
f.write('{0} {1}'.format(ltec,at))
f.close()

H.38 Calculating the Unstable Stacking Fault Energy (unstablefault.f)

    subroutine unstablefault(unstable)
    implicit none
    C global variables
    double precision a0,rcut
    common/pot_fitting/a0,rcut
    double precision mass, at
    common/mdprop/mass,at
    double precision ecoh, bulk, omega
    common/matprop/ecoh,bulk,omega
    double precision xmin, xmax, zmin, zmax, ymax, ymin
    common/boundarycondition/xmin,xmax,zmin,zmax,ymax,ymin
    integer max(3), min(3)
    common/pbcon/max,min
    integer satoms,indexatoms(15),layers
    common/specialatomcount/satoms,indexatoms,layers
double precision index(15,150)
common/indexing/index
C program variables
double precision unstable
C local variables
  integer atoms, atoms2
double precision surface(288,3), crystal(288,3),
axis111(288,3), sindex(288)
double precision index2(288), box2_1(288,3), box2(288,3)
double precision row(3), rotoatom(3), rotomat(3,3), rototrans(3,3), rotomat2(3,3), rotomat1(3,3)
double precision bottomsurf(250,3), topsurf(250,3)
double precision zvalues(15), currentz
double precision gammaenergy, surfen0, surfen, energy,
prevgamma
double precision maxx, minx, maxy, miny, nextminy, nextminx,
nextnextminy
double precision height, base, area, area1, area2
integer ymaxloc, yminloc, botsindex, topsindex
double precision step, vectmagnitude, magnitude, h, alpha
double precision const1(10), const2(10), dconst(10), grad(10)
integer i, j, k, l, m, n

c  a0=3.6150000000000d0
c  rcut=5.50679d0
  atoms=288
c  e_coh=-3.54d0
c  write (*,201) a0
c  call struct111(a0)
C read in atom structure and store in crystal
  open (1, file='(111).xyz')
do 10 i=1, atoms
    read (1, *) crystal(i,1), crystal(i,2), crystal(i,3)
10 continue
  close (1)
c  write (*,201) crystal

C rotation matrix1 for rotating by 30 degrees around the z axis
rotomat1(1,1)=sqrt(3d0)/2d0
rotomat1(1,2)=-.5d0
rotomat1(1,3)=0
rotomat1(2,1)=.5d0
rotomat1(2,2)=sqrt(3d0)/2d0
rotomat1(2,3)=0d0
rotomat1(3,1)=0D0
rotomat1(3,2)=0D0
rotomat1(3,3)=1D0

426
c write (*,*) 'rotomat1 is', rotomat1

c write (*,201) rotomat1

c WRITE (*,*)

C rotation matrix2 for converting axis system to (111) plane

rotomat2(1,1) = -1/sqrt(6d0)
rotomat2(1,2) = -1/sqrt(6d0)
rotomat2(1,3) = 2/sqrt(6d0)
rotomat2(2,1) = 1/sqrt(2d0)
rotomat2(2,2) = -1/sqrt(2d0)
rotomat2(2,3) = 0d0
rotomat2(3,1) = 1/sqrt(3d0)
rotomat2(3,2) = 1/sqrt(3d0)
rotomat2(3,3) = 1/sqrt(3d0)

C calculate rotation matrix (rotomat) which is rotomat1*rotomat2

DO 2 I = 1, 3
 DO 2 J = 1, 3
    rotomat(I,J) = 0D0
    DO 1 K = 1, 3
       rotomat(I,J) = rotomat(I,J) + rotomat1(I,K) * rotomat2(K,J)
1 CONTINUE
2 CONTINUE

C Converts numbers that should be zero to zero

DO 3 i=1,3
   DO 4 j=1,3
      IF (ABS(ROTOMAT(i,j)) .LT. 1D-10) THEN
      ROTOMAT(i,j) = 0D0
     ELSE
      CONTINUE
     END IF
4 CONTINUE
3 CONTINUE

C Define Transpose of Rotation matrix

rototrans(1,1) = rotomat(1,1)
rototrans(1,2) = rotomat(2,1)
rototrans(1,3) = rotomat(3,1)
rototrans(2,1) = rotomat(1,2)
rototrans(2,2) = rotomat(2,2)
rototrans(2,3) = rotomat(3,2)
rototrans(3,1) = rotomat(1,3)
rototrans(3,2) = rotomat(2,3)
rototrans(3,3) = rotomat(3,3)
C Rotate atoms in crystal so 2 axis are parallel with (111) surface.
C Store results in axis111.
C Also keep Crystal in original axis system.
c open (1,file='(111)rotate.xyz')
do 20 i=1,atoms
do 30 j=1,3
   row(j)=crystal(i,j)
30 continue
c write (*,*), row
   call dmvcalc(3,3,rotomat,row,rotoatom)
c write (*,*), rotoatom
do 40 j=1,3
   axis111(i,j)=rotoatom(j)
   if (abs(axis111(i,j)) .lt. 1d-8) axis111(i,j)=0
40 continue
c write (1,*) axis111(i,1), axis111(i,2), axis111(i,3)
20 continue
c close(1)

C go through the atoms in axis111
C Find the unique Z values of the atoms and place them into zvalues.
C Assign the number of unique zvalues as layers
C Order zvalues from greatest to least
C Assign atoms to an index matrix where the row corresponds to the Zlayers in Zvalues
C Assign the number of atoms in each row of the index matrix to the corresponding row in indexatoms
j=1
C stores first atom z value in zvalues
   zvalues(j)=axis111(1,3)
   j=j+1
do 50 i=2,atoms
   do 51 k=1,j-1
      if (real(axis111(i,3)) .eq. real(zvalues(k)) goto 50
   continue
   zvalues(j)=axis111(i,3)
   j=j+1
50 continue
c write (*,*), 'zvalues is', zvalues
   layers=j-1
c write (*,*), 'number of z layers is', layers
do 56 i=1,layers
  c write (*,*) 'zvalues is', zvalues, 'before step', i
  currentz=zvalues(i)
C zeroing indexatoms
  indexatoms(i)=0
  do 57 j=i-1,1,-1
    zvalues(j+1)=zvalues(j)
    if (currentz .lt. zvalues(j)) then
      zvalues(j+1)=currentz
      goto 56
    elseif (j .eq. 1) then
      zvalues(j)=currentz
    endif
  continue
  57 continue
  56 continue
C found right slot for current z to fit in so begin algorithm for
next zvalue
  goto 56
endif
57 continue
56 continue
C create bottom and top Surface using axis111 matrix
k=1
l=1
c open (1,file='bottomsurface.xyz')
c open (2,file='topsurface.xyz')
do 25 i=1,atoms
  if (axis111(i,3) .eq. 0) then
    do 35 j=1,3
      bottomsurf(k,j)=axis111(i,j)
    continue
  endif
  write (1,*) bottomsurf(k,1), bottomsurf(k,2), bottomsurf(k,3)
  k=k+1
if (axis111(i, 3) .gt. 0 .and. axis111(i, 3) .le. 3)
then
    do 45 j=1,3
        topsurf(l,j)=axis111(i,j)
        continue
    c 45
topsurf(l,3)
    l=l+1
endif
25 continue
c 202  format (D23.16, D25.18, D25.18)
botsindex=k-1
topsindex=l-1
c  write (*,*) 'bottom surface has this number of
atoms',botsindex
c  write (*,*) 'top surface has this number of atoms', topsindex
c  close(1)
c  close(2)
C Find maximum and minium y values as well as their position of
the bottom surface
  yminloc=1
  ymaxloc=1
  maxy=bottomsurf(1,2)
  nextminy=bottomsurf(1,2)
  miny=bottomsurf(1,2)
  do 55 i=2,botsindex
    if (bottomsurf(i,2) .lt. 1.01*miny) then
      if (miny .lt. 1.01*nextminy .and. nextminy .lt.
1.01*nextnextminy) then
        nextnextminy=nextminy
        nextminy=miny
      elseif (miny .lt. nextminy) then
        nextminy=miny
      endif
      miny=bottomsurf(i,2)
yminloc=i
    elseif (bottomsurf(i,2) .lt. 1.01*nextminy .and.
bottomsurf(i,2) .gt. .99*miny) then
      if (nextminy .lt. 1.01*nextnextminy)
        nextnextminy=nextminy
        nextminy=bottomsurf(i,2)
      elseif (bottomsurf(i,2) .lt. 1.01*nextnextminy .and.
bottomsurf(i,2) .gt. .99*nextminy) then
        nextnextminy=bottomsurf(i,2)
      elseif (bottomsurf(i,2) .gt. maxy) then
        maxy=bottomsurf(i,2)
ymaxloc=i
      endif
endif
55    continue
C write (*,201) nextnextminy
C write (*,201) nextminy
C write (*,201) miny
C write (*,*) 'max y position is', ymaxloc
C write (*,*) 'max y is', maxy
C write (*,*) 'min y position is', yminloc
C write (*,*) 'min y is', miny
C write (*,*) 'nextminy is', nextminy
C write (*,*) 'nextnextminy is', nextnextminy

C set min and max y as the periodic boundary conditions
    ymin=miny
    ymax=miny+(nextnextminy-miny)*3
    ymax=maxy+(nextminy-miny)

C calculate surface area of bottom surface (shape=parrallelogram)
C area=base*height
    height=maxy-miny
    height=ymax-ymin
C write (*,*) 'height is', height
C find x positions of the corners of the ymin side.
    minx=0
    nextminx=0
    maxx=0
    do 65 i=1,botsindex
        if (bottomsurf(i,2) .ge. (1.01*miny) .and. bottomsurf(i,2) .le. (.99*miny)) then
            write (*,*) 'location of miny is', i
            if (bottomsurf(i,1) .lt. minx) then
                if (minx .lt. nextminx) nextminx=minx
                minx=bottomsurf(i,1)
            elseif (bottomsurf(i,1) .lt. nextminx .and. bottomsurf(i,1) .gt. minx) then
                nextminx=bottomsurf(i,1)
            elseif (bottomsurf(i,1) .gt. maxx) then
                maxx=bottomsurf(i,1)
            endif
        endif
    enddo 65
    continue
C write (*,*) 'minx is', minx
C write (*,*) 'nextminx is', nextminx
C write (*,*) 'maxx is', maxx

C set min and max x as the periodic boundary condition
    xmin=minx
    xmax=minx+6*(nextminx-minx)
    xmax=maxx+(nextminx-minx)
C calculate the base and than calculate the area of the bottom surface
C
base=maxx-minx
base=xmax-xmin
area=base*height-(nextminx-minx)*(nextminy-miny)
area=base*height
write (*,*) 'base is', base
write (*,*) 'area is', area
C convert area from angstrom^2 to meter^2
area=1d-20*area
write (*,*) 'total area in m^2', area
C Boundary Conditions of axis111
C
write (*,*) 'x max boundary condition is', xmax
write (*,*) 'x min boundary condition is', xmin
write (*,*) 'y max boundary condition is', ymax
write (*,*) 'y min boundary condition is', ymin
write (*,201) ymax
write (*,201) ymin
C Sets which boundary conditions are periodic a 1 is on a 0 is off
C Is in order of x,y,z boundary condition
max(1)=1
max(2)=1
max(3)=0
min(1)=1
min(2)=1
min(3)=0
C Use Surface atoms to calculate the energy of the surface of the bottombox and top box
C inside the material
satoms=4*botsindex
C
write (*,*) '# surface atoms are', satoms
energy=satoms*ecoh
write(*,*) 'the energy is', energy
C convert from ev to mj
energy=energy*1.6021773d-16
write (*,*) 'the energy in mJ is', energy
C calculate ground surface energy
surfen=energy/area
write (*,*) 'the surface energy is', surfen
surfena0=surfen
gammaenergy=surfen-surfen0
open (1,file='gammasurface.txt')
c write (1,*) gammaenergy
c write (*,*) 'the gammaenergy is', gammaenergy

C setup previous gamma energy=gamma energy
prevgamma=gammaenergy

C go through the rotated atoms
C assign atoms (row of axis111) that have z<=0 into index1
(corresponds bottombox) [dont need anymore]
C assign atoms (row of axis111) that have z>0 into index2
(corresponds to topbox)
C assign atoms (row of axis111) that have z>=-3 and z<=5 into
sindex (corresponds to surfaces)
k=1
   do 60 i=1,atoms
      if (axis111(i,3) .gt. 0) then
         index2(k)=i
         k=k+1
      endif
60   continue
atoms2=k-1
l=1
   do 53 i=3,6
      do 52 j=1,indexatoms(i)
         do 261 n=1,3
            surface(l,n)=axis111(index(i,j),n)
            sindex(l)=index(i,j)
261      continue
c write (3,*) surface(l,1), surface(l,2),
surface(l,3)
   l=l+1
52   continue
53   continue
satoms=l-1
c write (*,*) '# surface atoms are', satoms
c write (*,*) sindex

C for testing only
c goto 161
c open (70,file='stackingfault.txt')
C begin finding the stable and unstable stacking fault
C Repeatedly move topbox in step*(-2, 1, 1) direction
   step=at/240d0
C the 40 steps shifts topbox atoms a total of 1/6*at<-2 1 1>
direction
C in the regular crystal system
   do 100 i=1,40
C assign atoms in axis111 to top box keeping the axis111 frame of reference
    do 240 j=1,atoms2
        do 245 k=1,3
            box2_1(j,k)=axis111(index2(j),k)
        continue
    245 continue
    240 continue
C convert top box from axis111 frame of reference to original crystal frame of reference
    do 250 k=1,atoms2
        do 255 j=1,3
            row(j)=box2_1(k,j)
        continue
    255 continue
    c write (*,*) row
    call dmvcalc(3,3,rototrans,row,rotoatom)
c write (*,*) rotoatom
    do 260 j=1,3
        box2(k,j)=rotoatom(j)
        if (abs(box2(k,j)) .lt. 1e-14) box2(k,j)=0
    260 continue
    250 continue
C Assign atoms from topbox to their corresponding atoms in axis111.
C Keep the axis 111 frame of reference
    do 105 j=1,atoms2
        box2(j,1)=box2(j,1)+step*2d0
        box2(j,2)=box2(j,2)+step*1d0
        box2(j,3)=box2(j,3)+step*1d0
    105 continue
C Convert topbox to the axis111 frame of reference.
    do 165 k=1,atoms2
        do 170 j=1,3
            row(j)=box2(k,j)
        continue
    170 continue
C write (*,*) row
    call dmvcalc(3,3,rotomat,row,rotoatom)
c write (*,*) rotoatom
    do 175 j=1,3
        box2_1(j)=rotoatom(j)
        if (abs(box2_1(j)) .lt. 1e-14) box2_1(j)=0
    175 continue
    165 continue
c open (2, file='topboxexam.xyz')
c open (3, file='topboxexam2.xyz')
do 12 j=1, atoms
   do 13 k=1, 3
      axis111(index2(j), k) = box2_1(j, k)
   13 continue
c if (i .eq. 15) then
   c write (2, *) box2_1(j, 1), box2_1(j, 2), box2_1(j, 3)
   endif
c if (i .eq. 30) then
   c write (3, *) box2_1(j, 1), box2_1(j, 2), box2_1(j, 3)
   endif
12 continue
c close (2)
c close (3)

C assign atoms from axis111 to surface
do 11 j=1, satoms
   do 14 k=1, 3
      surface(j, k) = axis111(sindex(j), k)
   14 continue
11 continue

c Calculate Original Energy of the Surface in the axis111 system
   call periodicbcenergy(axis111, atoms, surface, satoms, energy)
c write (*, *) 'the energy of the surface before optimization is', energy
c write (70, *) 'the energy of the surface before optimization is', energy
c write (*, *)

C minimize energy by moving layers of atoms in (0,0,1) direction
C this direction is according to axis111 frame of reference. In terms of the crystal framework
C this direction would be (1,1,1) direction.
   alpha=.002
C sets alpha which is used in the update formula of the steepest decent method
   h=1d-1
C sets h which is used in creating the other set of constants

C number of constants to minimize
   m=layers

C Assign const1 (Assign Positive numbers for the top box and negative numbers for the bottom box)
do 16 j=1,4
   const1(j)=.0415d0
16    continue
do 17 j=5,m
   const1(j)=-.0415d0
17    continue
C .04
c.033
C Assign const2
do 110 j=1,m
   const2(j)=const1(j)*(1+h)
110    continue
C This loop iteratively searches for the constant values which minimize the sum of errors squared
do 115 j=1,17
C Calculates the gradient and than the length of the gradient
   write (*,*) 'step', j, ' of optimization'
   write (70,*), 'step', j, ' of optimization'
call gradsolve(grad,const1,const2,m,axis111,sindex,atoms,energy)
   magnitude=vectmagnitude(grad,m)
   write (70,*) 'the original crystal energy is', energy
   write (70,*), 'after step', j, ' length of gradient is', magnitude
   write (70,*), 'after step', j, ' length of gradient is', magnitude
   write (*,*)
C New parameters are calculated using the steepest descent method
do 125 k=1,m
   dconst(k)=-1*grad(k)
   const1(k)=const1(k)+dconst(k)*alpha
125    continue
call linesearch (grad,dconst,const1,m,atoms)
do 120 k=1,m
C Calculate other set of parameters (const2) used for calculating the gradient
   const2(k) = const1(k)*(1d0+h)
120    continue
C The next step is to see if a local minimum has been found for the parameters
C if the length of the gradient is less than .05 than terminate minimization
if (magnitude .lt. 1.75d0) goto 130

C end of minimization

C the final parameters and the final length of the gradient
write (*,'optimization finished')
write (*,'the final length of gradient is', magnitude)
write (70,*,'optimization finished')
write (70,*,'the final length of gradient is', magnitude)
write (*,'the final const 1 is', const1)

C create final crystal using final optimized parameters
C the created Crystal, surface and boxes are in the axis111 frame of reference

call crystalcreation(const1,m,axis111,atoms)

C create new surface in the axis111 frame of reference using final crystal
open (2,file='surfaceexam.xyz')
open (3,file='surfaceexam2.xyz')
do 155 j=1,satoms
do 160 k=1,3
   surface(j,k)=axis111(sindex(j),k)
160    continue
if (i .eq. 15) then
   write (2,*,'(surface(j,1), surface(j,2), surface(j,3))')
endif
if (i .eq. 30) then
   write (3,*,'(surface(j,1), surface(j,2), surface(j,3))')
endif
155    continue
close (2)
close (3)

C calculate energy of the surface inside the crystal.
call periodicbcenergy(axis111,atoms,surface,satoms,energy)
write (*,'the final energy is', energy)
write (70,*,'the final energy is', energy)
energy=energy*1.6021773d-16
write (*,'the energy in mj is', energy)
C calculate gamma energy
surfen=energy/area
c  write (*,*) 'the surface energy is', surfen
c  write (70,*) 'the surface energy is', surfen
gammaenergy=surfen-surfen0
c  write (1,*) gammaenergy
c  write (*,*) 'the gamma energy is', gammaenergy
c  write (70,*) 'the gamma energy is', gammaenergy
c  write (*,*)
c  write (70,*)

C Check to see if top box has been moved past the unstable
C and if not set prevgamma as the current calculated gamma.
C When the top box has moved past the unstable stacking fault, the
gammaenergy will decrease,
C and the unstable stacking fault is prevgamma because the
gammaenergy is a maximum.
if (gammaenergy .lt. prevgamma) then
  if (i .eq. 1) then
    prevgamma=gammaenergy
  else
    goto 162
  endif
else
  prevgamma=gammaenergy
endif
C For testing purposes only
if (i .eq. 1) goto 161
C Finished both the top box move in step*<-2 1 1> direction
C and the energy optimization of the two box surfaces.
100  continue

162  if (gammaenergy .lt. prevgamma) then
      unstable=anint(prevgamma)
c      write (*,*) 'unstable stacking fault energy is ',
      unstable
      else
        unstable=anint(prevgamma)
c        write (*,*) 'unstable stacking fault energy is ',
        unstable
      endif
      write (*,*) ', which is the same as the stable stacking
      fault energy'
c      write (*,*) 'due to position being the same as the
      stable stacking fault.'
    endif

c 161  close (1)
c  close (70)
H.39 Optimization for Calculating Vacancy Formation Energy

(vacancy_conjugate.f)

C----------------------------------------------------------------
---ZXCG0030
C
ZXCG0040
C COMPUTER          - HP9000/D Double
ZXCG0050
C
ZXCG0060
C LATEST REVISION   - November 1, 1979
ZXCG0070
C
ZXCG0080
C PURPOSE           - A CONJUGATE GRADIENT ALGORITHM FOR
FINDING   ZXCG0090
C VARIABLES    ZXCG0100
C
ZXCG0110
C USAGE              - CALL ZXCR
(FUNCT,N,ACC,MN,DF,PR,X,G,F,W, ZXCG0120
C
ZXCG0130
C
ZXCG0140
C ARGUMENTS    FUNCT - A USER SUPPLIED SUBROUTINE WHICH
CALCULATES    ZXCG0150
C GRADIENT    ZXCG0160
C
ZXCG0170
C
ZXCG0180
C FOLLOWING FORM ZXCG0190
C
ZXCG0200
C N.    ZXCG0210
C FUNCTION.    ZXCG0220
C COMPONENTSZXCG0230

end
OF THE GRADIENT OF F.
FUNCT MUST APPEAR IN AN EXTERNAL
IN THE CALLING PROGRAM. FUNCT MUST
ALTER THE VALUES OF X(I), I = 1, ..., N OR

- THE NUMBER OF PARAMETERS OF THE
FUNCTION. (INPUT) (I.E., THE LENGTH OF

N - THE NUMBER OF PARAMETERS OF THE
OBJECTIVE FUNCTION. (INPUT) (I.E., THE LENGTH OF
X)

ACC - CONVERGENCE CRITERION. (INPUT)
THE CALCULATION ENDS WHEN THE SUM OF
OF THE COMPONENTS OF G IS LESS THAN

MAXFN - MAXIMUM NUMBER OF FUNCTION EVALUATIONS
CALLS TO SUBROUTINE FUNCT) ALLOWED.
IF MAXFN IS SET TO ZERO, THEN THERE
NO RESTRICTION ON THE NUMBER OF
EVALUATIONS.

DFPRED - A ROUGH ESTIMATE OF THE EXPECTED
REDUCTION IN F, WHICH IS USED TO DETERMINE THE
OF THE INITIAL CHANGE TO X. (INPUT)
NOTE THAT DFPRED IS THE EXPECTED
ITSELF, AND DOES NOT DEPEND ON ANY
A BAD VALUE OF DFPRED CAUSES AN ERROR
MESSAGE, WITH IER = 129, AND A RETURN
FIRST ITERATION. (SEE THE DESCRIPTION
IER BELOW)

X - VECTOR OF LENGTH N CONTAINING PARAMETER
VALUES.
ON INPUT, X MUST CONTAIN THE INITIAL PARAMETER ESTIMATES.
ON OUTPUT, X CONTAINS THE FINAL ESTIMATES AS DETERMINED BY ZXCR.

- A VECTOR OF LENGTH N CONTAINING THE COMPONENTS OF THE GRADIENT OF F AT
  FINAL PARAMETER ESTIMATES. (OUTPUT)

- A SCALAR CONTAINING THE VALUE OF THE AT THE FINAL PARAMETER ESTIMATES.
- WORK VECTOR OF LENGTH 6*N.
- ERROR PARAMETER. (OUTPUT)

IER = 0 IMPLIES THAT CONVERGENCE WAS ACHIEVED AND NO ERRORS OCCURRED.
TERMINAL ERROR
IER = 129 IMPLIES THAT THE LINE AN INTEGRATION WAS ABANDONED. THIS ERROR MAY BE CAUSED BY AN ERROR IN GRADIENT.
IER = 130 IMPLIES THAT THE CALCULATION CANNOT CONTINUE BECAUSE THE SEARCH DIRECTION IS UPHILL.
IER = 131 IMPLIES THAT THE ITERATION TERMINATED BECAUSE MAXFN WAS
IER = 132 IMPLIES THAT THE
CALCULATION WAS TERMINATED BECAUSE TWO
CONSECUTIVE ITERATIONS FAILED TO REDUCE F.

PRECISION/HARDWARE - SINGLE AND DOUBLE/H32
- SINGLE/H36,H48,H60

REQD. IMSL ROUTINES - UERTST,UGETIO

NOTATION - INFORMATION ON SPECIAL NOTATION AND
CONVENTIONS IS AVAILABLE IN THE
INTRODUCTION OR THROUGH IMSL ROUTINE
UHELP

REMARKS 1. THE ROUTINE INCLUDES NO THOROUGH CHECKS ON THE
PART OF THE USER PROGRAM THAT CALCULATES THE
DERIVATIVES OF THE OBJECTIVE FUNCTION. THEREFORE, BECAUSE
DERIVATIVE CALCULATION IS A FREQUENT SOURCE OF
ERROR, THE USER SHOULD VERIFY INDEPENDENTLY THE
CORRECTNESS OF THE DERIVATIVES THAT ARE GIVEN TO
THE ROUTINE.

2. BECAUSE OF THE CLOSE RELATION BETWEEN THE
CONJUGATE GRADIENT METHOD AND THE METHOD OF STEEPEST
DESCENTS, IT IS VERY HELPFUL TO CHOOSE THE SCALE OF THE
VARIABLES IN A WAY THAT BALANCES THE MAGNITUDES
THE COMPONENTS OF A TYPICAL DERIVATE VECTOR. IT CAN BE PARTICULARLY INEFFICIENT IF A FEW COMPONENTS OF THE GRADIENT ARE MUCH LARGER THAN THE REST.

SUBROUTINE vZXCGRF (N,ACC,MAXFN,DFPRED,X,G,F,W,IER)

INTEGER N,MAXFN,IER

DOUBLE PRECISION ACC,DFPRED,X(N),G(N),F,W(1)

DATA MAXLIN/M5/,MXFCON/M2/

FIRST EXECutable STATEMENT
IER = 0
THE WORKING SPACE ARRAY IS
SPLIT INTO SIX VECTORS OF LENGTH
N. THE FIRST PART IS USED FOR THE
SEARCH DIRECTION OF AN ITERATION.
THE SECOND AND THIRD PARTS
CONTAIN THE
INFORMATION THAT IS REQUIRED
THE CONJUGACY CONDITIONS OF
RESTART PROCEDURE. THE
CONTAINS THE GRADIENT AT THE
OF AN ITERATION. THE FIFTH
CONTAINS THE PARAMETERS THAT
THE LEAST CALCULATED VALUE
THE SIXTH PART CONTAINS THE
GRADIENT VECTOR WHERE F IS

SET SOME PARAMETERS TO BEGIN
THE CALCULATION. ITERC AND
NCALLS COUNT THE NUMBER OF
ITERATIONS AND CALLS OF
ITERFM IS THE NUMBER OF THE
RECENT ITERATION THAT

CALL SUBROUTINE FUNCT. LET THE
INITIAL SEARCH DIRECTION BE
THE GRADIENT VECTOR. USUALLY
PARAMETER ITERRS GIVES THE
ITERATION NUMBER OF THE MOST
RECENT RESTART, BUT IT IS
ZERO WHEN THE STEEPEST
DIRECTION IS USED.

5 NCALLS = NCALLS+1
CALL vFUNCT(N,X,F,G)
IF (NCALLS.GE.2) GO TO 20
10 DO 15 I=1,N
15 W(I) = -G(I)
ITERRS = 0
IF (ITERC.GT.0) GO TO 80
20 GNEW = 0.0D0
SUM = 0.0D0
DO 25 I=1,N
25
GNEW = GNEW+W(I)*G(I)

25 SUM = SUM+G(I)**2

IF (NCALLS.EQ.1) GO TO 35

FCH = F-FMIN

C STORE THE VALUES OF X, F AND
G, IF THEY ARE THE BEST THAT HAVE
BEEN CALCULATED SO FAR, AND NOTE
SQUARED AND THE VALUE OF
NCALLS.

C

IF (FCH) 35,30,50

30 IF (GNEW/GMIN.LT.-1.0D0) GO TO 45

35 FMIN = F

GQRST = SUM

NFOPT = NCALLS

DO 40 I=1,N

W(IxoPT+I) = X(I)

40 W(IgoPT+I) = G(I)

WRITE(*,*) 'SUM vs ACC',SUM,ACC

IF (SUM.LE.ACC) GO TO 9005

TEST IF THE VALUE OF MAXFN
ALLOWS ANOTHER CALL OF FUNCT.

50 IF (NCALLS.NE.MAXFN) GO TO 55

IER = 131

GO TO 9000

55 IF (NCALLS.GT.1) GO TO 100

ZXC1970

ZXC1980

ZXC1990

ZXC2000

ZXC2010

ZXC2020

ZXC2030

ZXC2040

ZXC2050

ZXC2060

ZXC2070

ZXC2080

ZXC2090

ZXC2100

ZXC2110

ZXC2120

ZXC2130

ZXC2140

ZXC2150
SET DFPR TO THE ESTIMATE OF REDUCTION IN F GIVEN IN THE ARGUMENT LIST, IN ORDER THAT INITIAL CHANGE TO THE IS OF A SUITABLE SIZE. THE VALUE OF STMIN IS USUALLY THE STEP-LENGTH OF THE MOST LINE SEARCH THAT GIVES THE CALCULATED VALUE OF F.

DFPR = DFPRED
STMIN = DFPRED/GSQRD

BEGIN THE ITERATION

STORE THE INITIAL FUNCTION GRADIENT, CALCULATE THE DIRECTIONAL DERIVATIVE, AND IF ITS VALUE IS NOT SBOUND TO MINUS ONE TO THAT A BOUND ON THE STEP IS KNOWN YET, AND SET NFBEG TO CURRENT VALUE OF NCALLS. THE PARAMETER IRETRY SHOWS THE NUMBER OF ATTEMPTS AT SATISFYING CONDITION.

ITERC = ITERC+1
VALUE AND ZXC2220
INITIAL ZXC2300
BRANCH ZXC2310
NEGATIVE, SET ZXC2320
INDICATE ZXC2330
NOT ZXC2340
THE ZXC2350
ZXC2360
NUMBER ZXC2370
THE BETA ZXC2380
ZXC2390
FINIT = F

GINIT = 0.0D0

DO 85 I=1,N

W(IGINIT+I) = G(I)

85 GINIT = GINIT+W(I)*G(I)

IF (GINIT.GE.0.0D0) GO TO 165

GMIN = GINIT

SBOUND = -1.0D0

NFBEG = NCALLS

IRETRY = -1

C

WITH THE ZXCG2510

SUBJECT ZXCG2520

DOES NOT ZXCG2530

THE ZXCG2540

STMIN BE ZXCG2550

CALCULATED ZXCG2560

STEPCH = DMIN1(STMIN,DABS(DFPR/GINIT))

ZXC2580

STMIN = 0.0D0

CALL SUBROUTINE FUNCT AT THE

OF X THAT IS DEFINED BY THE

CHANGE TO THE STEP-LENGTH,

THE NEW STEP-LENGTH BE STEP.
C VARIABLE WORK IS USED AS WORK.
C
ZXCG2650
90 STEP = STMIN+STEPCH
ZXCG2660
WORK = 0.0D0
ZXCG2670
DO 95 I=1,N
ZXCG2680
X(I) = W(I)+STEPCH*W(I)
ZXCG2690
95 WORK = DMAX1(WORK,DABS(X(I)-W(I)))
ZXCG2700
IF (WORK.GT.0.0D0) GO TO 5
ZXCG2710
C TERMINATE THE LINE SEARCH IF
STEPCH
C IS EFFECTIVELY ZERO.
ZXCG2730
IF (NCALLS.GT.NFBEG+1) GO TO 115
ZXCG2740
IF (DABS(GMIN/GINIT)-0.2D0) 170,170,115
ZXCG2750
C LET SPLN BE THE QUADRATIC
SPLINE
C THAT INTERPOLATES THE
CALCULATED
C FUNCTION VALUES AND
DIRECTIONAL
C DERIVATIVES AT THE POINTS
STMIN
C AND STEP OF THE LINE SEARCH,
WHERE
ZXCG2800
C THE KNOT OF THE SPLINE IS AT
ZXCG2810
C 0.5*(STMIN+STEP). REVISE
STMIN,
ZXCG2820
C GMIN AND SBOUND, AND SET
DDSPLN TO
ZXCG2830
C THE SECOND DERIVATIVE OF
SPLN AT
ZXCG2840
C THE NEW STMIN. HOWEVER, IF
FCH IS
ZXCG2850
C ZERO, IT IS ASSUMED THAT THE
MAXIMUM ACCURACY IS ALMOST

450
C ACHIEVED, SO DDSPLN IS CALCULATED USING ONLY THE CHANGE IN THE GRADIENT.

100 WORK = (FCH+FCH)/STEPCH-GNEW-GMIN

DDSPLN = (GNEW-GMIN)/STEPCH

IF (NCALLS.GT.NFOPT) SBOUND = STEP

IF (NCALLS.GT.NFOPT) GO TO 105

IF (GMIN*GNEW.LE.0.0D0) SBOUND = STMIN

STMIN = STEP

GMIN = GNEW

STEPCH = -STEPCH

105 IF (FCH.NE.0.0D0) DDSPLN = DDSPLN+(WORK+WORK)/STEPCH

C

C TEST FOR CONVERGENCE OF THE SEARCH, BUT FORCE AT LEAST TWO STEPS TO BE TAKEN IN ORDER NOT TO LOSE QUADRATIC TERMINATION.

110 IF (NCALLS.LT.NFOPT+MAXLIN) GO TO 120

115 IER = 129
GO TO 170
ZXC3120
C
CHANGE TO ZXC3130
C
THAT IS ZXC3140
C
LINE ZXC3150
C
GRADIENT ZXC3160
C
ZXC3170
C
CALCULATE ZXC3180
C
MINIMIZES ZXC3190
C
THEN ZXC3200
C
ZXC3210
C
VALUE OF ZXC3220
C
LENGTH. ZXC3230
   120  STEPCH = 0.5D0*(SBOUND-STMIN)
ZXC3240
   IF (SBOUND.LT.-0.5D0) STEPCH = 9.0D0*STMIN
ZXC3250
   GSPLN = GMIN+STEPCH*DDSPLN
ZXC3260
   IF (GMIN*GSPLN.LT.0.0D0) STEPCH = STEPCH*GMIN/(GMIN-GSPLN)
ZXC3270
   GO TO 90
ZXC3280
C
CALCULATE THE VALUE OF BETA
C
THAT OCCURS IN THE NEW SEARCH
C
DIRECTION.
ZXC3300
C
ZXC3310
   125  SUM = 0.0D0
ZXC3320
   DO 130 I=1,N
ZXC3330
   130  SUM = SUM+G(I)*W(IGINIT+I)
ZXC3340
   BETA = (GSQRD-SUM)/(GMIN-IGINIT)
ZXC3350
TEST THAT THE NEW SEARCH DIRECTION CAN BE MADE DOWNHILL. IF IT CANNOT, THEN MAKE ONE ATTEMPT TO IMPROVE THE ACCURACY OF THE SEARCH.

IF $(\text{DABS(BETA*GMIN)} \leq 0.2*GSQRD)$ GO TO 135

IRETRY = IRETRY + 1

IF (IRETRY \leq 0) GO TO 110

APPLY THE TEST THAT DEPENDS ON THE PARAMETER MXFCON.

SET DFPR TO THE PREDICTED REDUCTION IN F ON THE NEXT ITERATION.

IF (F LT. FINIT) ITERFM = ITERC

IF (ITERC LT. ITERFM + MXFCON) GO TO 140

IER = 132

GO TO 9000

DFPR = STMIN*GINIT

BRANCH IF A RESTART PROCEDURE REQUIRED DUE TO THE NUMBER OR DUE TO THE SCALAR PRODUCT OF CONSECUTIVE GRADIENTS.

IF (IRETRY GT. 0) GO TO 10

IF (ITERRS EQ. 0) GO TO 155

453
IF (ITERC - ITERRS .GE. N) GO TO 155
ZXCG3600
  IF (DABS(SUM) .GE. 0.2D0 * GSQRD) GO TO 155
ZXCG3610
C CALCULATE THE VALUE OF GAMA THAT OCCURS IN THE NEW SEARCH
C
ZXCG3620
C DIRECTION, AND SET SUM TO A
C
SCALAR ZXCG3640
C PRODUCT FOR THE TEST BELOW.
C
THE ZXCG3650
C VALUE OF GAMDEN IS SET BY
C
THE ZXCG3660
C RESTART PROCEDURE.
ZXCG3670
  GAMA = 0.0D0
ZXCG3680
  SUM = 0.0D0
ZXCG3690
    DO 145 I=1,N
ZXCG3700
      GAMA = GAMA + G(I) * W(IRSDG+I)
ZXCG3710
    145 SUM = SUM + G(I) * W(IRSDX+I)
ZXCG3720
      GAMA = GAMA / GAMDEN
ZXCG3730
C RESTART IF THE NEW SEARCH DIRECTION IS NOT SUFFICIENTLY
C
DOWNHILL. ZXCG3750
C
ZXCG3760
    IF (DABS(BETA * GMIN + GAMA * SUM) .GE. 0.2D0 * GSQRD) GO TO 155
ZXCG3770
C
ZXCG3780
C CALCULATE THE NEW SEARCH DIRECTION.
ZXCG3790
    DO 150 I=1,N
ZXCG3800
      150 W(I) = -G(I) + BETA * W(I) + GAMA * W(IRSDX+I)
ZXCG3810
      GO TO 80
ZXCG3820
C APPLY THE RESTART PROCEDURE.
ZXCG3830
GAMDEN = GMIN - GINIT

DO 160 I=1,N

W(IRSDX+I) = W(I)
W(IRSDG+I) = G(I) - W(IGINIT+I)

160 W(I) = -G(I) + BETA*W(I)

ITERRS = ITERC
GO TO 80

C SET IER TO INDICATE THAT THE SEARCH DIRECTION IS UPHILL.

165 IER = 130

C ENSURE THAT F, X AND G ARE OPTIMAL.

IF (NCALLS.EQ.NFOPT) GO TO 180

F = FMIN
DO 175 I=1,N
X(I) = W(IXOPT+I)

G(I) = W(IGOPT+I)

180 IF (IER.EQ.0) GO TO 125

CONTINUE

CALL UERTST (IER,'ZXCGRF77 ')

RETURN

END

H.40 Calculating the Vacancy Formation Energy (vacancy.f)

C****************************************************************
********
C     Identify the energy minimum using the Conjugate Gradient
method

********
C Identify the energy minimum using the Conjugate Gradient
Energy unit: eV
sub_conjugate asks for gradient of energy, i.e., negative force in -ev/A
Convergence criterion: Sigma_i[f(i)**2] < ACC
Interface with sub_conjugate is FUNCT(N,X,F,G)
Stress Controlled Conjugate Gradient method
work = Omega(tau:H^^(-T))
Last Modified: Aug. 5, 2007
Changed to the first PK-Stress Controlled Conjugate Gradient method
consistent with Si_SW CG code
Energy Minimization in the SXOH and H
Output SX0,SY0,SZ0
Last Modified: Dct. 13, 2008
Changed to the first PK-Stress Controlled Conjugate Gradient method
consistent with Si_SW CG code
Energy Minimization in the SXOH and H
Output SX0,SY0,SZ0
Last Modified: June 16, 2009

```
C********
c      INCLUDE 'vacancy_conjugate.f'
c      INCLUDE 'potential_vacancy_stress_new.f'
subroutine vacancy(E_coh)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
PARAMETER ( NPD = 499, NDOF = NPD*3+9)
PARAMETER ( EV_IN_J=1.60217733E-19 )
COMMON/POS/X0(NPD),Y0(NPD),Z0(NPD),NATOMTYPE(NPD)
COMMON/SIZE/NX,NY,NZ,NP,SIZE_X,SIZE_Y,SIZE_Z
COMMON/FILES/IFILE,II,III
COMMON/BOUND/NBOUNDARY(NPD)
COMMON/NABLSTA/RLIST,NTIMES,KSQRT
COMMON/NABLSTB/LIST(NPD*400),NABORS(NPD)
COMMON/NABLSTC/LISTBIG(NPD*400),NABORSBIG(NPD)
COMMON/FREE/NATOMFREE
COMMON/STRESS/S(NPD,6),VONMISES(NPD),N_STRESSFLAG
COMMON/conj_counter/i_conj
DIMENSION X(NDOF),G(NDOF),W(NDOF*6)
DIMENSION X00(NPD),Y00(NPD),Z00(NPD)
DIMENSION FX(NPD),FY(NPD),FZ(NPD)
DIMENSION SX0H(NPD),SY0H(NPD),SZ0H(NPD)
DIMENSION POS_RED/SX0(NPD),SY0(NPD),SZ0(NPD)
```
COMMON/POS_RED1/H(3,3),Hinv(3,3)
COMMON/stress_control/
H0(3,3),H0inv(3,3),H0inv_TRANS(3,3),H0Volume
COMMON/alpha/alpha

dimension pair1(3000),pair2(3000),pair_der2(3000)
dimension den1(3000),den2(3000),den_der2(3000)
dimension embed1(3000),embed2(3000),embed_der2(3000)
common/sp_pair/pair1,pair2,pair_der2
commom/sp_den/den1,den2,den_der2
common/sp_embed/embed1,embed2,embed_der2

COMMON/pot_fitting/a0,rcut
common/fit/c11,c12,c44,e_vacancy
C potential paramters
C Ni: Voter-Chen; E_coh = 4.45eV
c a0 = 3.51999998
c rcut = 4.7895
c E_coh = -4.44999981

CALL vINITPARAMETER
CALL vloadtable
C read in config
C CALL vINPUTCONFIG_PARA_FCC_111
CALL vINPUTCONFIG
C perfect crystal
SIZE_X = a0*5
SIZE_Y = a0*5
SIZE_Z = a0*5
C H of input config, same as H0
H(1,1) = Size_X
H(1,2) = 0
H(1,3) = 0
H(2,1) = 0
H(2,2) = Size_Y
H(2,3) = 0
H(3,1) = 0.
H(3,2) = 0.
H(3,3) = Size_Z

CALL vMATINV(H,Hinv,HVolume)

DO 1 I = 1,NPD
SX0(I) = Hinv(1,1)*X0(I) + Hinv(2,1)*Y0(I) + Hinv(3,1)*Z0(I)
SY0(I) = Hinv(1,2)*X0(I) + Hinv(2,2)*Y0(I) + Hinv(3,2)*Z0(I)
SZ0(I) = Hinv(1,3)*X0(I) + Hinv(2,3)*Y0(I) + Hinv(3,3)*Z0(I)
1 CONTINUE

c CALL vOUTPUTFILE
C stop

C############
C Update the ref state
C############
C
a224.cfg.xyz
Size_XH0 = Size_X
Size_YH0 = Size_Y
Size_ZH0 = Size_Z

H0(1,1) = Size_XH0
H0(1,2) = 0.
H0(1,3) = 0.
H0(2,1) = 0.
H0(2,2) = Size_YH0
H0(2,3) = 0.
H0(3,1) = 0.
H0(3,2) = 0.
H0(3,3) = Size_ZH0
C Set free moving atoms

CALL vMATINV(H0,H0inv,H0Volume)
CALL vMTRANS(H0inv,H0inv_TRANS)

DO 2 I = 1,NPD
  SX0H(I) = H0(1,1)*SX0(I) + H0(2,1)*SY0(I) + H0(3,1)*SZ0(I)
  SY0H(I) = H0(1,2)*SX0(I) + H0(2,2)*SY0(I) + H0(3,2)*SZ0(I)
  SZ0H(I) = H0(1,3)*SX0(I) + H0(2,3)*SY0(I) + H0(3,3)*SZ0(I)
2 CONTINUE

IDOF = 1
DO 10 I = 1,NPD
  X(IDOF) = SX0H(I)
  X(IDOF+1) = SY0H(I)
  X(IDOF+2) = SZ0H(I)
  IDOF = IDOF + 3

10 CONTINUE
CONTINUE

C rescale H by alpha

C alpha = (500.**(1/6.))
alpha = 1
X(IDOF) = H(1,1)*alpha
X(IDOF+1) = H(1,2)*alpha
X(IDOF+2) = H(1,3)*alpha
X(IDOF+3) = H(2,1)*alpha
X(IDOF+4) = H(2,2)*alpha
X(IDOF+5) = H(2,3)*alpha
X(IDOF+6) = H(3,1)*alpha
X(IDOF+7) = H(3,2)*alpha
X(IDOF+8) = H(3,3)*alpha

N = IDOF+8

WRITE(*,*) 'Total degree of freedom=',N
STOP

C Convergence and Initiation parameters for CG
MAXFN = 100
ACC = 0.005E-2
DFPRED = 0.01
i_conj = 0
N_STRESSFLAG = 0
CALL vZXCGRF77 (N,ACC,MAXFN,DFPRED,X,G,F,W,IER)

WRITE(*,*) 'If IER=0, Done; If IER=129, Line search fails'
WRITE(*,*) 'CG Search: IER = ',IER
WRITE(*,*)
'*------------------------------------------------------*
WRITE(*,*) 'Structural parameters of Cu at zero stress'
WRITE(*,*) 'Lattice constant (A): ', H(1,1)/5
WRITE(*,*) 'Energy per atom (eV): ', F/NPD
WRITE(*,*)
'*------------------------------------------------------*

C cohesive energy per atom
E_coh = -3.54
open (33, File='./vacancy/info.txt', status='unknown')
E_vacancy = F - E_coh * NPD
WRITE(*,*) 'Vacancy formation energy (eV): ', E_vacancy
WRITE(*,*) 'Mishin EAM1: 1.272eV'
C     atomic volume per atom in a perfect crystal
C     a0 = 3.615
aL0 = 5 * a0
V_atom_perfect = aL0**3/(NPD+1)
V_perfect = aL0**3/(NPD+1)*NPD
V_vacancy = H(1,1)*H(2,2)*H(3,3)
V_atom_ratio = (V_vacancy-V_perfect)/V_atom_perfect
C     write(*,*) 'Vacancy formation volume (eV): ', V_atom_ratio
C     write(*,*) 'Mishin EAM1: 0.701 V0'
C     write(*,*) a0, rcut
C     write(*,*)
C     '***************************************************'
C     write(*,*) 'Summary: relaxed volume for a vacancy'
C     write(*,*) 'Unrelaxed total volume: V_un = V_perfect/
500*499;'
C     write(*,*) 'Relaxed excess volume: V_relaxed - V_un =
0.7012 V0'
C     write(*,*) 'Agree with Mishin : 0.701 V0'
C     write(*,*)
C     '***************************************************'
C     CALL vOUTPUTFILE

END

C*******************
C     INITIALIZATION
C*******************
SUBROUTINE vINITPARAMETER
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
PARAMETER ( NPD = 499, NDOF = NPD*3+9)
PARAMETER ( WTMOL_O = 15.9994E-3, WTMOL_SI = 28.0855E-3)
PARAMETER ( WTMOL_H = 1.00794E-3)
PARAMETER ( AV0 = 6.0225E+23, BOLZ = 1.38054)
PARAMETER ( EPSI = 120., SIGMA = 3.405)
COMMON SIZE/NX,NY,NZ,NP,SIZE_X,SIZE_Y,SIZE_Z
COMMON NABLSTA/RLIST,NTIMES,KSQRT
COMMON FILES/IFILE,II,III
COMMON STRESS/S(NPD,6),VONMISES(NPD),N_STRESSFLAG
COMMON STRAIN/FM(NPD,9),DETF(NPD)
double precision rcut
COMMON TOTAL NUMBER OF ATOMS
WRITE(*,*) 'NX,NY,NZ = '
READ(*,*) NX,NY,NZ
nx = 24
ny = 30
nz = 20
NP = NPD

C     SET POTENTIAL CUT-OFF AND NEIGHBOR LIST CUT-OFF

NTIMES = 0
KSQRT = 100

C

C     SET FILE NUMBER
C

WRITE(*,*)
WRITE(*,*) 'NUMBER OF OUTPUT CONFIGURATION FILES:'
READ(*,*) NUMBERFILE
NUMBERFILE = 10

C

C     SET FILE NUMBER
C

IFILE = 0
II = 48
III = 48

C

C     SET ATOMIC MASS
C

AMASS_O = WTMOL_O/AVO
AMASS_SI = WTMOL_SI/AVO
AMASS_H = WTMOL_H/AVO

C

C     INIT STRESS, N_STRESSFLAG = 1, write color info
C

N_STRESSFLAG = 0
DO 10 I=1,NPD
  DO 20 J=1,6
    S(I,J) = 0.
  CONTINUE
  VONMISES(I) = 0.
10 CONTINUE

20 CONTINUE
SUBROUTINE vFUNCT(N,X,Gibbs,G)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)

PARAMETER ( NPD = 499, NDOF = NPD*3+9)

PARAMETER ( EV_IN_J=1.60217733E-19)
DIMENSION X(N),G(N)
DIMENSION FX(NPD),FY(NPD),FZ(NPD)
DIMENSION FXS(NPD),FYS(NPD),FZS(NPD)
DIMENSION X00(NPD),Y00(NPD),Z00(NPD)
DIMENSION SX0(H(NPD),SY0(H(NPD),SZ0(H(NPD))
DIMENSION H2(3,3),H2_TRANS(3,3),H3(3,3),DH(3,3)
DIMENSION H_TRANS(3,3),Hinv_TRANS(3,3),FH_Vir(3,3),FH_ext(3,3)
DIMENSION S_V(3,3),S_V_EVA(3,3)
DIMENSION TAU(3,3),TAU_EVA(3,3),TAU_N_EVA(3,3)

COMMON/POS/X0(NPD),Y0(NPD),Z0(NPD),NATOMTYPE(NPD)
COMMON/BOUN/NBOUNDARY(NPD)
COMMON/FILES/IFILE,II,III
COMMON/conj_counter/i_conj

COMMON/POS_RED/SX0(NPD),SY0(NPD),SZ0(NPD)
COMMON/POS_RED1/H(3,3),Hinv(3,3)

COMMON/STRESS/S(NPD,6),VONMISES(NPD),N_STRESSFLAG
COMMON/stress_control/
H0(3,3),H0inv(3,3),H0inv_TRANS(3,3),H0Volume
COMMON/stress_control2/AJacob
COMMON/alpha/alpha

C targeted stresses in GPa

TAU(1,1) = 0.
TAU(1,2) = 0
TAU(1,3) = 0
TAU(2,1) = 0.
TAU(2,2) = 0.
TAU(2,3) = 0
TAU(3,1) = 0.
TAU(3,2) = 0
TAU(3,3) = 0.

C DOF: *H0

IDOF = 1
DO 9 I = 1,NPD
SX0H(I) = X(IDOF)
SY0H(I) = X(IDOF+1)
SZ0H(I) = X(IDOF+2)
IDOF = IDOF + 3
9 CONTINUE

H(1,1) = X(IDOF)/alpha
H(1,2) = X(IDOF+1)/alpha
H(1,3) = X(IDOF+2)/alpha
H(2,1) = X(IDOF+3)/alpha
H(2,2) = X(IDOF+4)/alpha
H(2,3) = X(IDOF+5)/alpha
H(3,1) = X(IDOF+6)/alpha
H(3,2) = X(IDOF+7)/alpha
H(3,3) = X(IDOF+8)/alpha

DO 10 I = 1,NPD
SX0(I) = H0Inv(1,1)*SX0H(I)+H0Inv(2,1)*SY0H(I)
+H0Inv(3,1)*SZ0H(I)
SY0(I) = H0Inv(1,2)*SX0H(I)+H0Inv(2,2)*SY0H(I)
+H0Inv(3,2)*SZ0H(I)
SZ0(I) = H0Inv(1,3)*SX0H(I)+H0Inv(2,3)*SY0H(I)
+H0Inv(3,3)*SZ0H(I)
10 CONTINUE

DO 11 I = 1,NPD
X00(I) = H(1,1)*SX0(I) + H(2,1)*SY0(I) + H(3,1)*SZ0(I)
Y00(I) = H(1,2)*SX0(I) + H(2,2)*SY0(I) + H(3,2)*SZ0(I)
Z00(I) = H(1,3)*SX0(I) + H(2,3)*SY0(I) + H(3,3)*SZ0(I)
11 CONTINUE

C Hinv needs to be updated prior to calling for force calculation
CALL vMATINV(H,Hinv,HVolume)
CALL vEVALFORCE_COPPER(X00,Y00,Z00,SUMPOTE,FX,FY,FZ)
F = SUMPOTE
CALL vMTRANS(Hinv,Hinv_TRANS)
CALL vStress_Unit(TAU,TAU_EVA3)
CALL vMPROD(Hinv_TRANS,TAU_EVA3,H2)
CALL vMPROD(H0inv_TRANS,TAU_EVA3,H3)

DO 15 I = 1,3
  DO 15 J = 1,3
    DH(I,J) = H(I,J) - H0(I,J)
  CONTINUE

C     CALL vMPROD(DH,H2_TRANS,H3)
c     write(*,*) 'DH',H(1,1),H(2,2),H(3,3)
c     write(*,*) 'HVolume=',HVolume,H0Volume
C     write(*,*) 'TAU_EVA3=',TAU_EVA3(1,1)

work = 0.
DO 16 I = 1,3
  DO 16 J = 1,3
    work = work + DH(I,J)*H3(I,J)*H0Volume
  CONTINUE

Gibbs = F - Work

c     write(*,*) 'Gibbs energy in eV=',Gibbs
c     write(*,*) 'Work =',Work,'F energy =',F
C     stop
C     Eq.(20), forces on s

CALL vMTRANS(H,H_TRANS)
CALL vMPROD(H_TRANS,H0inv_TRANS,H3)

DO 13 I = 1,NPD
  FXS(I) = H3(1,1)*FX(I) + H3(2,1)*FY(I) + H3(3,1)*FZ(I)
  FYS(I) = H3(1,2)*FX(I) + H3(2,2)*FY(I) + H3(3,2)*FZ(I)
  FZS(I) = H3(1,3)*FX(I) + H3(2,3)*FY(I) + H3(3,3)*FZ(I)
13 CONTINUE

C     Eq.(21), forces on H
AJacob = HVolume/H0Volume

c     write(*,*) 'AJacob',AJacob

CALL vStress_Virial(S_V,HVolume)
CALL vStress_Unit(S_V,S_V_EVA3)

CALL vMPROD(Hinv_TRANS,S_V_EVA3,FH_Vir)
CALL vMPROD(H0inv_TRANS,TAU_EVA3,FH_Ext)

C     DO 14 I = 1,3
C     DO 14 J = 1,3
C     TAU_N_EVA3(I,J) = S_V_EVA3(I,J) - TAU_EVA3(I,J)
C14 CONTINUE
C     CALL vMPROD(Hinv_TRANS,TAU_N_EVA3,FH)
If (i_conj.gt.10) Then
   CALL vOUTPUTFILE
   i_conj = 0
else
   i_conj = i_conj + 1
Endif

********** change force to energy gradient
FACTOR = -1.

DO 20 I = 1,NPD
   G(IDOF) = FXS(I) * FACTOR
   G(IDOF+1) = FYS(I) * FACTOR
   G(IDOF+2) = FZS(I) * FACTOR
C** force unit for conjugate subroutine: ev/A

20 CONTINUE

FACTOR0 = 1/1.
FACTOR0 = 1/alpha
G(IDOF) =(FH_Vir(1,1) * HVolume - FH_Ext(1,1)*H0Volume)*
FACTOR0
G(IDOF+1)= (FH_Vir(1,2) * HVolume - FH_Ext(1,2)*H0Volume)*
FACTOR0
G(IDOF+2) = (FH_Vir(1,3) * HVolume - FH_Ext(1,3)*H0Volume)*
FACTOR0
G(IDOF+3) = (FH_Vir(2,1) * HVolume - FH_Ext(2,1)*H0Volume)*
FACTOR0
G(IDOF+4) = (FH_Vir(2,2) * HVolume - FH_Ext(2,2)*H0Volume)*
FACTOR0
G(IDOF+5) = (FH_Vir(2,3) * HVolume - FH_Ext(2,3)*H0Volume)*
FACTOR0
G(IDOF+6) = (FH_Vir(3,1) * HVolume - FH_Ext(3,1)*H0Volume)*
FACTOR0
G(IDOF+7) = (FH_Vir(3,2) * HVolume - FH_Ext(3,2)*H0Volume)*
FACTOR0
G(IDOF+8) = (FH_Vir(3,3) * HVolume - FH_Ext(3,3)*H0Volume)*
FACTOR0

F_t = 0
DO 1000 I = IDOF,IDOF+8
   F_t = F_t + G(I)**2
write(*,*) 'G(i)',FH_Vir(1,1),FH_Vir(1,1)* HVolume
C1000 CONTINUE
C     write(*,*) 'ftotal',G(1),G(5),G(8),sqrt(F_t)
C     write(*,*) 'HVolume',HVolume
C     write(*,*) 'H(1,1) in A',H(1,1),(H(1,1))/h0(1,1)
C     write(*,*) 'H(2,2) in A',H(2,2),(H(2,2))/h0(2,2)
C     write(*,*) 'H(3,3) in A',H(3,3),(H(3,3))/h0(3,3)
C     write(*,*) 'H(1,2) in A',H(1,2)
C     write(*,*) 'H(2,1) in A',H(2,1)
C     write(*,*) 'H(2,3) in A',H(2,3)
C     write(*,*) 'FH11 in eV/A',G(IDOF)
C     write(*,*) 'FH22 in eV/A',G(IDOF+4)
C     write(*,*) 'FH33 in eV/A',G(IDOF+8)
C     write(*,*)
CALL vMPROD(S_V,Hinv,H1)
CALL vMPROD(H1,H0,H2)
C     write(*,*) '--- PK1 stress ---'
C     write(*,*) 'S11, S22, S33 in GPa',
C &     H2(1,1)*AJacob,H2(2,2)*AJacob,H2(3,3)*AJacob
C     write(*,*) 'S12, S13, S23 in GPa',
C &     H2(1,2)*AJacob,H2(1,3)*AJacob,H2(2,3)*AJacob
C     write(*,*) 'S21, S31, S32 in GPa',
C &     H2(2,1)*AJacob,H2(3,1)*AJacob,H2(3,2)*AJacob
RETURN
END

C*******************
C      OUTPUT
C*******************

SUBROUTINE vOUTPUTFILE

CHARACTER FILENAME*18
COMMON/FILES/IFILE,II,III

FILENAME(1:11) = './vacancy/v'
FILENAME(15:18) = '.cfg'

IF(IFILE.EQ.10) THEN
  IFILE = 1
  II = II+1
  I = IFILE+47
IF(II.EQ.58) THEN

RETURN
END
II = 48
III = III + 1
ENDIF
ELSE
  IFILE = IFILE + 1
  I = IFILE + 47
ENDIF
FILENAME(12:12) = char(III)
FILENAME(13:13) = char(II)
FILENAME(14:14) = char(I)
LP = 30
OPEN(LP, FILE=FILENAME, STATUS='UNKNOWN')
CALL vOUTPUT(LP)
RETURN
END

SUBROUTINE vOUTPUT(LP)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
PARAMETER (NPD = 499, NDOF = NPD*3 + 9)
PARAMETER (NATOM_O = 8, NATOM_Ar = 18, NATOM_Al = 28)
COMMON/POS/X0(NPD), Y0(NPD), Z0(NPD), NATOMTYPE(NPD)
COMMON/BOUN/NBOUNDARY(NPD)
COMMON/SIZE/NX, NY, NZ, NP, SIZE_X, SIZE_Y, SIZE_Z
COMMON/STRESS/S(NPD, 6), VONMISES(NPD), N_STRESSFLAG
COMMON/POS_RED/SX0(NPD), SY0(NPD), SZ0(NPD)
COMMON/POS_RED1/H(3, 3), Hinv(3, 3)
DISPLAY = 1.
WRITE(LP, 101) NP
101 FORMAT('Number of particles =', I6)
WRITE(LP, 102) H(1, 1)
102 FORMAT('H0(1,1) =', E15.8, ' A')
WRITE(LP, 103) H(1, 2)
103 FORMAT('H0(1,2) =', E15.8, ' A')
WRITE(LP, 104) H(1, 3)
104 FORMAT('H0(1,3) =', E15.8, ' A')
WRITE(LP, 105) H(2, 1)
105 FORMAT('H0(2,1) =', E15.8, ' A')
WRITE(LP, 106) H(2, 2)
106 FORMAT('H0(2,2) =', E15.8, ' A')
WRITE(LP, 107) H(2, 3)
107 FORMAT('H0(2,3) =', E15.8, ' A')
WRITE(LP, 108) H(3, 1)
c 108   FORMAT('H0(3,1) = ',E15.8,' A')
c       WRITE(LP,109) H(3,2)
c 109   FORMAT('H0(3,2) = ',E15.8,' A')
c       WRITE(LP,111) H(3,3)
c 111   FORMAT('H0(3,3) = ',E15.8,' A')
c       WRITE(LP,120)
c 120   FORMAT('.NO VELOCITY.')
c       WRITE(LP,112)
c 112   FORMAT('entry_count = 6')
c       WRITE(LP,113)
c 113   FORMAT('auxiliary[0] = Mises stress [in GPa]')
c       WRITE(LP,114)
c 114   FORMAT('auxiliary[1] = S11 [in GPa]')
c       WRITE(LP,115)
c 115   FORMAT('auxiliary[2] = S12 [in GPa]')
c       WRITE(LP,121)
c 121   FORMAT('63.546')
c       WRITE(LP,123)
c 123   FORMAT('Ni')
c       DO 100 I=1,NP
          IF(NBOUNDARY(I).EQ.3) THEN
          ELSE
             WRITE(LP,76) SX0(I),SY0(I),SZ0(I),
            &                 VONMISES(I),S(I,1),S(I,6)
c 76           FORMAT(F15.10,F15.10,F15.10,F15.10,F15.10,F15.10)
c        ENDIF
 100   CONTINUE

c       CLOSE(LP)
c       RETURN

c       END

SUBROUTINE vMATINV(A,B,C)
IMPLICIT DOUBLE PRECISION(A-H,O-Z)
DIMENSION A(3,3),B(3,3)
D11=A(2,2)*A(3,3)-A(2,3)*A(3,2)
D22=A(3,3)*A(1,1)-A(3,1)*A(1,3)
D33=A(1,1)*A(2,2)-A(1,2)*A(2,1)
D12=A(2,3)*A(3,1)-A(2,1)*A(3,3)
D23=A(3,1)*A(1,2)-A(3,2)*A(1,1)
D31=A(1,2)*A(2,3)-A(1,3)*A(2,2)
D13=A(2,1)*A(3,2)-A(2,3)*A(3,1)
D21=A(3,2)*A(1,3)-A(1,2)*A(3,3)
D32=A(1,3)*A(2,1)-A(2,1)*A(1,1)
C=A(1,1)*D11+A(1,2)*D12+A(1,3)*D13
B(1,1)=D11/C
B(2,2)=D22/C
C(3,3)=D33/C
B(1,2)=D21/C
B(2,3)=D32/C
B(3,1)=D13/C
B(2,1)=D12/C
B(3,2)=D23/C
B(1,3)=D31/C
RETURN
END

C******************************************************************************

SUBROUTINE vMTRANS(A,ATRANS)
C
THIS SUBROUTINE CALCULATES THE TRANSPOSE OF AN 3 BY 3
C
MATRIX [A], AND PLACES THE RESULT IN ATRANS.
C******************************************************************************

IMPLICIT DOUBLE PRECISION(A-H,O-Z)
Dimension A(3,3),ATRANS(3,3)

DO 1 I=1,3
   DO 1 J=1,3
      ATRANS(J,I) = A(I,J)
1 CONTINUE

RETURN
END

C******************************************************************************

SUBROUTINE vMPROD(A,B,C)
C
THIS SUBROUTINE MULTIPLIES TWO 3 BY 3 MATRICES [A] AND [B],
C
AND PLACE THEIR PRODUCT IN MATRIX [C].
C******************************************************************************

IMPLICIT DOUBLE PRECISION(A-H,O-Z)
Dimension A(3,3),B(3,3),C(3,3)

DO 2 I = 1, 3
   DO 2 J = 1, 3
      C(I,J) = 0.D0
   DO 1 K = 1, 3
      C(I,J) = C(I,J) + A(I,K) * B(K,J)
1 CONTINUE
2 CONTINUE
SUBROUTINE vStress_Virial(S_V,HVolume)

IMPLICIT DOUBLE PRECISION(A-H,O-Z)
PARAMETER ( NPD = 499, NDOF = NPD*3+9)

COMMON/STRESS/S(NPD,6),VONMISES(NPD),N_STRESSFLAG
double precision a0, rcut
COMMON/pot_fitting/a0,rcut
C COMMON/stress_control2/AJacob
Dimension S_V(3,3)

c a0 = 3.52
V0 = (a0**3)/4.

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

DO 10 I = 1,NPD
   S_V(1,1) = S_V(1,1) + S(I,1)
   S_V(2,2) = S_V(2,2) + S(I,2)
   S_V(3,3) = S_V(3,3) + S(I,3)
   S_V(2,3) = S_V(2,3) + S(I,4)
   S_V(1,3) = S_V(1,3) + S(I,5)
   S_V(1,2) = S_V(1,2) + S(I,6)
10 CONTINUE

S_V(1,1) = S_V(1,1)*V0/Hvolume
S_V(2,2) = S_V(2,2)*V0/Hvolume
S_V(3,3) = S_V(3,3)*V0/Hvolume
S_V(2,3) = S_V(2,3)*V0/Hvolume
S_V(1,3) = S_V(1,3)*V0/Hvolume
S_V(1,2) = S_V(1,2)*V0/Hvolume

c write(*,*)
c write(*,*) '--- Virial stress ---'
c write(*,*)'S11, S22, S33 in GPa',S_V(1,1),S_V(2,2),S_V(3,3)
c write(*,*)'S12, S13, S23 in GPa',S_V(1,2),S_V(1,3),S_V(2,3)
S_V(3,2) = S_V(2,3)
S_V(3,1) = S_V(1,3)
S_V(2,1) = S_V(1,2)

c      write (*,*) a0, rcut

RETURN
END

SUBROUTINE vStress_Unit(S_GPA,S_V_EVA3)

IMPLICIT DOUBLE PRECISION(A-H,O-Z)
PARAMETER ( NPD = 499, NDOF = NPD*3+9)

Dimension S_GPA(3,3),S_V_EVA3(3,3)

USTRESS_IN_GPA = (((1.60217733e-19)/1e-30*1e-9)

DO 10 I = 1,3
DO 10 J = 1,3
   S_V_EVA3(I,J) = S_GPA(I,J)/USTRESS_IN_GPA
10   CONTINUE
C     change stress unit from in GPa to in eV/A^3

RETURN
END

SUBROUTINE vINPUTCONFIG

IMPLICIT DOUBLE PRECISION (A-H,O-Z)

PARAMETER ( NPD = 499, NDOF = NPD*3+9)

COMMON/POS/X0(NPD),Y0(NPD),Z0(NPD),NATOMTYPE(NPD)
COMMON/BOUN/NBOUNDARY(NPD)
COMMON/FREE/NATOMFREE
COMMON/SIZE/NX,NY,NZ,NP,SIZE_X,SIZE_Y,SIZE_Z

double precision rcut, a0
COMMON/pot_fitting/a0,rcut

character dummy*2

a0_cu = 3.615

OPEN(1,FILE='vacancy.xyz')

C     WRITE(*,*) 'NUM OF FREE ATOMS =',NATOMFREE
       NATOMFREE = 0
DO 10 I=1,NPD
   READ(1,*) dummy,X0(I),Y0(I),Z0(I)
   X0(I) = X0(I)/a0_cu*a0
   Y0(I) = Y0(I)/a0_cu*a0
   Z0(I) = Z0(I)/a0_cu*a0
   NATOMFREE = NATOMFREE + 1
10    CONTINUE
  c WRITE(*,*) 'NUM OF FREE ATOMS =',NATOMFREE
  
RETURN
END

H.41 Calculating the Magnitude of a Vector (vectormagnitude.f)

double precision function vectmagnitude(vect,length)
implicit none
C program variables
integer length
double precision vect(length)
call dot(length, vect, vect,vectmagnitude)
vectmagnitude=sqrt(vectmagnitude)
return
end
APPENDIX I

METAL POTENTIAL TESTS

I.1 Analyzing the Fit Sr From the Potential Fitting

```python
m=[]
sr=[]
f=open('potfit.txt','r')
for line in f:
    row=line.split()
    if row==[]: continue
    if row[0]=='fitting':
        if row[2]=='from':
            sr.append(float(row[7]))
        elif row[2]=='is':
            sr.append(float(row[3]))
        else:
            print 'unknown fitting command'
    elif row[0]=='function':
        if row[2]=='for':
            a=row[4].split(',
            a=int(a[0])
            b=row[6].split(',
            b=int(b[0])
            c=int(row[8])
            m.append(a+b+c)
        elif row[2]=='from':
            pass
    else:
        print 'unknown function command'

f.close()
# print 'm is', m
# print 'sr is', sr
from pylab import *
xlabel('number of knots')
ylabel('sum of residuals squared for the fitting properties')
xlim(18,25)
yscale('log')
plot(m,sr,'rh',markersize=9)
show()
```

I.2 Analyzing the Test Sr From the Potential Fitting

```python
m=[]
sr=[]
f=open('potfit.txt','r')
```
for line in f:
    row=line.split()
    if row==[]:continue
    if row[0]=='testing':
        if row[2]=='from':
            sr.append(float(row[7]))
        else:
            print 'unknown testing command'
    elif row[0]=='First':
        if row[3]=='is':
            sr.append(float(row[4]))
        else:
            print 'unknown testing command'
    elif row[0]=='function':
        if row[2]=='for':
            a=row[4].split(',,
            a=int(a[0])
            b=row[6].split(',,
            b=int(b[0])
            c=int(row[8])
            m.append(a+b+c)
        elif row[2]=='from':
            pass
        else:
            print 'unknown function command'
f.close()
print 'm is', m
print 'sr is', sr
from pylab import *
xlabel('number of knots')
ylabel('sum of residuals squared for the testing properties')
xlim(18,25)
yscale('log')
plot(m,sr,'rh',markersize=9)
show()

I.3 Calculating the Lattice Temperature Relationship

#!/usr/bin/env python
#
#   texp.py
#
#   read in improtant information from ./controls/control.txt
#   reads in important property information from ./database/\[element names]\materialconst.txt
#   writes a file named in.texp
#   Run Lammmps using in.texp
#   Use two different log files to calculate thermal expansion coefficient
# write the thermal expansion coefficient to ans.txt in this folder

def lammpsvolcalc(potential, temperatures, latticesize, bulkmod, element='', pflag=''):  
    # calculate the berendson pressure control parameter using the bulk modulus
    cbulk=1.383e11  # copper bulk modulus
    cberendson=100  # copper berendson pressure control parameter
    berendson=bulkmod*cberendson/cbulk
    import os
    for temperature in temperatures:  # loop over all temperatures
        f = open('in.texp', 'w')
        if temperature==0:
            f.write('# simulation for finding the lattice constant and cohesive energy of nial
# read in data from nial.dat and equilibrate structure under 0 pressure conditions

boundary p p p
units metal
atom_style atomic
dimension 3
variable x index 1
variable y index 1
variable z index 1
variable xx equal 20*$x
variable yy equal 20*$y
variable zz equal 20*$z
dimension 3
boundary p p p
units metal
atom_style atomic''')

        f.write('
\lattice {0}\n'.format(latticesize))

        f.write('''region box block 0 ${xx} 0 ${yy} 0 ${zz} create_box 1 box create_atoms 1 box''')

        if pflag=='alloy':
            f.write('
\pair_style eam/alloy
\pair_coeff * * {0}
\pair_coeff * * {1}'''.format(potential, element))

475
else:
    f.write(\"pair_style eam\n\rf.write(\"pair_coeff * * \n{0}\n\'.format(potential))
    
    f.write(\"neighbor 1.0 bin
neigh_modify every 1 delay 5 check yes

thermo 1
fix 1 all box/relax iso 0
min_modify line quadratic
thermo_style custom step c_1_temp c_1_press vol\"
    f.write(\"log \n{0}\n\'.format(temperature))
    f.write(\"minimize 2e-11 1.0e-6 200 1000\"

else:
    f.write(\"# stablize pressure and temperature
at an increased temperature

variable x index 1
variable y index 1
variable z index 1

variable xx equal 20*$x
variable yy equal 20*$y
variable zz equal 20*$z

dimension 3
boundary p p p
units metal
atom_style atomic\"
    
    f.write(\"\nlattice fcc \n{0}\n\'.format(latticesize))
    
    f.write(\"region box block 0 ${xx} 0 ${yy} 0 ${zz}
create_box 1 box
create_atoms 1 box\"

    if pflag==\'alloy\':
        f.write(\"pair_style eam/alloy\n        f.write(\"pair_coeff * * \{0\n{1}\n\'.format(potential,element))
    else:
        f.write(\"pair_style eam\n        f.write(\"pair_coeff * * \n{0}\n\'.format(potential))
```python
f.write('variable t index {0}'.format(temperature))

f.write('''velocity all create $t 376847 loop geom
neighbor 1.0 bin
neigh_modify every 1 delay 5 check yes
timestep 0.001
#timestep 0.005
thermo 10
fix myfix all nve
fix mytfix all temp/berendsen $t $t .1'''
)

f.write('''run
fix mypfix all press/berendsen iso 1 1 {0}'.format(berendson))

f.write('''run 5000
# keep temparture and pressure the same and record thermo data in new log for the purpose of calculating thermal expansion.
unfix myfix
unfix mytfix
unfix mypfix
fix 1 all npt temp $t $t .1 iso 1 1 50 drag 2
thermo_style custom step c_1_temp c_1_press vol'''
)

f.write('''run 10000'''
)

f.close()

os.system('mpirun -np 4 ../src/lmp_openmpi<in.texp')

def volcalc(temperatures):
    tempav=[]
    latav=[]
    for temperature in temperatures:
        f=open('log.{0}'.format(temperature),'r')
        Temp=[] #initialize temperature list
        Lat=[] # initialize volume List
        for line in f:
            row=line.split()
            try:
                int(row[0]) # Checks to make sure line in file is a data line and not a text line
            except ValueError:
                if row[0]=='Loop': break #ends data acquisition from file
```
else: continue  # otherwise continues reading the file
else:  #acquires important data
    Temp.append(float(row[1]))
    Lat.append(float(row[3])**((1.0/3.0)/20.0))

if temperature == 0:
    tempav.append(0)
    latav.append(Lat[len(Lat)-1])
else:
    tempav.append(sum(Temp)/len(Temp))
    latav.append(sum(Lat)/len(Lat))
# calculates the average temperature and volume at 293k
f.close()
return tempav, latav

def datawrite(potential, x, y):
    f = open('data{0}'.format(potential), 'w')
    for i in range(len(x)):
        f.write('{0} {1}
'.format(x[i], y[i]))
    f.close()

def cpread(element):
    f = open(element + '.cp', 'r')
    T = []
    cp = []
    for line in f:
        row = line.split()
        T.append(row[0])
        cp.append(row[1])
    f.close()
    n = len(T)
    return T, cp, n

# lattice relationship 1
a0, p1 = 3.5146706441739, 0.00000240053795439774
T, cp, n = cpread('ni')
from pylab import *
T = array(T)
cp = array(cp)
import spline
b, c, d = spline.natspline(T, cp, n)

478
for x in xarray:  # for this array can't use data write.
    E = spline.splintegator(T, cp, n, b, c, d, x0, x)
    lat = a0 * math.exp(p1 * E)  # p1 is a fitted parameter for this equation
    f.write('{0} {1}\n'.format(x, lat))
    latarray.append(lat)
f.close()
plot(xarray, latarray, label='lattice relationship 1')

# lattice relationship 2 (use k)
lattarray2 = []
f = open('lattempni2.txt', 'w')
k0 = 3.4903
alpha = 2.518e-2
beta = 344.61
for x in xarray:  # for this array can't use data write.  
    # need to solve (at-k0)^2/alpha^2 - T^2/beta^2 = 1
    lat = sqrt(alpha**2 * (1 + x**2 / beta**2)) + k0
    f.write('{0} {1}\n'.format(x, lat))
    lattarray2.append(lat)
fx.close()
plot(xarray, lattarray2, 'k--', label='cryogenics June 1977')

# temperatures for md potentials
temperatures = [0, 25, 50, 75, 100, 150, 200, 250, 293]

# daw_baskes_ni.eam #Potential Currently Does not work do to unit problems

# Ni_u3.eam
lammppsvolcalc('Ni_u3.eam', temperatures, 3.5240, 1.903e11)
tempav, latav = volcalc(temperatures)
datawrite('Ni_u3.eam', tempav, latav)
plot(tempav, latav, 'ro', label='Ni_u3.eam', markersize=9)  # use custom marker size 9

# Ni_smf7.eam
lammppsvolcalc('Ni_smf7.eam', temperatures, 3.5240, 1.903e11)
tempav, latav = volcalc(temperatures)
datawrite('Ni_smf7.eam', tempav, latav)
plot(tempav, latav, 'g^', label='Ni_smf7.eam', markersize=9)  # use custom marker size 9

# Ni99.eam.alloy
lammppsvolcalc('Ni99.eam.alloy', temperatures, 3.5240, 1.903e11, 'Ni', 'alloy')
tempav, latav = volcalc(temperatures)
datawrite('Ni99.eam.alloy', tempav, latav)
plot(tempav, latav, 'mp', label='Ni99.eam.alloy', markersize=9)
I.4 Calculating the Young’s Modulus Temperature Relationship

```python
#!/usr/bin/env python
#

def lammpsstresscalc(potential, temperatures, latticesize, bulkmod, element='', pflag=''):  
    # calculate the berendson pressure control parameter using the bulk modulus
    cbulk=1.383e11  #copper bulk modulus
    cberendson=100  #copper berendson pressure control parameter
    berendson=bulkmod*cberendson/cbulk

    import os
    for temperature in temperatures:  
        #loop over all temperatures
        if temperature==0:  
            continue  #As far as I know its impossible to do a tensile test in lammps at 0k
```
# so skip to the next temperature in temperatures

\textbf{else:}

\begin{verbatim}
f=open('in.tensile','w') #file for running tensile test
f.write('''# stablize pressure and temperature at an increased temperature

variable x index 1
variable y index 1
variable z index 1

variable xx equal 10*$x
variable yy equal 10*$y
variable zz equal 10*$z

dimension 3
boundary p p p
units metal
atom_style atomic'''
)

f.write('''\nattice fcc
{}
'''.format(latticesize))

f.write('''region box block 0 ${xx} 0 ${yy} 0 ${zz}
create_box 1 box
create_atoms 1 box'''
)

\end{verbatim}

\begin{verbatim}
if pflag=='alloy':
f.write('''\npair_style eam/alloy''
f.write('''\npair_coeff *
{}'''.format(potential,element))

\end{verbatim}

\begin{verbatim}
else:
f.write('''\npair_style eam'''
f.write('''\npair_coeff *
{}'''.format(potential))

f.write('''\nnvariable t index
{}
'''.format(temperature))

f.write('''\nvelocity all create $t 376847 loop
''
geom
neighbor 1.0 bin
neigh_modify every 1 delay 5 check yes
timestep 0.001 thermo 10
\end{verbatim}
fix 1 all npt temp $t$ $t$ 1 iso 0 0 1 drag 2
run 30000
unfix 1

# store final cell length for strain calculations
variable tmp equal "lx"
variable LO equal ${tmp}
print "Initial Length, LO: ${LO}"

fix 1 all npt temp $t$ $t$ 1 y 0 0 1 z 0 0 1 drag 2
variable srate equal 1.5e9
variable srate1 equal "v_srate / 1.0e12"
fix 2 all deform 1 x erate ${srate1} units box remap x
variable strain equal "(lx-v_LO)/v_LO"
variable p1 equal "-pxx/10000"
thermo_style custom step temp v_strain v_p1 press
run 1000

unfix 2''
f.write('\nlog
{0}\n'.format(temperature))
f.write('run 10000'
)
f.close()
os.system('mpirun -np 4 ../src/lmp_openmpi<in.tensile')

def stresscalc(temperatures):
    youngmod=[]
    for temperature in temperatures:
        f=open('log.{0}'.format(temperature),'r')
        strain=[] #initialize temperature list
        stress=[] # initialize volume List
        for line in f:
            try:
                row=line.split()
                int(row[0]) # Checks to make sure line in file is a data line and not a text line
            except ValueError:
                if row[0]=='Loop': break #ends data acquisition from file
            else: continue # otherwise continues reading the file
            else: #acquires important data
                strain.append(float(row[2]))
                stress.append(float(row[3]))
        f.close()
        # try calculating youngmodulus using 1000 step data.
stepsize=10
guessedmod=0.0
for steps in range(10000,0,-stepsize):
    i=steps/stepsize
guessedmod+=stress[i-1]/strain[i-1]  #guessed
youngmod
    num=float(10000/stepsize)
youngmod.append(guessedmod/num)
return youngmod
def stressexam(temperatures):
    for temperature in temperatures:
        f=open('log.{0}'.format(temperature),'r')
        strain=[]  #initialize temperature list
        stress=[]  # initialize stress list
        for line in f:
            row=line.split()
            try:
                int(row[0])  # Checks to make sure line in
file is a data line and not a text line
            except ValueError:
                if row[0]=='Loop': break  #ends data
                else: continue  # otherwise
                else: acquire important data
                    strain.append(float(row[2]))
                    stress.append(float(row[3]))
        f.close()
    return strain,stress
def datawrite(potential,x,y):
    f=open('modulusdata{0}'.format(potential),'w')
    for i in range(len(x)):
        f.write('{0} {1}
'.format(x[i],y[i]))
    f.close()

def dataread(potential):
    f=open('modulusdata{0}'.format(potential),'r')
    temperatures=[]
youngmod=[]
    for line in f:
        row=line.split()
        temperatures.append(float(row[0]))
youngmod.append(float(row[1]))
    return temperatures,youngmod
def elasticread(element):
    f=open(element+'.elastic','r')
T = []
c11 = []
c12 = []
c44 = []
for line in f:
    row = line.split()
    T.append(row[0])
c11.append(float(row[1]) * 100)  # converts from 10^11 pa to Gpa
    c12.append(float(row[2]) * 100)  # converts from 10^11 pa to Gpa
    c44.append(float(row[3]) * 100)  # converts from 10^11 pa to Gpa
f.close()
return T, c11, c12, c44

# youngs modulus
T, c11, c12, c44 = elasticread('ni')
from pylab import *
youngarray = []
f = open('youngmodni.txt', 'w')
for i in range(len(T)):
    # calculate young modulus from elastic constants c11, c12, c44
    # youngsmodule = (c44*(c11+2*c12))/(c12+c44)
    youngarray.append((c11[i] - 2 * c12[i] ** 2) / (c11[i] + c12[i]))
    f.write('{0} {1}\n'.format(T[i], youngarray[i]))
f.close()
plot(T, youngarray, label='""Young\'s modulus Ni""')

# temperatures for md potentials
temperatures = [0.5, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 293]
# temperatures = [293]
# daw_baskes_ni.eam # Potential Currently Does not work do to unit problems

# Ni_u3.eam
lammmpsstresscalc('Ni_u3.eam', temperatures, 3.5240, 1.903e11)
youngmod = stresscalc(temperatures)
datawrite('Ni_u3.eam', temperatures, youngmod)
# temperatures, youngmod = dataread('Ni_u3.eam')
plot(temperatures, youngmod, 'ro', label='Ni_u3.eam', markersize=9)
# use custom marker size 9

# Ni_smf7.eam
lammmpsstresscalc('Ni_smf7.eam', temperatures, 3.5240, 1.903e11)
youngmod = stresscalc(temperatures)
datawrite('Ni_smf7.eam', temperatures, youngmod)
# temperatures, youngmod = dataread('Ni_smf7.eam')
plot(temperatures,youngmod,'g^',label='Ni_smf7.eam',markersize=9)  
#use custom marker size 9

# Ni99.eam.alloy
lammpsstresscalc('Ni99.eam.alloy',temperatures,3.5240,1.903e11,'Ni','alloy')
youngmod=stresscalc(temperatures)
datawrite('Ni99.eam.alloy',temperatures,youngmod)
#temperatures,youngmod=dataread('Ni99.eam.alloy')
plot(temperatures,youngmod,'mp',label='Ni99.eam.alloy',markersize=9)

# NiAl.eam.alloy
lammpsstresscalc('NiAl.eam.alloy',temperatures,3.5240,1.903e11,'Ni','alloy')
youngmod=stresscalc(temperatures)
datawrite('NiAl.eam.alloy',temperatures,youngmod)
#temperatures,youngmod=dataread('NiAl.eam.alloy')
plot(temperatures,youngmod,'bh',label='NiAl.eam.alloy',markersize=9)

# yuNi.eam.alloy
lammpsstresscalc('yuNi.eam.alloy',temperatures,3.5240,1.903e11,'Ni','alloy')
youngmod=stresscalc(temperatures)
datawrite('yuNi.eam.alloy',temperatures,youngmod)
#temperatures,youngmod=dataread('yuNi.eam.alloy')
plot(temperatures,youngmod,'cs',label='yuNi.eam.alloy',markersize=9)

# niK.eam.alloy
lammpsstresscalc('niK.eam.alloy',temperatures,3.5147,1.903e11,'Ni','alloy')
youngmod=stresscalc(temperatures)
datawrite('niK.eam.alloy',temperatures,youngmod)
#temperatures,youngmod=dataread('niK.eam.alloy')
#strain,stress=stressexam(temperatures)
#plot(strain,stress,'k-',label='Kraus_ni')
plot(temperatures,youngmod,'kD',label='Kraus_ni',markersize=9)

# graphing variables
xlabel('temperature (k)')
ylabel("Young's Modulus (GPa)"")
xticks(arange(0,325,25))
yticks(arange(20,380,20))
xlim(-7,300)
ylim(55,360)
legend(loc=2)
#savefig('latticerelationship.pdf')
show()
J.1 Introduction

This algorithm was developed to allow controlled bonding of polymers to ceramic nano particles for LAMMPS. The software uses an object oriented approach where the key classes are “Lmpsdata”, “Lmpsmolecule”, and “particlesurface”. The Lmpsdata class reads, writes, extracts, and stores all of the data required for LAMMPS to run simulations. The Lmpsmolecule class only stores molecular style data which is used for manipulating molecules for bonding. The “particlesurface” class stores the data of the nanoparticle surface that is interacting/bonding with the polymer. These classes interact in such a way to allow the user to read and write data files, and implement reactions between the particle surface and the polymer chains.

J.2 Current Version: Version 1.0

Capabilities:
1. Bonding ceramic nano particles to the polymer matrix
2. Inserting nano particles into openings in the polymer matrix
3. Calculating the density of spherical shells centered around the origin
4. Creating xyz files for VMD
5. Create a polymer system with a specific molecular weight distribution (untested)

Limitations:
1. For large systems, code can run extremely slowly
2. Python’s multiprocessing has not been tested on distributed memory machines
3. Designed to only handle one nano particle which is in the center of the simulation box

J.3 Terminology

Most of the terminology used below is explained in the LAMMPS manual or doc folder under the “read_data” command. The rest of the terminology comes from object oriented coding descriptions and procedural coding descriptions.

J.4 Sample Scripts

J.4.1 Example 1: Inserting a Nano Particle

In this example, a nano particle is inserted into a spherical whole in the polymer box. Also, the density is calculated, which illustrates the nano particle was successfully inserted. The velocities were deleted because the nano particle has no velocity and LAMMPS wont let you have a data file where only some of the atoms have velocities. In this case, the pair coefficients were also deleted because LAMMPS will not let you read in pair coefficients if the pair style is set to hybrid or hybrid/overlay.

```python
import lmpsdata
from pylab import *

# Read in the alumina nanoparticle
f=open('alumina.pmma80','r')
nanoparticle=[]
for line in f:
    row=line.split()
    nanoparticle.append(row)

# Read in the polymer datafile
data=lmpsdata.Lmpsdata('pmma80data.finaeq3','full')

# Add in the mass information for the atom type 7, and 8
# 7 is aluminum cation and 8 is oxygen anion
massinfo=[['7','26.981539'],['8','15.9994']]
```
data.adddata(massinfo, 'Masses')

# Add in the alumina nanoparticle
data.addatoms(nanoparticle, False)

# Delete the data for the velocities and pair coefs
data.deletebodydata('Velocities')
data.deletebodydata('Pair Coeffs')

# Delete Velocities and Pair Coeffs from the keywords
data.keywords.remove('Velocities')
data.keywords.remove('Pair Coeffs')

# Write the density calculations to the file testden.txt
r, rho = data.density(.5, 0, 20)
plot(r, rho)
xlabel('distance (angstrom)')
ylabel('density (amu/angstrom^3)')
#legend(loc=1)
show()

# Write the new nanocomposite into a new datafile
data.write('pmma80_nano_composite_data.initial', 1)

J.4.2 Example 2: Bonding of PMMA to a alumina nano particle.

In this example PMMA is bonded to an alumina nano particle with this reaction:

Before this script can be run, the polymer needs to be near or at equilibrium. The starting bonding distance used in this example was the cation-cation distance for alpha aluminum. This value can be increased by an angstrom or two to form the number of bonds required. In this script the value is increased by about an angstrom. For brevity purposes, I only included one of the three bonding cases shown in the top portion of this example.

```python
import lmpsdata, copy
data = lmpsdata.Lmpsdata('pmma80compositedata.init', 'full')
#copies pmma80compositedata.init to a new folder
directory='./bulk/'
data.write(directory+'pmma80_composite_data.initial', 0)
```
#orders atomdata before doing bonding procedure.
data.atomorder()

#seperate nanocomposite atoms into polymer and nanoparticle portion
# Note:Only the atom structures will be present in these variables.
polymer=lmpsdata.molecules(data,1,5)
nanoparticle=lmpsdata.molecules(data,6,6,'atom')

#seperate nanoparticle into its particle surface
surface=lmpsdata.particlesurface(nanoparticle[0], 1.94, 8, 'full')
surface.createxyz('alumina_surface_initeq.xyz',data)

#setup copies of molecules for different reaction processes
crosslink=copy.deepcopy(polymer) #crosslink case is 2 bonds formed
weakbond=copy.deepcopy(polymer) #weakbond case is 4 bonds formed
strongbond=copy.deepcopy(polymer) #strongbonds case is 8 bonds formed

#setup copies of surface for different reaction processes
surface_crosslink=copy.deepcopy(surface)
surface_weakbond=copy.deepcopy(surface)
surface_strongbond=copy.deepcopy(surface)

#add mass data for the bonded carbonyl type
massinfo=[['9','15.9994']]
data.adddata(massinfo,'Masses')

#setup copies of data for different reaction processes
data_crosslink=copy.deepcopy(data)
data_weakbond=copy.deepcopy(data)
data_strongbond=copy.deepcopy(data)

#find possible bonding between crosslink(polymer) and surface_crosslink
bondinglen=0
for molecule in crosslink:
    molecule.findparticlebondingpoints(surface_crosslink,6,3.2,2)
    #need to ensure the number of bonds are in multiples of 2
    if len(molecule.bondinginformation)%2:
        i=len(molecule.bondinginformation)-1
        del molecule.bondinginformation[i]
        print 'the length of the bonding information is',
        len(molecule.bondinginformation)
bondinglen+=len(molecule.bondinginformation)
bondinglen=bondinglen/float(len(crosslink))

#Bond crosslink(polymer) to surface_crosslink
count=1
for molecule in crosslink:
    print 'the iteration is', count
    molecule.bondtoparticle(surface_crosslink,1,'9','-.7825')
    #add 1 atom to surface_crosslink per 2 atoms bonded
    for j in range(len(molecule.bondinginformation)/2):
        surface_crosslink.addatom(8,-.945,5)
        count+=1

# extract the crosslink surface
surface_crosslink.extractparticle()

# extract molecule informtaion to data_crosslink
data_crosslink.extractmolecules(crosslink)

# add in the crosslinked nanoparticle to data_crosslink
data_crosslink.addatoms(surface_crosslink.particle)

# delete the data for the velocities and pair coeffs from data_crosslink
data_crosslink.deletebodydata('Velocities')

# delete the Velocities and Pair Coeffs from the keywords from data_crosslink
data_crosslink.keywords.remove('Velocities')

# write the crosslink bonded nanocomposite into a new data file
directory='./crosslink/
data_crosslink.write(directory+'pmma80_composite_data.initial',1)
# write the crosslink bondinglen into a file in the crosslink directory
f=open(directory+'bondinglen.info','w')
f.write('{0}'.format(bondinglen))
f.close()

J.5 Methods by class

Notes: Not all of the methods are included in this guide. The methods I have left out are meant to be private methods, but as all methods in python are public, they can still be accessed. Therefore, to avoid people including them in Python scripts and possibly causing unintended behavior, I will just avoid placing them in the guide.
All methods using atom information are written assuming the image flags are included. If you don’t use the image flags, this software may crash.

**J.5.1 Lmpsdata(file,atomtype)**

Initiates the class and reads from “file” which is a LAMMPS data file. All information from the read data file is stored in this class. In order to utilize the atom information later, the atom style must be assigned from “atomtype”.

**J.5.2 write(file, modflag)**

Writes the information stored in this class out to a LAMMPS data file. The modflag allows the user to control whether certain header data will be calculated or written to the data file as is. The modflag set to “0” accesses the portion of the method which writes out all information to the data file unchanged, but the modflag set to “1” accesses the portion of the method which calculates certain header data. The header data that can be calculated are the number of atoms, bonds, angles, dihedrals, impropers and their number of types.

**J.5.3 atomorder()**

This method organizes the atoms by atom id from least to greatest and allows the bonding algorithm to always attempt to bond the first and last bondable atom on the molecule.

**J.5.4 addatoms(atoms,retlist=false)**

This method adds “atoms” to the atoms already stored in the “Lmpsdata” class. If “retlist” is set to “false” the method returns nothing. If “retlist” is set to “true” the method returns a list of the modified atom-ids of the added atoms.

**J.5.5 adddata(data, keyword)**

This method adds “data” to the information already stored in the “Lmpsdata” class. The specific structure the data is added to is determined by the “keyword” given. Only a body keyword input will result in data being added.

**J.5.6 deletebodydata(keyword)**
This method empties the corresponding structure of the “keyword”. The “keyword” must be a body keyword for the method to work.

**J.5.7 extractmolecules(molecule)**

Extracts the information contained in “molecule” to this class. The input, “molecule”, is a list of “Lmpsmolecule” objects. This method should be used with care, because the method empties the “Lmpsdata” class’s structures which correspond to the keywords stored in the first “Lmpsmolecule” object. Note, if the keywords in the “Lmpsmolecule” objects are different, this may cause this method to not function properly. If the program requires multiple lists of “Lmpsmolecule” objects, only the first list being added to this class can use this method. The rest of the lists must be added manually using the “addatoms” and “adddata” methods. Otherwise, important information will be lost from the class’s structures.

**J.5.8 density(ringsize, init, final, file=’ ’)**

Creates spherical shells from the “init” radius to the “final” radius. The thickness of these shells are defined by the “ringsize”. The method then calculates the density in each shell. The resulting list of radii and densities are either returned to the calling script or printed to a file. The default option is to return the values to the calling script. To print the values to a file, set the desired destination with the “file” variable.

**J.5.9 createxyz(file, routine=’mass’, values=None)**

Stores the atom information from the class in a xyz formatted file specified by “file”. This format is readable by VMD. There are two different algorithms used by “createxyz” for this conversion. One uses the masses of the atoms and the other uses the atomtype of the atoms. The mass method is the default method. In this method the mass of the atom is converted to its corresponding element number in the code. Currently, the code only contains this conversion for carbon, oxygen and aluminum. The masses used for these materials are formatted to four decimal points and the atom’s mass must exactly match in order for a
conversion to occur. The atomtype method can be used by setting routine to ‘atomtype’ and values to a list of element numbers. The atomtype is used to access the list at the index atomtype-1.

**J.5.10 particlesurface (particle, cutoff, atomid, atomtype, shape='sphere')**

Initiates the class and stores the “particle”’s atoms. Then produces the particle’s surface using the “cutoff”, “atomid”, “atomtype”, and “shape”. Currently, only one shape of the nano particle is supported, a sphere. The current implementation assumes the nano particle is centered around the origin and finds the atom with atomtype matching “atomid” which is the maximum distance away from the center. The algorithm than fills all atoms with atomtype matching “atomid” which are the “cutoff” distance away from the maximum distance. The “atomtype” input corresponds with the atomstyle which is required for all distance calculations and atom manipulations.

**J.5.11 addatom(atomtype, charge=None, moleculenum=None)**

This method adds an atom to the surface of the nano particle between the cutoff distance and the maximum distance. This also adds an atom to the particle’s atoms stored in this class. The “atomtype” corresponds to the atomtype of the added atom. The “charge” and “moleculenum” have the default setting of None. The “moleculenum” corresponds with the molecule number. If the atom style of this class requires either a charge or a molecule number and the default settings are used, undefined behavior may occur because the current code does not check if these variables are using the default. This method also adds image flags automatically at the end of the atom information. These flags are all set to zero.

**J.5.12 extractparticle()**

This method removes atoms from the particle surface which have been marked as being replaced during the bonding process. These atoms are also removed from the particle’s atoms stored in the class.
J.5.13 createxyz(file, data, routine='mass', values=None)

Stores the surface atom information from the class in a xyz formatted file specified by “file”. This format is readable by VMD. There are two different algorithms used by “createxyz” for this conversion. One uses the masses of the atoms and the other uses the atomtype of the atoms. The mass method is the default method. In this method the mass of the atom is converted to its corresponding element number in the code. Currently, the code only contains this conversion for carbon, oxygen and aluminum. The masses used for these materials go out to four decimal points and the atom’s mass must exactly match in order for a conversion to occur. The atomtype method can be used by setting routine to ‘atomtype’ and values to a list of element numbers. The atomtype is used to access the list at the index atomtype-1. The “data” input is a “Lmpsdata” class which is required for the mass algorithm to work. But even if your not using that algorithm the input is still required. Note: if you are trying to create a xyz file after the surface has bonded, the particle will need to be extracted than reinserted into the “particlesurface” class. Afterwards, the “createxyz” method can be run.

J.5.14 Lmpsmolecule (moleculenum, data, method)

Initiates the class and stores the molecular information from “data”. The input “data” is a “Lmpsdata” class. The molecular information consists of atoms, angles, bonds, dihedrals, impropers and velocities. There are two “methods”; the default method incorporates all the molecular data. The other method incorporates only the atom data and can be accessed by setting method to ‘atom’.

J.5.15 deleteatoms(atomnumbers, atomid)

Algorithm finds all atoms in the “Lmpsmolecule” class bonded to “atomnumbers” in the direction of “atomid”. The “atomnumbers” is a list of atom ids in the “Lmpsmolecule” class where the nano particle surface is forming bonds. The “atomid” can either be a list of atom id values or a single atom type. The “atomid” is used to find the direction of deletion for the algorithm. This
algorithm can be used to create a polymer system with a specific molecular weight distribution; though, this use for the algorithm has not been tested.

J.5.16 findparticlebondingpoints(particle, atomid, cutoffdistance, bondnumber)

This method finds and stores the bonding points between the “Lmpsmolecule” object and the “particle”, a “particlesurface” object. Around the particle is a bonding region where atoms in the molecule with the correct atom type, “atomid”, can form bonds with the nano particle surface. The thickness of this region is defined by the “cutoffdistance”. The max number of possible bonds formed is defined by the “bondnumber”; though, this number may not be reached.

J.5.17 bondtoparticle(particle, atomid, newid, newcharge)

This method bonds the molecule to the particle surface using the previously found bonding points. The “particle” is a “particlesurface” object. The “atomid” can either be a list of atom id values or a single atom type. The method changes the bonding atom in the molecule to have “newid” as the atomtype and “newcharge” as the charge. The method than uses the “atomid” to run the “deleteatoms” method. Finally, the method communicates with the “particle” which atoms on the particle surface need removing.

J.5.18 createxyz(file, data, routine=‘mass’, values=None)

Stores the surface atom information from the class in a xyz formatted file specified by “file”. This format is readable by VMD. There are two different algorithms used by “createxyz” for this conversion. One uses the masses of the atoms and the other uses the atomtype of the atoms. The mass method is the default method. In this method the mass of the atom is converted to its corresponding element number in the code. Currently, the code only contains this conversion for carbon, oxygen and aluminum. The masses used for these materials go out to four decimal points and the atom’s mass must exactly match
in order for a conversion to occur. The atomtype method can be used by setting routine to ‘atomtype’ and values to a list of element numbers. The atomtype is used to access the list at the index atomtype-1. The “data” input is a “Lmpsdata” class which is required for the mass algorithm to work. But even if your not using that algorithm the input is still required.

J.6 Functions

**J.6.1 molecules(data, init, final, processors, method='all')**

Builds a list of “Lmpsmolecule” classes using Python’s multiprocessing module. The “Lmpsmolecule” classes begin with the molecular number “init” and end with the molecular number “final”. The “Lmpsdata” which is passed to the “Lmpsmolecule” classes come from the input “data”. The “processors” tells the function how many processes to devote to building the list. The multiprocessing used here is an embarrassingly parallel algorithm. There are two “methods”; the default method incorporates all the molecular data. The other method incorporates only the atom data and can be accessed by setting method to ‘atom’.

J.7 Future Versions

Updates to the software will add bonding of metallic and polymer nano particles to the polymer matrix. This expansion of the software capabilities may change which language the code is written in, but the current functionality and method calls should change little.
APPENDIX K

MANUAL FOR RNG ALGORITHM

K.1 Introduction

This algorithm was developed in Fortran 77 to allow the random walk algorithm to be used for initializing the polymer in nano composites. In this algorithm there are two RNG algorithms. One is the traditional seed based RNG that uses a table or functions. The other is a novel algorithm which uses mirrors along with the nano particle position and chain position. The final output of the algorithm is the initialized polymer written in the format of a LAMMPS data file. This manual will not go into the details of how to build a LAMMPS data file as this information can be found inside the LAMMPS package itself. But, the manual will explain how to set up the software to build an initialized polymer. Additionally, the requirements and limitations of compiling the software will be discussed.

K.2 Compiler Requirements and Limitations

To run this software, the g77 compiler needs downloading. This software has not been tested with other Fortran 77 compilers; so, other compilers may not compile the code correctly. If g77 can not be installed on your machine, the code can easily be converted to Fortran 90.

To convert the code to Fortran 90, two steps are required. The first is to replace all of the “c”s at the beginning of comment lines with “!”. The comment lines are not indented and are left justified. The second step is to change the file from “.f” to “.f90”. This conversion process has been tested with the compiler gfortran.

To compile, open up a terminal and type “compiler” “software file” -o “program name” and than hit enter. A file with “program name should be created”. Just type in .”/program name”. If you are worried about optimization flags for additional speed, the software runs very fast and so there was no need for extra
flags to be added. This also means if optimization flags are added, the software may not run correctly.

K.3 Setting up the Algorithm to Initialize the Polymer Chains

K.3.1 Setting up the Parameters for the Polymer Chains

The parameters needed to set up the polymer chains are below. The chainnum is the number of polymer chains that are produced by the algorithm. The unitnum is the number of units in each polymer chain. The atomnum is the number of atoms or particles in each unit. The kuhncount is the number of units placed before the mirror based RNG algorithm is used. The kuhncount should either be set to match the Kuhn length or to 0 which makes the mirror based RNG algorithm run every time a unit is placed.

```
integer chainnum, atomnum, unitnum, kuhncount
parameter (chainnum=30, unitnum=100, atomnum=7, kuhncount=7)
```

K.3.2 Setting up the Number of Coefficients for Molecular Structures

The different molecular structures are the the atoms, bonds, angles, dihedrals and impropers. For each of these structures, there are coefficients which effect how the simulations are run. The atomtypenum is the number of different atom types in the polymer. This numbers effects the number of atomic masses needed in the data file. The pair potentials which also use atom types are not stored in the data file but are in LAMMPS scripts. The bondtypenum is the number of bond potentials needed to describe the polymer chain. The bond potentials are stored in the matrix bondcoef. The first number next to bondcoef should be the number of parameters needed for the bond potentials. The angletypenum is the number angle potentials needed to describe the polymer chain. Again, the first number next to anglecoef is the number of parameters required by the potential. The dihedraltypenum and dihedralcoef correspond to the dihedrals and follow the same pattern as the angles. The impropertypenum and impropercoef follows the same pattern as the angles as well.

```
integer atomtypenum, bondtypenum, angletypenum, dihedraltypenum, impropertypenum
```
parameter (atomtypenum=6, bondtypenum=6, angletypenum=6, dihedraltypenum=6, impropertypenum=1)
double precision bondcoef(2,bondtypenum), anglecoef(2,angletypenum)
double precision dihedralcoef(5,dihedraltypenum), impropercoef(2,impropertypenum)

K.3.3 Setting up the Size of Different Molecular Structures

The code below shows the size of different matrices which make up the molecular structure. The atoms is the atomic information in one unit of the polymer chain. The bondinit matrix is the bond information for the initial unit of the polymer chain and bonds matrix is the bond information for all other units in the polymer chain. The angleinit matrix and angles matrix are for the angles information and follow the same pattern as the bond matrixes. The dihedral information is stored in dihedralinit and dihedrals and follows the same pattern as the bond matrixes. The impropers matrix does not follow this pattern. In all matrixes, the first number corresponds to the amount of descriptors needed for a molecular structure, and the second number corresponds to the number of the molecular structures in each unit. These values can be changed to fit the requirements for each structure.

double precision atoms(5,atomnum)
integer bondinit(3,6), bonds(3,7), angleinit(4,7), angles(4,11)
integer dihedralinit(5,6), dihedrals(5,13), impropers(5)

K.3.4 Setting up the Box Length and Nano Particle Size

The code below sets up the box length and the nano particle size. The lattice size is the box length in angstroms. The hspherer is the radius of the nano particle in angstroms. If there is no nano particle, set the value to 0.0.

double precision hspherer, latticesize
parameter (hspherer=0.0, latticesize=75.017531)

K.3.5 Setting up the Different Molecular Structures and Their Coefficients

The code below shows how to set up the different molecular structures and their coefficients. Each row corresponds to the descriptors needed for a single item of the structure. The number of items in each row should correspond to the first number after the matrix declaration. The number of rows in each
structure correspond to the second number after the matrix declaration. The only exception is the for the variables addition, and masses, since these are vectors and not matrixes. All of these structures follow the rules in the LAMMPS manual except the id value and molecular id have been removed. These are taken care of in the code.

data atoms/3, .11, 0, 0, 0,
& 2, 0, -.152429914864554, 0, .2193447182826,
& 1, 0, .333005967244697, -.123820341059349, -.8529708903439,
& 4, .31, .75, 0, 1.29903810567666,
& 5, -.37, .262586755537238, .528275496308508, 2.28481360198517,
& 6, -.31, 2.03631224555881, -.619208163705874, 1.38199626938253,
& 1, .26, 2.43869486125905, -.42813272480992, 1.28481360198517,
& adition/333005967244697, 1.23820341059349, -.852970890343899,
& masses/12.0107, 12.0107, 12.0107, 12.0107, 15.9994, 15.9994/,
& bondinit/1, 1, 3,
& 2, 1, 2,
& 3, 1, 4,
& 4, 4, 5,
& 5, 4, 6,
& 6, 6, 7/,
& bonds/2, 1, -5,
& 1, 1, 3,
& 2, 1, 2,
& 3, 1, 4,
& 4, 4, 5,
& 5, 4, 6,
& 6, 6, 7/,
& bondcoef/368, 1.539,
& 300, 1.549,
& 326, 1.517,
& 968, 1.209,
& 471, 1.360,
& 342, 1.446/,
& angleinit/2, 2, 1, 3,
& 2, 3, 1, 4,
& 2, 2, 1, 4,
& 4, 1, 4, 5,
& 3, 1, 4, 6,
& 5, 6, 4, 5,
& 6, 7, 6, 4/,
K.3.6 Calculating the Number of Bonds, Angles, Dihedrals, and Improper
This is the code used to calculate the number of bonds, angles, dihedrals and impropers. These values are written in the LAMMPS data file. If these equations dont match your specific system, than change them.

\[
\begin{align*}
\text{bond num} &= \text{chain num} \times (\text{unit num} - 1) \times \text{atom num} + \text{chain num} \times (\text{atom num} - 1) \\
\text{angle num} &= \text{chain num} \times (\text{unit num} - 1) \times (\text{atom num} + 4) + \text{chain num} \times (\text{atom num}) \\
\text{dihedral num} &= \text{chain num} \times (\text{atom num} - 1) + \text{chain num} \times (\text{unit num} - 1) \times (2 \times \text{atom num} - 1) \\
\text{improper num} &= \text{chain num} \times \text{unit num}
\end{align*}
\]

K.3.7 Setting up the Random Number Generator in Fortran

The code below is used to set up the random number generator that comes with Fortran. The value 14 in the code below is the seed number. This random number generator is used to place the initial unit in the polymer chain.

```fortran
CALL SRAND(14)
```

K.3.8 How to Remove Bonds, Angles, Dihedrals, or Impropers From the Data File

If for some reason you don’t need to place the bonds, angles, dihedrals, or impropers in the LAMMPS data file. The code that takes care of each of these sections can either be removed or commented out. An example of the angles and angle coefficient sections are shown below.

```fortran
C write the angles section
WRITE (25,*) 'Angles'
WRITE (25,*)
COUNT = 1
DO 40 I = 1, CHAINNUM
   DO 41 J = 1, ATOMNUM
      WRITE (25,*) COUNT, ANGLEINIT(1, J),
                     ANGLEINIT(2, J) + (I-1)*UNITNUM*ATOMNUM,
                     &
                     ANGLEINIT(3, J) + (I-1)*UNITNUM*ATOMNUM,
                     ANGLEINIT(4, J) + (I-1)*UNITNUM*ATOMNUM
      COUNT = COUNT + 1
   41    CONTINUE
   DO 42 J = 2, UNITNUM
      DO 44 K = 1, ATOMNUM + 4
         WRITE (25,*) COUNT, ANGLES(1, K),
                     ANGLES(2, K) + (I-1)*UNITNUM*ATOMNUM + (J-1)*ATOMNUM,
                     &
                     ANGLES(3, K) + (I-1)*UNITNUM*ATOMNUM + (J-1)*ATOMNUM,
```

503
& angles(3,k)+(i-1)*unitnum*atomnum+
(j-1)*atomnum,
& angles(4,k)+(i-1)*unitnum*atomnum+
(j-1)*atomnum
    count=count+1
44 continue
42 continue
40 continue
    write (25,*)

C write the angle coefficients section
write (25,*) 'Angle Coeffs'
write (25,*)
do 43 i=1,angletypenum
    write (25,*) i, anglecoef(1,i), anglecoef(2,i)
43 continue
    write (25,*)
APPENDIX L

MANUAL FOR METAL POTENTIAL FITTING

L.1 Introduction

In this appendix, the framework design, which was discussed in the main body of text, is explained in more detail. The text files in the controls and database are examined, and methods to insert more methods/property calculations in each module are discussed. Additionally a brief explanation on compiling the current code is given, and a short discussion on the Main Controller module.

L.2 Compiling Current Code

To run this software, the g77 compiler needs downloading. This software has not been tested with other Fortran 77 compilers; so, other compilers may not compile the code correctly.

Before compiling the code a Makefile."os" needs to be created. The file will look like:

# Makefile for macintosh for calculating potentials

SHELL = /bin/sh
#.IGNORE:

# System-specific settings

F90 = g77
F90FLAGS =
LINK = g77
LINKFLAGS =
USRLIB =
SYSLIB =
SIZE = size

# Link rule

$(EXE): $(OBJ)
    $(LINK) $(LINKFLAGS) $(OBJ) $(USRLIB) $(SYSLIB) -o $(EXE)
    $(SIZE) $(EXE)

# Compilation rules

.f.o:
    $(F90) $(F90FLAGS) -c $<

.c.o:
    $(CC) $(CCFLAGS) -c $<

# Individual dependencies

include Makefile.depend

If the Fortran 77 compiler on your system is different than g77, then replace g77 with your Fortran 77 compiler. After setting up your Makefile."os", you need to add the os to the Makefile. This Makefile is below:

# Multiple-machine Makefile

SHELL = /bin/sh

# Files

506
SRC = potfit.f fccstruct.f hexagonalclosepackstruct.f struct110.f struct111.f\  optimize.f srsolvetest.f functionfolderwriter.f letterchoozer.f functionwriter.f\  srsolvefit.f fpart.f stablefault.f unstablefault.f splinemethod.f\  shortneldermead.f\  splinehandler.f elastic.f vacancy.f potential_vacancy_stress_new.f\  vacancy_conjugate.f\  surface100.f surface110.f surface111.f hexagonalclosepackenergy.f\  roseeos.f dmvcalc.f\  periodicboundaryconditionenergy.f gradsolve.f vectormagnitude.f dot.f\  crystalcreation.f\  propreader.f mdsetupfit.f mdsetuptest.f lammpspot.f

# Definitions

ROOT = nifit
EXE = $(ROOT)_$@
OBJ = $(SRC:.f=.o)

# Help

help:
  @echo 'Type "make target" where target is one of:.'
  @echo ' serial (for SGI O2/10000)'  
  @echo ' sgi (for SGI O2/10000 w/ MPI)' 
  @echo ' tflop (for Intel Tflop)' 
  @echo ' alaska (for CPlant - alaska)'  
  @echo ' siberia (for CPlant - siberia)'

507
@echo 'origin (for SGI Origin)'
@echo 't3e (for Cray T3E)'
@echo '8400 (for Dec 8400 Cluster)'
@echo 'geo (for Dec geo)'
@echo 'c90 (for Cray C90)'
@echo 'sith (for GT CC sith cluster)'
@echo 'ia64 (for Titan)'
@echo 'opt (for Penguin)'
@echo 'linux (for Ubuntu Linux)'
@echo 'mac (for Macintosh)'

# Targets

serial sgi tflop alaska siberia origin t3e 8400 geo c90 sith ia64 opt linux mac:
  @if [ ! -d Obj_$@ ]; then mkdir Obj_$@; fi
  @cp -p $(SRC) Makefile.depend Obj_$@
  @cp Makefile.$@ Obj_$@/Makefile
  @cd Obj_$@; \
  $(MAKE) "OBJ = $(OBJ)" "EXE = ../$(EXE)" ../$(EXE)
  @if [ -d Obj_$@ ]; then cd Obj_$@; rm $(SRC) Makefile*; fi

depend:
  Depend.pl $(SRC)

# Cleans

clean_serial:
  rm -r Obj_serial

clean_sgi:
  rm -r Obj_sgi
clean_alaska:
    rm -r Obj_alaska

clean_siberia:
    rm -r Obj_siberia

clean_tflop:
    rm -r Obj_tflop

clean_origin:
    rm -r Obj_origin

clean_t3e:
    rm -r Obj_t3e

clean_8400:
    rm -r Obj_8400

clean_geo:
    rm -r Obj_geo

clean_c90:
    rm -r Obj_c90

clean_sith:
    rm -r Obj_sith

clean_ia64:
    rm -r Obj_ia64
clean_opt:
    rm -r Obj_opt

clean_linux:
    rm -r Obj_linux

clean_mac:
    rm -r Obj_mac
To add the os, add an @echo "os information" in the section marked help; add
the os in the line after "#targets", and add a line clean_"os": with a tabbed line rm
-r Obj_"os" after the "#Cleans" line. After all these steps, open a terminal and go
to the directory where the Makefile is. Type make "os" and the software will
compile.

L.3 Controls

The controls are placed in the controls folder. They consist of three
different text files. One of these text files selects the number of components for
the metallic material, components of the metallic material, optimization method
and method/equations to produce the potential from the fitting parameters. The
other two text files control the fitting and testing properties.

L.3.1 Selecting Methods and Components of Metallic Material
The name of this text file is control.txt. The file is ordered in the following manner:
number of elements
element 1
element 2
...
...
element n
optimization method
method/equations to produce the potential.
Each of these lines except the elements is a non zero positive integer. The elements are two characters corresponding to the elemental symbol in the periodic table.

L.3.2 Selecting the Properties for Fitting and Testing

The names of the files for fitting and testing are fit.txt and test.txt. The files are ordered in the following format:

- number of fitting or testing properties
- property 1
- property 2
- ...
- ...
- property n.

Each of these lines is a non zero positive integer.

L.4 Database

The database is placed in the database folder. This folder includes initial parameters and constant parameters. These parameters are used to build the potential. The values of the different properties that are calculated during fitting or testing and other properties that are never calculated are in the database as well. Additionally the database contains information on the location of the x values for doing cubic spline fitting and weights of the different properties used for calculating the weighted sum of residuals squared.

L.4.1 Parameters Used to Build the Potentials

The parameters used to build the potentials come from both the optimization software and constants stored in a file called "element"knownconst.txt. The “element” in the file name is a two character element symbol. This file is for the cubic splines and contains the two y values that are set constant before the program is run. The optimization software
requires an initial set of parameters. These parameters are got from the file “element”initialconstants”a””b””c”.txt. The “element” is the same definition as was used previously; “a”, “b”, and “c” correspond to number of knots, points used for each function where “a” corresponds to the pair potential, “b” corresponds to the embedding energy, “c” corresponds to the embedding energy. For the cubic spline method the format of a file with “a”, “b” and “c” knots is shown below:

pair potential parameter 2
pair potential parameter 3
....
....
pair potential parameter “a”-1
electron density parameter 2
electron density parameter 3
....
....
electron density parameter “b”-1
embedding energy parameter 2
embedding energy parameter 3
....
....
embedding energy parameter “c”
cutoff radius.

The first pair potential and electron density parameter are constant and stored in the “element”knownconst.txt. The final pair potential and electron density parameter are zero because that is where the cut off for the functions occur. For the embedding energy, the first parameter is zero. The rest of the parameters except the one at integer(c/2) is in text file. The parameter at integer(c/2) occurs at the equilibrium electron density of one and therefore is calculated using the equilibrium material.

L.4.2 Handling the x axis Values for the Cubic Spline
The section J.4.1 handles the y values for the cubic spline but does not explain how to handle the x values of the cubic spline. The x values for the three functions in the EAM are made complicated by the fact the cut off radius shifts during the fitting procedure. The x values in “element”scale”a””b””c”.txt are scaled values rather than absolute values. What this means is the value is the actual x value/length of axis. The length of the axis is the cut off radius for the pair potent and electron density, and the value of 2.0 for the embedding energy. The format of this file is

pair potential x scale 2
pair potential x scale 3
...
...
pair potential x scale “a”-1
electron density x scale 2
electron density x scale 3
...
...
electron density x scale “b”-1
embedding energy x scale 2
embedding energy x scale 3
...
...
embedding energy x scale “c”-1.

The first x value for the pair potential and electron density are permanently set to one. The final x value for the pair potential and electron density and set to the cut off radius which changes. The x scaling values for the pair potential and electron density allow the x values in between the first and the final to shift with the changing cut off radius. The first x value for the embedding energy is set at zero which is where the function starts. The final x value for the embedding energy is set at two. The x scaling values for the embedding energy between the final and the first are just for consistency; they do not enable any new features.
L.4.3 Material Property Values Stored in the Database

Two types of properties are stored in the database. Those that are calculated during fitting and testing and those that are used to set up property calculations. The properties that are calculated during fitting and testing are stored in a file called “element"knownvalues.txt in the database folder. The format of this file is the following:

dummy line
dummy line
property #, property value(s)
property name.
The first dummy line is “property # property value(s)”. The second dummy line is property name. The properties that are used to set up property calculations are stored in the file “element"materialconst.txt. The format of this file is mass, lattice constant at temperature, lattice constant at 0K, cohesive energy, bulk modulus.

L.4.4 Weights Used for Calculating Weighted Sum of Residuals Squared

The weights used for calculating the weighted sum of residuals squared are stored in the file “element"weightvalues.txt. The format used in this file is
dummy line
dummy line
property #, weight value
property name.
The first dummy line is property # weight value. The second dummy line is property name. The weights used currently are either 1, 0.666, or 0.3333. The weights should correspond with the accuracy of the material properties.

L.5 Inserting Methods, Calculations, Algorithms into the Different Modules

There are four ways to add methods, calculations and optimization algorithms into the different modules. Additionally each module has its own method of adding new methods.
L.5.1 Adding Fortran Files

Fortran 77 code is added to the software by adding the file name to the make file. The file will additionally need to be connected to its module by adding the function call into the module; more information about this will be discussed in each module’s section. Additionally, if the file needs to connect to other modules, function calls to those modules will need to be added.

L.5.2 Adding C and C++ Files

For C and C++ code, the make file needs to be modified to handle multiple language compilations. Additionally, the file needs to be added to the makefile. When the file is connected to it’s corresponding module, special function calls will need to be used. Also when connecting the file to other modules, special function calls will need to be made there as well. There are a lot of great resources on how to do this mix compilation and how to do mix programming language function calls; so, no further information will be given on this topic.

L.5.3 Adding Python Files

There are literally two types of python files that can be added. The first type is added to the fitting and testing properties modules and is significantly easier to add. For the first type, a system call in fortran is made to execute the python script. The python script can call other software packages and do calculations of its own. The final results are communicated back to the fitting and testing module through the file ans.txt. The second type is added to other modules and is more difficult to add. Unfortunately its not 100 percent clear how to do this addition. the best way may be to embed python in a c file; though, this method hasn’t been tested.

L.5.4 Optimization Module

In the optimization module the fitmethod variable controls which optimization method is selected. So, to add another optimization method, add a
new if statement with the number fitmethod should equal to utilize that optimization method. Add the new method’s code or make a call to its method. The optimization methods read the initial parameters from the database. So this should be added into your method unless this feature does not need to be implemented. Additionally the optimization methods need to call the potential creation module which is currently called splinemethod.

L.5.5 Potential Creation Module

Adding new potential creation methods to the potential creation module is the most confusing addition to the framework because additions and modifications are not just made to the potential creation module but also to the main controller module. To add a method to the potential creation module, add a new if statement with the number potmethod should correspond with to call the new method. The variable potmethod controls which potential creation method is called. In this new method, an EAM potential should be created with the parameters coming from the optimization module. After the potential has been created in the new method a function call to the fitting module, srsolvefit, should be in the code.

Additionally, since creating the potential requires the correct parameters, the method in the main controller module must access the correct “a”, “b”, and “c” values that correspond to the parameters in the potential creation module. The method in the main controller module also uses the same potmethod selection variable as was used in the potential creation module. Additionally each method in the main controller module can use different criteria for selecting the best potential.

L.5.6 Fitting and Testing Properties Modules

The properties that are calculated in the fitting and testing properties modules are selected using the variable propnum. So to add a new property calculation, add a new if statement with the number propnum should equal to call that calculation. In the if statement, the call to the file/function should be added. Additionally a call to propreader should be added which returns the exact value
and the weight of the property. These values are than used to calculate weighted sum of residuals squared. You will need to alter propreader if there are more than one value for a property.

L.6 Extra Information on “a”, “b” and “c”

The “a”, “b”, and “c” values can only be 1 to 9. This limited value range gives a finite number of initialization text files that can be used. Currently for the spline fitting, “a” can either be 8 or 9; “b” can be 6 to 8; and “c” can be 5 to 7. Consequently, these values of “a”, “b” and “c” are taken for nickel. The rest of the values are open for new potential creation methods being added into the potential creation module.