5-2015

Improved self-consistency for SCED-LCAO.

Lyle C. Smith
University of Louisville

IMPROVED SELF-CONSISTENCY FOR SCED-LCAO

By

Lyle C. Smith, III
B.S. Physics, Virginia Tech, 1995
M.S. Mathematics, Virginia Tech, 1997
M.S. Physics, University of Louisville, 2005
M.Div., Southern Baptist Theological Seminary, 2005

A Dissertation
Submitted to the Faculty of the
College of Arts and Sciences of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy in Applied and Industrial Mathematics

Department of Mathematics
University of Louisville
Louisville, Kentucky

May 2015

IMPROVED SELF-CONSISTENCY FOR SCED-LCAO

By

Lyle C. Smith, III
B.S. Physics, Virginia Tech, 1995
M.S. Mathematics, Virginia Tech, 1997
M.S. Physics, University of Louisville, 2005
M.Div., Southern Baptist Theological Seminary, 2005

A Dissertation Approved on

24 April 2015

by the following Dissertation Committee:

Dr. Shi-Yu Wu, Dissertation Director

Dr. Chakram Jayanthi

Dr. Lee Larson

Dr. Thomas Riedel

Dr. Prasanna Sahoo

Dr Ming Yu

This dissertation is lovingly dedicated to my wife, Erin Smith. Her support, encouragement, and constant love have sustained me throughout my life. Together with my children, she has shown patience and love when none was deserved. She never wavered in her belief in me and for that I cannot offer sufficient thanks.

ACKNOWLEDGEMENTS

I would like to acknowledge the inspirational instruction and guidance of Dr. Shi-Yu Wu. He has given me a deep appreciation and love for the beauty and detail of quantum mechanics in general and condensed matter theory specifically. His guidance was also essential to this work.

I owe a great debt to Dr. Yu Ming. Her continued help and advice in the operation of the molecular dynamics code was indispensable, especially when time was short. She has always assisted with every request and I look forward to continuing our efforts together.

Dr. Chakram Jayanthi has been a fountain of suggestions and directions for improvement of the research and this dissertation. Thanks also to her for introducing me to the joy of scientific computing and for instruction in the finer points of solid state physics.

Professor Thomas Riedel and everybody in the University of Louisville Department of Mathematics has supported me both academically and financially at every turn.

Dr. Lee Larson taught me both advanced real analysis and typesetting in LaTeX, later providing an assistantship where I could hone my skills in both. I truly enjoyed my time spent working on *The Real Analysis Exchange*.

Dr. Prasanna Sahoo took interest in my work and provided valuable feedback on the improvement of this work.

Finally, I would also like to acknowledge the support and assistance given me by my co-workers Dr. Christopher Leahy, Paul Tandy, and Harrison Simrall. Their patient explanations and mutual encouragement were absolutely necessary in the generation of all results obtained.

ABSTRACT

IMPROVED SELF-CONSISTENCY FOR SCED-LCAO

Lyle C. Smith, III

24 April 2015

In this document I describe a novel implementation of the generalized bisection method for finding roots of highly non-linear functions of several variables. Several techniques were optimized to reduce computation time. The implementation of the bisection method allows for the calculation of heterogeneous systems with SCED-LCAO, since derivative-based methods often fail for these systems.

Systems composed of Gallium and Nitrogen are currently receiving much interest due to their behavior as semi-conductors and their ability to form nano-wires. The methods developed here were employed to create a set of SCED-LCAO parameters for homogeneous Gallium and heterogeneous Gallium Nitride systems. These parameters were shown to provide SCED-LCAO with predictive power for future Gallium Nitride systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 – Motivating Applications

As physics pushes into the nano scale and begins to create objects on the order of 10^{-9} meters, classical modeling of these atomic systems begins to break down. Numerous solid state phenomena can be predicted only with quantum mechanics. The creation of complex machinery comprised of only hundreds of atoms is rapidly becoming a reality. Recent research has seen the creation of a nanocar by a research group at Rice University (see figure 1.1)[1] that can crawl along material surfaces. Exciting opportunities exist in manufacturing, medicine, electronics, and other areas for this emerging technology.

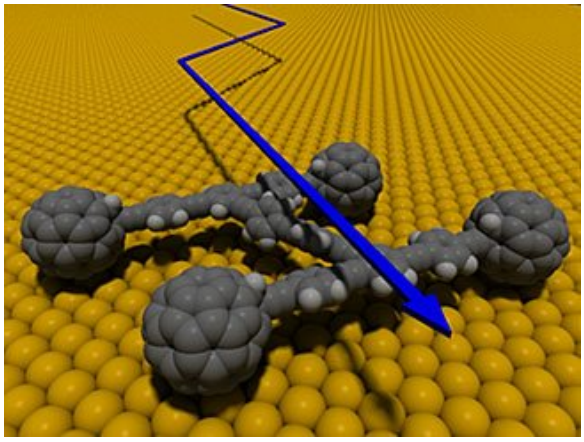


Figure 1.1: Nanocar

The rediscovery of Carbon Nanotubes (CNTs) in 1991 has resulted in a blossoming field of applications. Along with this renewed interest, there is a great need for modeling capable of predicting electrical, chemical, thermal, and other properties of these structures (see figure 1.2). New applications for CNTs are being discovered at a rapid pace. These include photo-voltaic cells, transistors, chemical sensors, and protective clothing.

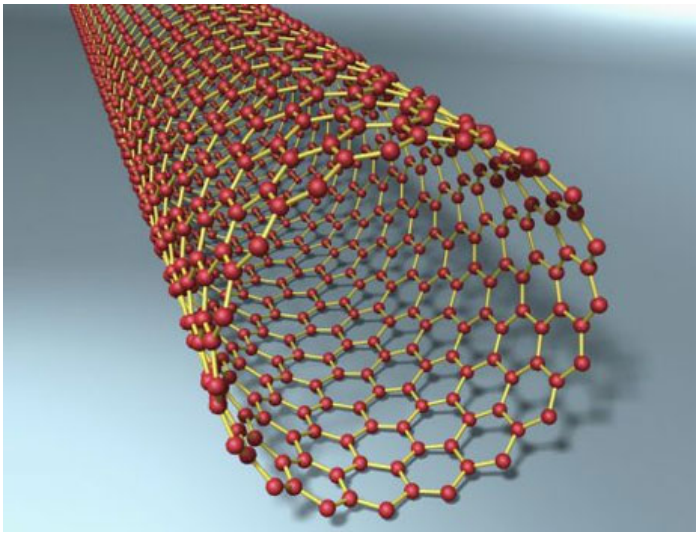


Figure 1.2: Carbon Nanotube

Another focus of recent research is with heterogeneous gallium nitride structures. Gallium nitride is a semiconductor that is known to be able to handle high thermal stress and high voltage. Thus, it is ideal for power amplifiers, especially when operating at high frequencies. As the structural limit of silicon manufacturing of microchips becomes a larger factor, gallium nitride is seen as a possible substitute that should allow higher frequency computing. Gallium nitride is also resistant to ionizing radiation, making it an excellent choice for electronics operating in space. Enhancement mode GaN transistors are poised to replace standard MOSFETs, providing improved efficiency under higher load. This technology is proving invaluable for the construction of the Smart Power Grid. Gallium Nitride has also been shown to form nanotubes [2]. These have been employed to create blue and ultraviolet light emitting diodes and for stimulated emission in blue lasers. Gallium nitride is proving to be a versatile and valuable material. Research into its various properties is rapidly expanding.

Given the many and emerging technologies that rely heavily on nano-scale architectures and quantum mechanical effects, there exists a need for a computation scheme that can accurately model these systems quickly. Even systems as large as thousands of atoms will behave in strictly quantum mechanical ways. In order to explain and predict this behavior,

we need a method that can calculate properties for thousands of atoms while maintaining the salient quantum mechanical flavor.

1.2 – Basics of the Problem

Quantum Mechanics has shown itself to be an invaluable framework for understanding physics at the atomic and molecular levels. Modern physics relies on the assumptions and conclusions of quantum mechanics in nearly every field. While the results of quantum mechanics have proven to be very reliable, the mathematics employed often leads to highly complex and intractable problems.

The Schrödinger Equation is used analogously to Newton's Second Law ($F = ma$) and thus forms the basis of most quantum mechanical calculations. It is a partial differential equation relating the time derivative of the wave function to the energy operator (Hamiltonian) operating on the wave function. The wave function ψ itself is a mathematical construct that when multiplied by its complex conjugate yields the probability density function for the system. For example, if $\vec{r}$ represents the position vector of the particles in the system, then the Schrödinger Equation has the form

$$i\hbar\frac{\partial}{\partial t}\psi(\vec{r},t) = \hat{H}\psi(\vec{r},t),$$

where $i = \sqrt{\text{-}1}$, $\hbar$ is Planck's constant divided by 2π, t is time, and $\hat{H}$ is the Hamiltonian of the system. For a system of many particles, this becomes

$$i\hbar\frac{\partial}{\partial t}\psi(\vec{r_1},\vec{r_2},\ldots,t) = -\frac{\hbar^2}{2m}\nabla^2\psi(\vec{r_1},\vec{r_2},\ldots,t) + V(\vec{r})\psi(\vec{r_1},\vec{r_2},\ldots,t)$$

where the Hamiltonian has been expressed in terms of its kinetic operator ($\text{-}\frac{\hbar^2}{2m}\nabla^2$) and potential ($V(\vec{r})$) component. The solution of this equation is intractable for all but the simplest problems. Therefore, simplifications and approximations are necessary to tackle many of the problems in solid state physics.

The field of computational material science is the branch of physics that attempts to predict chemical, electrical, structural, and other properties of a system of atomic particles. As with most fields of computational science, there is an inherent trade-off between computational speed and accuracy. Increases in speed come with additional assumptions and approximations that often reduce accuracy. Since the nuclear mass is orders of magnitude larger than the electron mass, we can apply the Born-Oppenheimer Approximation and solve the wave function for the electrons while treating the nucleus classically. We also assume that the system is non-relativistic. Then we employ the variational principle to calculate the stationary state associated with the lowest total energy of the system, since this will be favored by nature. A stationary state is an eigenvector of the Hamiltonian, i.e. $\hat{H}\Psi = E\Psi$, where the energy (the eigenvalue) is independent of time. This leads to the Many-Body Schrödinger Equation, with potential given by

$$V = \frac{1}{2}\sum_{n\neq m}\frac{1}{|\vec{r_n} - \vec{r_m}|} - \sum_{n,m}\frac{Z_{I_m}}{|\vec{r_n} - \vec{R}_{I_m}|} + \frac{1}{2}\sum_{n\neq m}\frac{Z_{I_n}Z_{I_m}}{|\vec{R}_{I_n} - \vec{R}_{I_m}|}.$$

A variety of different approximations are employed at this point. Nearly all of them restrict the solutions to the ground state of the system. Hartree-Fock Theory was developed early in the history of quantum mechanics, but was not widely used until the advent of computers. This method approximates the many-body wave function with a Slater Determinant of single orbital functions. Here we express the total wave function of the system as a product of one-electron atomic orbitals (AO). An atomic orbital is a stationary state of a one-electron atom, whereas a molecular orbital is a stationary state for an electron in a molecule. For a many-electron system, a molecular orbital is very different from a total stationary state. The Hartree-Fock Method uses the atomic orbitals to approximate the molecular orbitals and then combines the molecular orbitals to find a total stationary state of the system through a linear combination of atomic orbitals (LCAO). The wave function is found through an anti-symmetrized (to account for fermions) determinantal product of atomic orbitals known

as the "Slater" determinant. In this way, Schrodinger's equation is transformed into a set of Hartree-Fock equations.

The Hartree-Fock method was ground-breaking in that it allowed the calculation of wave functions for many-body systems. It is a self-consistent field (SCF) method in that the final result is required to be consistent with the initial field. In practice, this means that Hartree-Fock calculations are performed iteratively until the output field matches the input field. The accuracy of the representation of the true molecular orbital increases with the size of the basis set. Therefore, there is a wide array of basis sets with varying degrees of computational expense. However, the concept of a one-electron atomic orbital is only meaningful if we ignore the electron-electron repulsion terms in the Hamiltonian. Due to this Coulomb correlation, the total Hartree-Fock electronic energy is always higher (less negative) than the actual electronic energy. The difference is called the correlation energy. This rather large omission limits the accuracy of Hartree-Fock. Other methods and refinements were developed to provide more accurate results.

Another approximation is Density Functional Theory (DFT), which transforms the problem by operating on the electron density function rather than the wave function itself. DFT, after refinement to better model the exchange and correlation interactions, provides highly accurate results for most systems, but still struggles in some areas, such as modeling Van der Waals forces. When DFT is employed with a large basis set and a modern correlation-exchange functional, it is computationally expensive and is limited to smaller atomic systems for practical calculations.

At the opposite extreme is classical potential modeling, which models only pairwise atomic interactions using a classical energy potential. The Lennard-Jones potential and its refinements have proven useful for modeling very large systems, but at the expense of removing any quantum mechanical flavor. Tight-Binding methods attempt to provide additional accuracy, but still only deal with two-center interactions.

1.3 – SCED-LCAO Theory

The Self-Consistent Environment Dependent Linear Combination of Atomic Orbitals (SCED-LCAO) approach was developed by Dr. Shi-Yu Wu in order to provide a reliable and transferable semi-empirical method for quantum-mechanics based simulations of materials. Charge redistributions are calculated through the use of a Self-Consistent (SC) iteration and the effects of electron screening and electron-electron correlation are contained in Environment-Dependent (ED) multi-center terms which handle two-center and three-center interactions explicitly and four-center interactions implicitly.

In the framework of a semi-empirical LCAO-based approach, we seek to find functions which adequately model the interactions between electrons for each atomic orbital α and for each atom i. We define $\vec{R}_{ij} = \vec{R}_i - \vec{R}_j$ to be the relative position vector from atom i to atom j and solve the matrix form of the Schrödinger Equation

$$Hc_\lambda = E_\lambda S c_\lambda, \tag{1.1}$$

where H is the SCED Hamiltonian and S is the overlap matrix corresponding to the basis functions $\phi_{i\alpha}(\vec{r})$ used. It is important to note that this formulation admits a range of basis functions, thus providing flexibility. This forms a general eigenvalue problem where we solve for the eigenvectors c_λ, which define the coefficients of expansion of the eigenfunction ψ_λ in terms of the basis functions $\phi_{i\alpha}$, and E_λ which corresponds to the energy of each orbital. The geometrical configuration of the atomic nuclei for which the energy is minimized gives the equilibrium structure, from which we find bond lengths, bond angles, the lattice parameters, etc.

What makes SCED unique is the construction of the Hamiltonian for Equation (1.1). The Hamiltonian matrix for the many-body system can be written as

$$H = -\sum_l \frac{\hbar^2}{2m}\nabla_l^2 + \sum_{l,i} v(\vec{r}_l - \vec{R}_i) + \sum_{l,l'} \frac{e^2}{4\pi\varepsilon_0 r_{ll'}} + \sum_{i,j} \frac{Z_i Z_j e^2}{4\pi\varepsilon_0 R_{ij}}$$

where the summations over l and l' are taken over all valence electrons, $r_{l,l'} = |\vec{r}_l - \vec{r}_{l'}|$, $R_{i,j} = |\vec{R}_i - \vec{R}_j|$, and Z_i is the number of valence electrons associated with the ion at $\vec{R}_i$. The first term captures the kinetic energy of the electron, the second term is the potential energy between an electron at r_l and the ion (nucleus and inner electrons) at R_i, the third term represents the electron-electron interaction, and the last term represents the ion-ion interaction.

The SCED Hamiltonian diagonal (on-site) elements are rewritten as

$$H_{i\alpha,i\alpha} = \varepsilon^0_{i\alpha} + u^{intra}_{i\alpha} + u^{inter}_{i\alpha} + v_{i\alpha}.$$

For the electron in orbital $i\alpha$, $\varepsilon^0_{i\alpha}$ is the kinetic energy and the interaction with its own ionic core, $u^{intra}_{i\alpha}$ is the interaction with other electrons from the same atom, $u^{inter}_{i\alpha}$ is the interaction with electrons in orbital $j\beta$ (from other atoms), and $v_{i\alpha}$ is the interaction with off-site ions. More specifically, we let $\varepsilon^0_{i\alpha} = \varepsilon_{i\alpha} - Z_i U_i$, where $\varepsilon_{i\alpha}$ is the energy of the orbital α for the isolated atom at i. For the SCED formulation, we take this to be the value of the Hartree-Fock calculated energy for the orbital. Again Z_i is the positive charge of the ion at i (nucleus and inner electrons) and also the number of valence electrons associated with the uncharged atom at i. The term U_i is a Hubbard-like term representing the effective energy of electron-electron interactions for electrons associated with the atom at site i. This Hubbard term is allowed to vary when optimizing the parameters, but we restrict its values to be near the Hubbard calculated value. We let $u^{intra}_{i\alpha} = N_i U_i$, where N_i is the number of valence electrons associated with the atom at i when the atom is in self-consistent equilibrium within the system (also called the Self-Consistency vector). We also rewrite $u^{inter}_{i\alpha} + v_{i\alpha} = \sum_{k\neq i} \left[N_k V_N(R_{ik}) - Z_k V_Z(R_{ik})\right]$. Here $N_k V_N$ represents the effective energy of interaction between an electron associated with an atom at site i and electrons associated with an atom at k, and $Z_k V_Z$ models the effective energy of interaction between an electron associated with an atom at i and an ion at site k. The terms V_N and V_Z are treated as

parametrized exponential functions that will be optimized for each elemental species.

The addition of a W term models the spread shift in the energy of the electron orbitals due to the effects of electrons from neighboring atoms. Especially with atomic species which are prone to de-localizing electrons, the electronic environment can push the local electrons into higher orbitals. This effect could be captured by adding higher energy orbitals (d, for instance), but it is more efficiently modeled with this W term. We model each type of orbital with a separate short-ranged function. For instance, for sp^3 bonding, we have

$$W_s(R_{ik}) = w_{as} \cdot e^{-w_{es} \cdot R_{ik}}$$
$$W_p(R_{ik}) = w_{ap} \cdot e^{-w_{ep} \cdot R_{ik}}$$

where w_{as}, w_{es}, w_{ap}, and w_{ep} are taken to be parameters which will be optimized.

Therefore, the diagonal elements are

$$H_{i\alpha,i\alpha} = \varepsilon_{i\alpha} + \sum_{k\neq i} W_{i\alpha}(R_{ik}) + (N_i - Z_i)U_i + \sum_{k\neq i}[N_k V_N(R_{ik}) - Z_k V_Z(R_{ik})].$$

Similarly, we defined the off-diagonal (off-site) elements of the SCED Hamiltonian as

$$\begin{aligned} H_{i\alpha,j\beta} =& \frac{1}{2}\Big\{ \varepsilon'_{i\alpha} + \varepsilon'_{j\beta} + \sum_{k\neq i} W_{i\alpha}(R_{ik}) + \sum_{k\neq j} W_{j\beta}(R_{jk}) K(R_{ij}) \\ &+ \big[(N_i - Z_i)U_i + (N_j - Z_j)U_j\big] + \sum_{k\neq i}[N_k V_N(R_{ik}) - Z_k V_Z(R_{ik})] \\ &+ \sum_{k\neq j}[N_k V_N(R_{jk}) - Z_k V_Z(R_{jk})]\Big\} S_{i\alpha,j\beta}(R_{ij}). \end{aligned} \tag{1.2}$$

Once again, N_i and Z_i are the number of electrons at site i and for a neutral site i. The first term is related to the Wolfsberg-Helmholtz relation in the extended Hückel theory. The coefficient is given by $K(R_{ij}) = e^{a_K R_{ij}}$. The term $V_Z(R_{jk})$ models the interaction between site k ion and site i electrons. Together with V_Z, $V_N(R_{jk})$ forms the environment-dependent multi-center terms. Thus, the off-site Hamiltonian elements include three-center interactions

explicitly (i, j, and k) and four-center interactions implicitly. From Equation (1.2), it can be seen that the environment-dependent multi-center interactions are dependent on V_N and V_Z. More precisely, the interactions are governed by the difference $\Delta V_N = V_N - V_Z$. Since V_Z is defined as the energy of effective interaction per ionic charge between an ion at site k and an electron associated with the atom at site i, we may model V_Z by the following parametrized function

$$V_Z(R_{ik}) = \left(\frac{e^2}{4\pi\varepsilon_0}\right)\frac{1}{R_{ik}}\left[1 - (1 + B_Z R_{ik})e^{-\alpha_Z R_{ik}}\right].$$

Similarly, we define

$$\Delta V_N(R_{ik}) = (A_N + B_N R_{ik})\frac{\left[1 + e^{-\alpha_N d_N}\right]}{\left[1 + e^{-\alpha_N (d_N - R_{ik})}\right]}.$$

The overlap matrix $S_{i\alpha,j\beta}(R_{ij})$ is comprised of mixing factors for each pair of orbitals in the system. Typically, each atom is represented by its valence electrons in their sp^3 orbitals, although d orbitals and higher are possible with SCED. In this case, each atom contributes four rows and four columns for the $ss\sigma$, $sp\sigma$, $pp\sigma$ and $pp\pi$ valence orbitals. Each is a short-ranged function of the distance between the two atoms R_{ij} represented by

$$S_{ij,\tau} = (A_\tau + B_\tau R_{ij})\frac{1 + e^{-\alpha_\tau d_\tau}}{1 + e^{-\alpha_\tau (d_\tau - R_{ik})}}$$

where τ runs over the four orbitals. Based on the orthogonality of the s and p orbitals of the same atom, we have $A_{ss\sigma} = A_{pp\sigma} = A_{pp\pi} = 1$ and $A_{sp\sigma} = 0$.

With these functions, the SCED-LCAO Hamiltonian is completely defined. In total, we have 25 parameters, using sp^3 bonding, which will need to be optimized for each elemental species. There are 12 overlap (S_{ij}) parameters, three each for the four orbitals. In addition, we have U, w_{as}, w_{es}, w_{ap}, w_{ep}, a_K, B_Z, α_Z, B_N, α_N, d_N, ε'_s, and ε'_p.

The Hamiltonian and overlap matrix elements constructed using the SCED-LCAO for-

mulation can then be used to solve the general eigenvalue Equation (1.1) for a given system of atoms to yield a set of SCED–LCAO band structure eigen-values E_λ and the corresponding eigen-vectors c_λ. Once the eigenvector coefficients are known, one can determine the total number of electrons N_i associated with an atom at site i through the expression

$$N_i = \sum_\lambda \sum_\alpha \sum_{j\beta} (c_{\lambda,i\alpha})^* c_{\lambda,j\beta}\; n_\lambda\; S_{i\alpha,j\beta},$$

where n_λ is the electron occupation number determined by the Fermi–Dirac distribution function for the specified temperature of the system. The total charge at each site (-eN_i) is calculated self-consistently through an iterative procedure and is subsequently used in the evaluation of the total energy and the atomic forces.

For a given system of atoms, the total energy consistent with the SCED-LCAO Hamiltonian is given by $E = E_{BS} - E_{dbc} + E_{\text{ion-ion}}$. Here E_{BS} is the band-structure energy, E_{dbc} is the correction to the double counting of the electron-electron interactions between the valence electrons in the band structure energy calculation, and $E_{\text{ion-ion}}$ is the repulsive interaction between ions. Using the notation above for the SCED-LCAO formulation, this can be rewritten as

$$E = \sum_\lambda n_\lambda E_\lambda + \frac{1}{2}\sum_i (Z_i^2 - N_i^2)U_i - \frac{1}{2}\sum_i \sum_{j\neq i} N_i N_j V_N(R_{ij}) + \frac{1}{2}\sum_i \sum_{j\neq i} Z_i Z_j \frac{E_0}{R_{ij}},$$

where $E_0 = \frac{e^2}{4\pi\varepsilon_0}$, E_{BS} is the first term, E_{dbc} is terms two and three, and $E_{\text{ion-ion}}$ is the final term.

1.4 – Previous SCED Work

Much work has already been done to create a working collection of SCED-LCAO tools. In 2006, Leahy et.al.[3] presented the initial findings from the first element to be fully modeled – Silicon. This included modeling binding energy for Si bulk, intermediate size clusters,

the Silicon (100) surface, and the adsorption of a Si atom on the Si (111) surface. SCED-LCAO was found for silicon to be reliable and highly transferable, thus providing predictive power for future studies. In 2009, Yu, et.al. [4], [5] extended SCED-LCAO to carbon, with special emphasis on the modeling of small carbon clusters, specifically comparing stability of fullerene, bucky-diamond, and other geometries developed by relaxing sections of bulk carbon. Also in 2009, the condensed matter theory group at University of Louisville also published a review article [6] detailing the development of SCED-LCAO theory and application. In this article, we find the modeling of the first heterogeneous systems composed of Silicon and Carbon. Unique and stable heterogeneous bucky-diamond and cage structures were discovered. This was later extended [7], [8] to include SiC tubular and graphitic structures, as well as SiC nanowires [9]. Additional results [10] extend SCED-LCAO further to include Boron and Phosporous.

In this work, I generate parameters for Gallium and Nitrogen. This represents the first effort of constructing a SCED-LCAO parameter set for an element (Nitrogen) based on a heterogeneous database. Special care is required when dealing with self-consistency on heterogeneous systems because of the higher degree of charge transfer among the atoms in a cluster. This is especially true when pairing column **III** elements with column **V** elements, such as Gallium and Nitrogen.

1.5 – Heterogeneous Systems

For SCED-LCAO to be truly transferable and applicable to a wide range of elemental types, it needs to accurately calculate results for heterogeneous systems. True transferability requires that we use the same set of parameters for each element regardless of the environment created by the surrounding atoms of the system. However, the interaction of two different types of atoms will require a mixing of the two sets of parameters. The simplest approach would be to simply average the two parameter sets to calculate the interactions between the two elements. However, it seems more natural to allow for a weighting of one

element more heavily. This is due to the fact that some elements are significantly more chemically active than others. For example, we expect a column **III** element to interact with a column **V** element in such a way as to emphasize one parameter set more than the other. To this end, we introduce a mixing term that depends on the two elements used. For example, to determine a parameter p for Ga and N, we introduce $\alpha_{Ga,N}$ and mix each parameter as

$$p_{Ga,N} = \alpha_{Ga,N} p_{Ga} + (1 - \alpha_{Ga,N}) p_N.$$

In this way, the two elemental species retain their original parameters, but share a unique mixing term, as outlined in [6].

CHAPTER 2

SELF-CONSISTENCY

2.1 – The Self-Consistent Problem

Solving the generalized Schrödinger Equation (1.1) with the SCED-LCAO Hamiltonian yields a set of eigenvalues E_λ and eigenvectors c_λ for the molecular orbitals. These eigenvalues are then ordered and electrons are assigned to them starting with the lowest energy. The assignment algorithm allows for non-integer electron values at each atomic site. This distribution of electrons to each atomic site is stored in the vector $\vec{N}$. The sum of these electron values over the atomic sites ($\sum_i N_i$) is constant, i.e. the total charge is conserved at each iteration. We will denote this total charge by T. While physically meaningless, the computer algorithm allows for the possibility of a negative number of electrons at a given atomic site. Therefore each entry in $\vec{N}$ can be any real number, i.e. for a system of n atoms $\vec{N}$ is an element of $\mathbb{R}^n$ with each entry being the number of electrons on one atom. However, the self-consistent solution will only contain non-negative real numbers.

SCED-LCAO is a self-consistent field formulation. For a given system, the first solution of Equation (1.1) will likely yield a charge distribution that, if used to solve the system a second time, will produce different results. The self-consistent solution will produce a charge distribution that yields identical results each time Equation (1.1) is solved. Therefore, we must solve iteratively for the self-consistent charge vector $\vec{N}^*$. That is, we seek to find $\vec{N}^*$ such that Equation (1.1) becomes $\hat{H}_\lambda(\vec{N}^*)c_\lambda(\vec{N}^*) = E_\lambda S(\vec{N}^*)c_\lambda(\vec{N}^*)$. Since the solution of the charge vector depends on the charge vector itself, this process can be viewed as solving for the steady state, or fixed point, of a discrete system of n equations. The equations are transformed into a root finding problem in the usual way. We rewrite the equation so that we want to find $\vec{N}^*$ such that $\hat{H}_\lambda(\vec{N}^*)c_\lambda(\vec{N}^*) - E_\lambda S(\vec{N}^*)c_\lambda(\vec{N}^*) = 0$.

A large body of literature exists concerning the self-consistency problem, but nearly all publications deal with the continuous case [11], [12]. As a result, we choose to deal with the system as if it were a black box. We know only the values of the input and output vectors. A typical Self-Consistency iteration consists of the following for a fixed set of parameters and fixed geometry.

1. Calculate $\hat{H}_\lambda(\vec{N}_{in,i})$, the SCED Hamiltonian for step i.
2. Solve the general eigenvalue problem, Equation (1.1), $\hat{H}_\lambda(\vec{N}_{in,i})c_\lambda(\vec{N}_{in,i}) = E_\lambda S(\vec{N}_{in,i})c_\lambda(\vec{N}_{in,i})$ for E_λ and c_λ.
3. Calculate the new charge vector $\vec{N}_{out,i}$.
4. Apply root-finding algorithm to choose $\vec{N}_{in,i+1}$ based on previous step(s).

We continue this process until the norm of the difference $|\vec{N}_{out,i} - \vec{N}_{in,i}|$ is satisfactorily small. The final charge vector is then the steady state solution $\vec{N}^*$. The system solution then consists of the eigen-values E_λ and eigen-vectors c_λ found by solving Equation (1.1) using $\vec{N}^*$ as the charge distribution. This is of course just the last solution found in the iterative process. Note that each step involves the computationally expensive solution of the general eigenvalue problem, including a matrix inversion on the order of $(4n)^2$ for only sp^3 bonding, where n is the number of atoms. For this reason, we would like to find the solution in a minimum number of self-consistency steps.

2.2 – System Characteristics

In its simplest form, the self-consistent problem is only an n-dimensional root-finding problem. Much work has been done to solve these types of systems so that typically one would use a readily-available numerical recipe to find the solution. However, this self-consistent problem has proven resistant to these algorithms. This is largely due to the nature of the problem – it is highly non-linear. In one dimension with one unknown atomic charge, the function for a typical dimer closely resembles the Fermi-Dirac function (see

figure 2.1). This is largely due to the use of the Fermi-Dirac function in determining the occupation of each orbital. The difficulty in solving this system is that near the solution, the

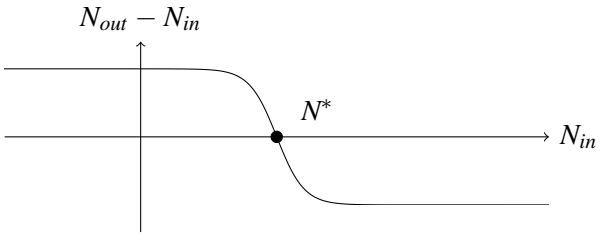


Figure 2.1: Example one-dimensional self-consistency curve.

self-consistency (SC) curve is highly non-linear, having the shape of a step function.

Given the physical origin of the system, we expect the SC curve to be continuous and differentiable. In practice, we have found that the SC curve is continuous, but with some systems the SC curve can become a step function to within computer accuracy. Thus, differentiating the function $\vec{N}_{out} - \vec{N}_{in}$ accurately is sometimes difficult or impossible. This will be a determining factor in our choice of solution algorithms. The curve is also found to be monotone decreasing in each component, with $\vec{N}_{out} - \vec{N}_{in}$ steadily decreasing with increasing N_{in} in all n components. This feature is present throughout all the SC curves we found. Essentially, this results from the self-consistent nature of SCED-LCAO. An increase in the input number of electrons N_{in} on a given atomic site will decrease the additional electrons assigned to that site $N_{out} - N_{in}$.

2.3 – Reducing System Size

Much effort was spent to precondition the system to simplify the calculations. First, we invoke the conservation of charge for a given atomic configuration. That is, the total number of electrons is fixed. For the self-consistent problem, that means we require the sum of the components of $\vec{N}$ to remain constant, i.e. $\sum_{i=1}^{n} N^i = T$, a constant. This enables us to remove one of the dimensions from the problem. We use our self-consistency algorithms to solve

for the first $n-1$ entries and then set the final entry to be whatever number of electrons are remaining (possible negative), $N^n = T - \sum_{i=1}^{n-1} N^i$.

Further reductions in the dimensions of the self-consistent problem can be obtained by exploiting the symmetry of the atomic cluster. The effect of symmetry is to define two or more atomic sites to have the same characteristics. We take this to encompass the charge on the atoms. In other words, the physics of the system requires two or more components of $\vec{N}$ to always be equal.

Figure 2.2: Ga_1N_2 cluster with D_∞ h symmetry.

In figure 2.2 the two Nitrogen atoms (1N and 3N) are at equal distances from the central Gallium atom (2Ga). Thus, we assume that they will have equal charge for every solution for this geometry. Therefore, there are only two unknowns for this cluster – the charge on the central Gallium atom and the charge on either Nitrogen atom. Combine this with the conservation of charge, and we can reduce the self-consistent problem for this cluster to one dimension.

Unfortunately, while the reduction in size for the SC problem will greatly speed up the computational process, it does not improve the shape of the curve. For example, we found that for one 8-atom gallium cluster with one degree of freedom in the self-consistency problem, the resulting SC curve is highly non-linear near the root, as shown in figure 2.3. The behavior near the solution is still nearly a step function, but the behavior away from the root is more complex. Indeed, it is nearly ideal for frustrating derivative-based root-finding methods as we will see.

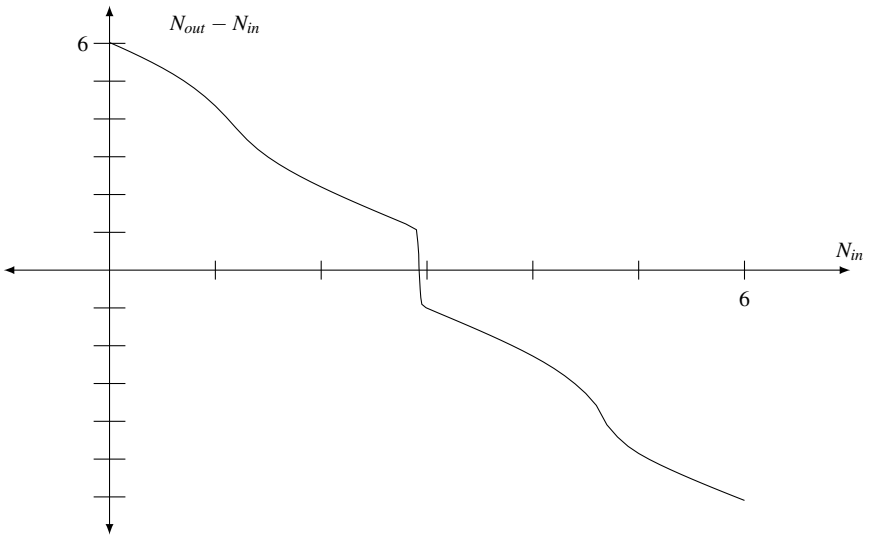


Figure 2.3: A self-consistency curve for Ga_8 D_4h.

2.4 – Higher Dimensions

Moving to higher dimensions introduces additional complexity. Let $f = \vec{N}_{out} - \vec{N}_{in}$. In two dimensions, we are looking for the fixed point of a function $f : \mathbb{R}^2 \to \mathbb{R}^2$. If we consider each component of $f = (f_1, f_2)$ as a separate function, $f_1 : \mathbb{R}^2 \to \mathbb{R}^1$ and $f_2 : \mathbb{R}^2 \to \mathbb{R}^1$, then we examine each as a three-dimensional surface above the domain. Then $z = f_1(x, y)$ and $z = f_2(x, y)$ can be plotted to find the points where $z = 0$.

Consider a 5-atom cluster composed of boron and nitrogen, B_2N_3 C_2v. The resulting self-consistency problem can be reduced from five unknowns to two unknowns using charge conservation and exploiting symmetry. Thus $\vec{N}_{out} - \vec{N}_{in}$ forms a function from $\mathbb{R}^2$ to $\mathbb{R}^2$. Breaking this into f_1 and f_2, we create heat plots for each. In these heat plots in figure 2.4 and figure 2.5, the domain $[0, 8] \times [0, 8]$ is plotted with various colors representing the value of $z_1 = f_1(x, y)$ and $z_2 = f_2(x, y)$. The x and y axes represent the number of electrons in the two-dimensional input vector $\vec{N}_{in}(x, y)$ and the color of the plot displays the change in the number of electrons in each component of the output vector, $z_1 = (N^1_{out} - N^1_{in})(x, y)$ and $z_2 = (N^2_{out} - N^2_{in})(x, y)$.

The surfaces are far from flat, but they appear to be monotonically decreasing, i.e. z_1 decreases monotonically with increasing N^1_{in} for every value of N^2_{in} and z_2 decreases mono-

tonically with increasing N_{in}^2 for every value of N_{in}^1. This is a feature we will exploit to slightly reduce the number of calculations needed to find the root.

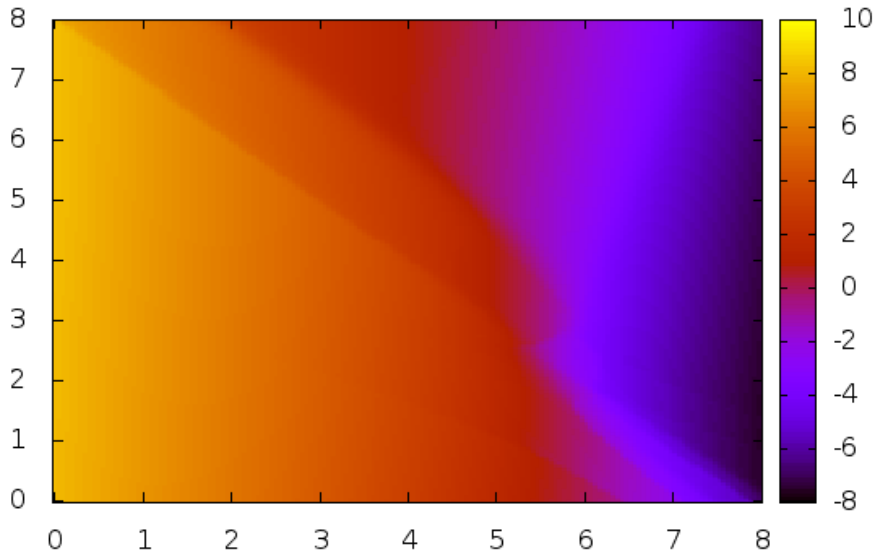


Figure 2.4: SC heat plot for dimension one of B_2N_3 C_2v.

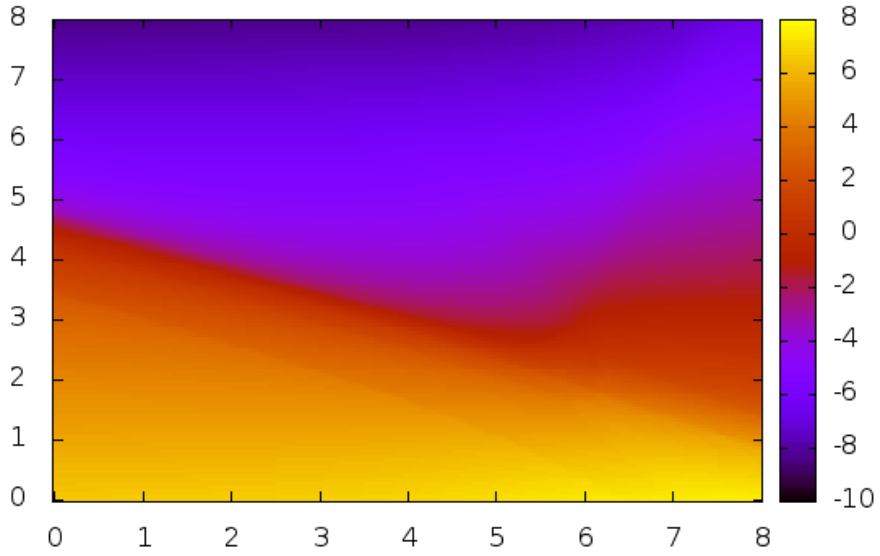


Figure 2.5: SC heat plot for dimension two of B_2N_3 C_2v.

To better visualize the solutions to two-dimensional SC problems, isocline plots are often helpful. In an isocline plot, a curve is plotted showing all values where z is equal to some desired value. For our purposes, we want to plot all values where $z_1 = 0$ and where

$z_2 = 0$. In the heat plots, this would be the curve where the plot is a medium red color, corresponding to $z = 0$.

Since these are now just curves in the same plane, we can plot both on the same axes. The self-consistent solution is then the point of intersection of these two curves. This is where $z_1 = z_2 = 0$ and thus $\vec{N}_{out} - \vec{N}_{in} = (0, 0)$.

Computationally, these isocline plots were constructed by fixing one dimension and generating a solution for the root in the other dimension. For example, fix $N^1_{in} = 2$ and solve for y^* where $z_2 = (N^2_{out} - N^2_{in})(2, y^*) = 0$. Then $(2, y*)$ is plotted in the isocline. The first dimension is incrementally varied throughout its domain to produce a curve representing the fixed points of the function f_1. The second dimension is then varied to plot the fixed points of the second function f_2. An example isocline plot is given in figure 2.6. While it

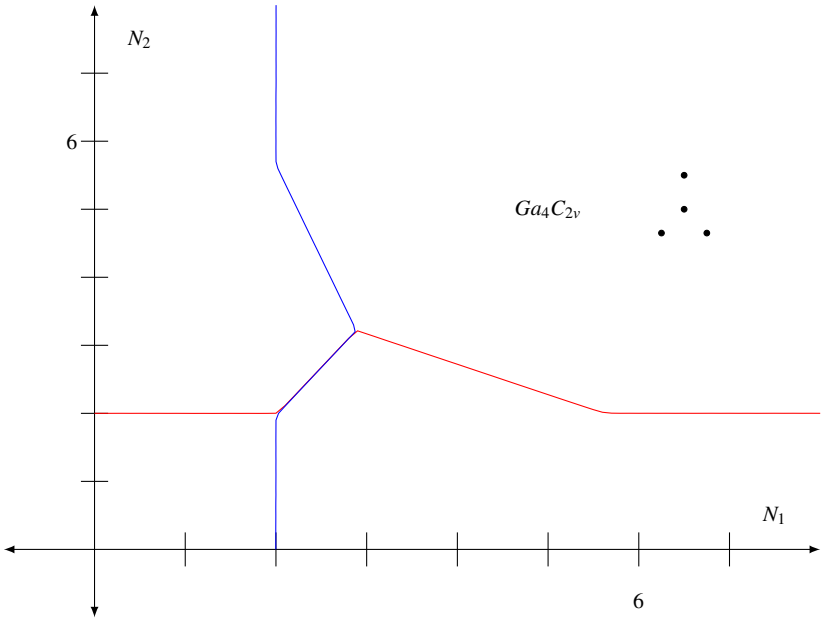


Figure 2.6: Isocline plot for Ga_4 C_2v.

appears from this plot that the uniqueness of the solution has been lost, a closer examination reveals that the single root is preserved. Figure 2.7 illustrates a closer look at another isocline, this time for the Ga_7 C_3v cluster. The solution was also found numerically through the iterative process and plotted with an $\times$ in the figure. Isoclines with this behavior are

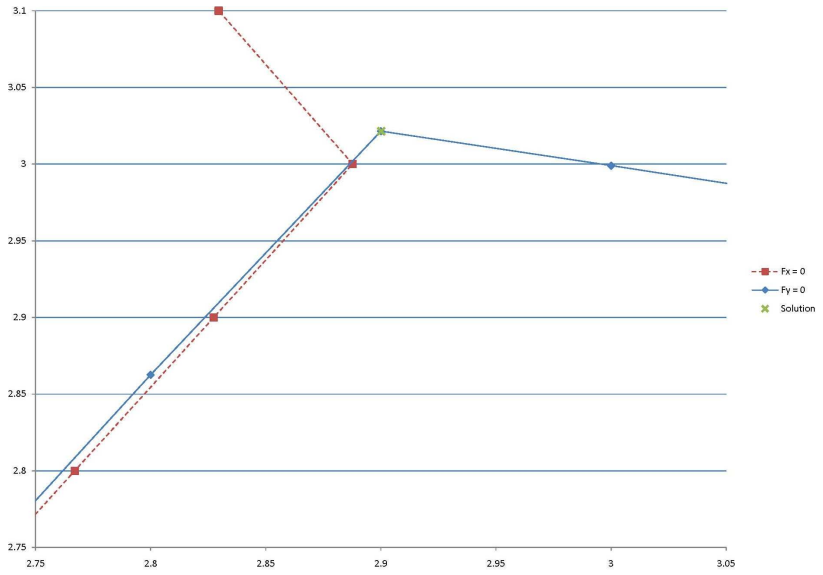


Figure 2.7: Isocline plot for Ga_7 C_3v.

typical for two-dimensional self-consistent systems in the author's experience.

2.5 – Derivative Methods

Newton's Method is the prototypical derivative method. At an initial point, the derivative is calculated (or Jacobian matrix in more than one dimension) and the next point of evaluation is derived from the direction of the derivative. Newton's method requires that the function being evaluated is continuous with continuous first derivative. Since our SC curves are often approximately discontinuous, Newton's method may not be the best choice.

In more than one dimension, Newton's method requires the inversion of an $n \times n$ matrix. Broyden's method eliminates this inversion and modifies the calculation to speed up computation time, but it also requires the function to have continuous first derivatives. Many other methods have been developed to reduce calculation time and to improve reliability [13], [14], but all require continuous first derivatives. Some hybrid of these derivative methods, such as Brent's Method, is usually employed for multi-dimensional root finding, including self-consistency [15], [16].

As an example of the failure of derivative methods in self-consistency algorithms, consider the self-consistency curve for the Ga_8 D_4h cluster given in figure 2.8. If given an initial point a on the curve, we calculate the derivative and use Newton's method to follow to point

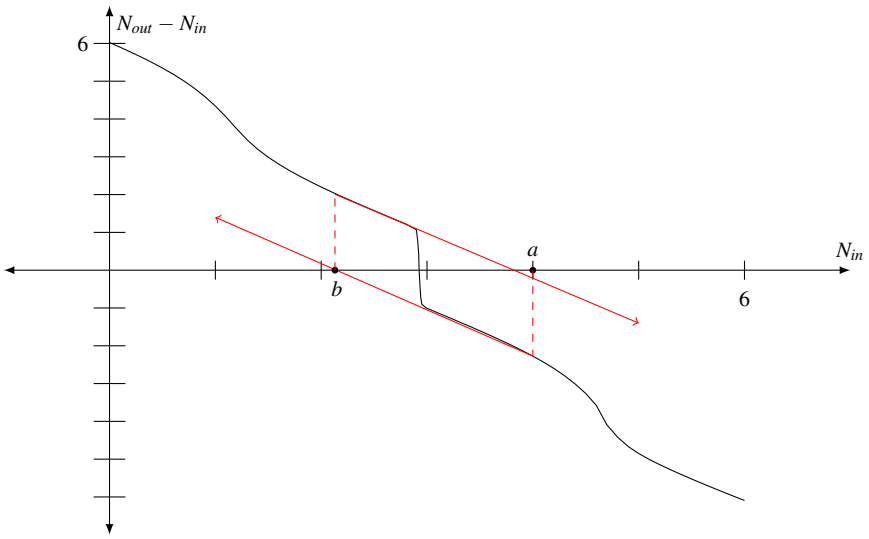


Figure 2.8: Charge Sloshing for Ga_8 D_4h.

b. Applying Newton's method at point b, the derivative leads us back to a point near a. This

system using Newton's method will oscillate between two or more points, never reaching the root. When this occurs in the context of the self-consistency problem, this phenomenon is known as "Charge Sloshing" due to the pattern of back-and-forth movement of the electrons in the system between a few points in the domain. Thus, derivative methods often fail to converge to a solution when applied to the self-consistency problem, even when a root is known to exist.

2.6 – Generalized Bisection

An efficient form of the bisection method generalized to higher dimensions was first described by B. Kearfott [17] and refined by M.N. Vrahatis [18], [19]. An excellent description of the Generalized Bisection and the associated issues was given by G. R. Wood [20]. Building on the research in degree theory, this method removes the need to compute the topological degree of the system (the number of times a function crosses zero along a curve), instead opting to remove the certainty of finding a root in order to greatly reduce computation time. It was later shown to provide solutions in a variety of problems where derivative-based methods had failed [21]–[23].

To develop the Generalized Bisection method, first consider the one-dimensional bisection method. The bisection method is a root finding algorithm for functions $f : \mathbb{R}^1 \to \mathbb{R}^1$. If for some interval $[a, b]$ in the domain of f, the product $f(a)f(b) < 0$ and $f \in \mathcal{C}^0[a, b]$ then the Intermediate Value Theorem guarantees that a root of f exists in $[a, b]$. There are less requirements on f, only that f be continuous in the region of interest and f must attain opposite signs on the boundary. The highly non-linear behavior of the SC curves at the root do not cause the bisection method to fail. This is largely because most of the computation for the algorithm is performed in the domain of the function. For the self-consistency problem, the domain ($\mathbb{R}^n \times \mathbb{R}^n$) is smooth and continuous.

Moving to higher dimensions poses challenges. Begin by considering the simplest domain in two dimensions, a rectangle. While degree theory guarantees a root in the domain

if the component functions change sign only once on the boundaries, this requirement is nearly impossible to check numerically. Attempting to check this requirement would also be very computationally expensive. Therefore, we proceed with the assumption that the signs change only once along the edge and develop a routine for when this assumption fails. We will also make the assumption that the function is continuous on the entire domain. In the context of the self-consistency problem, this assumption is believed to be valid.

Consider the signs (positive or negative) of the component functions on the corners of the region. If the two-dimensional root lies within the box and the component functions have exactly one root on each side, we expect two opposite corners to have opposite signs for the two component functions f_1 and f_2 as shown in figure 2.9. There will be one pair that is positive-positive and one that is negative-negative. The assumption is that f_1 and f_2 change only once on the edge if the endpoints are of opposite sign and do not change if the endpoints are of same sign. It is recognized that this may not be true, but a relaxation procedure (described below) will be performed when this assumption breaks down. In figure 2.9, f_1 will have one root along the top and bottom edges while f_2 will have one root along the left and right edges. As we will see, this is generally the case for the self-consistency problem.

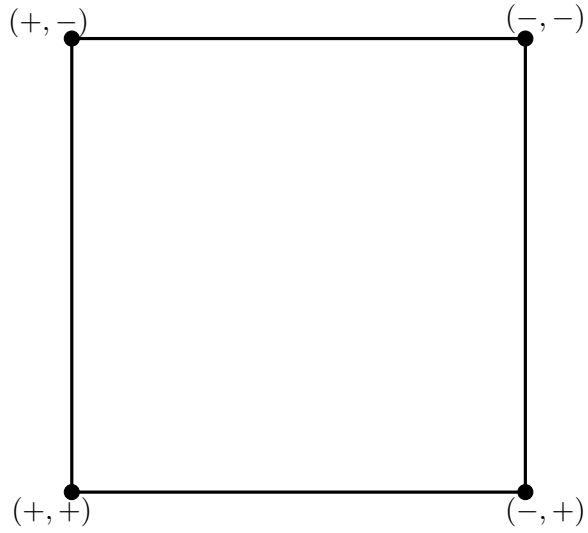


Figure 2.9: Initial domain for two-dimensional Generalized Bisection.

To proceed with the Generalized Bisection, we first choose a side and test its midpoint. If the functions at the midpoint are found to have the same sign as one of the endpoints of that side, that endpoint is removed and the midpoint becomes the new corner point representing that sign configuration. We then proceed to the next edge (or line segment in higher dimensions) and test its midpoint. Continuing with this procedure reduces the size of the domain where we believe the root exists. In figure 2.10, we see the first five steps of Gen-

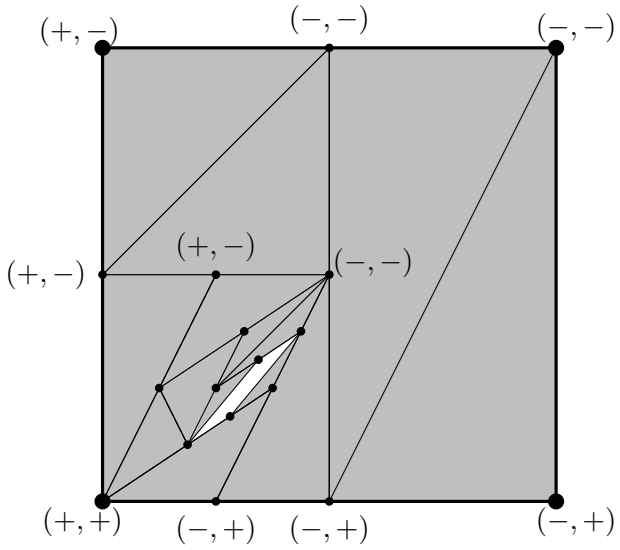


Figure 2.10: Two-dimensional Generalized Bisection.

eralized Bisection for the Ga_4 C_2v cluster. The isocline plot for this cluster was plotted in figure 2.6. The shaded areas are regions of the domain that have been excluded. Notice how the unshaded region matches the part of the isocline plot (figure 2.6) where both isoclines run together. In general, Generalized Bisection works well up to this point.

The Generalized Bisection continues until the desired accuracy is achieved. While this may be changed in the program, it is set to 10^{-10} by default. Thus, when the norm of $\vec{N}_{out} - \vec{N}_{in}$ is sufficiently small,

$$\|\vec{N}_{out} - \vec{N}_{in}\| < 10^{-10},$$

the root $\vec{N}_{out}$ is passed to the next step in the SCED-LCAO system. This same criterion is

also employed for Broyden's Method.

2.7 – An Example Using Generalized Bisection

Consider for example the function $f : [0,1] \times [0,1] \to \mathbb{R}^2$ defined by

$$f(x,y) = \left(\frac{\pi}{12} + x - 2y, xy + x - \frac{1}{2}\right).$$

Here $f_1 = \frac{\pi}{12} + x - 2y$ and $f_2 = xy + x - \frac{1}{2}$. Since we have the algebraic form of the function, we can solve this system algebraically. Solving $f_1 = 0$ and $f_2 = 0$ yields the equations for the isoclines, $y_1 = \frac{\pi}{24} + \frac{1}{2}x$ and $y_2 = \frac{1}{2x} - 1$. The equations for the isoclines can then be used to find the steady state solution of the function by equating $y_1 = y_2$. We find $(x^*, y^*) \approx (0.3787, 0.3203)$. We can check our answer by plugging back into the original function to find $f(x^*, y^*) = (0,0)$. These isoclines and the steady state solution are plotted

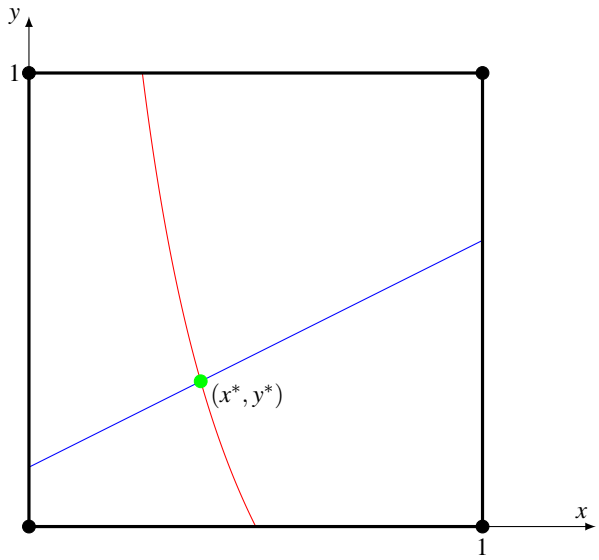


Figure 2.11: Example of Two-dimensional Root Finding.

in figure 2.11.

If we instead use Generalized Bisection to find the solution, we begin by finding the signs of the functions f_1 and f_2 at each of the four corners of the domain. We check to make sure that each corner has a unique pair of signs and that $(-,-)$ and $(+,+)$ are not connected

by an edge. In SCED-LCAO, this is a direct result of the monotonicity of the component functions. Next we find the signs of the functions at the mid-point of one of the edges. It does not matter which edge is chosen as the initial edge, so we will start on the edge along the y-axis. Here we find the signs of the functions at $(0, 0.5)$ are $(-,-)$. Thus we remove the corner $(0, 1)$ since it has the same set of signs. The result is a set of four corner sharing the same set of signs for the component functions as the initial corners.

We then proceed to the next edge. The algorithm used in SCED-LCAO chooses edges in a clockwise manner, although it need not choose them in that order. The next edge would then be the newly created edge from $(0, 0.5)$ to $(1, 1)$. We evaluate the signs at the edge midpoint $(0.5, 0.75)$ to find $(-,+)$. Thus, we replace $(1, 1)$ with $(0.5, 0.75)$ and proceed to the next edge. This is then repeated until the norm of the function at an edge midpoint is less than the desired accuracy A, $|f(x_k, y_k)| < A$. The initial bisection and several following steps are shown in figure 2.12.

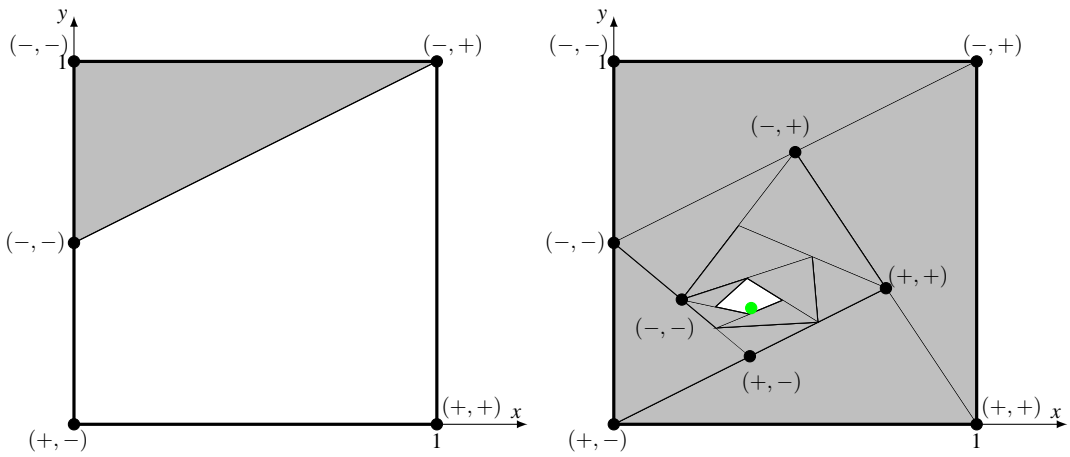


Figure 2.12: Example of Two-dimensional Generalized Bisection.

2.8 – Generalized Bisection Relaxation

However, there are some issues that need to be addressed. First, we would like to avoid the situation where the acceptable region of the domain becomes very narrow or distorted. Repeated attempts to force uniform convergence rates on all sides were met with failure. This

is what prompted the creation of the isocline plots. The isocline in figure 2.6 makes it clear that this is a problem inherent to the SC solution. Secondly, we would like to maintain the quadrilateral shape of the acceptable region. This will ensure that the acceptable region does not become a triangle to within computer accuracy (which does happen), thus destroying our ability to further bisect. This was successfully enforced by skipping over any line segment that is 50 times smaller than the longest line segment.

If the functions at the midpoint are found to have a different sign configuration then either of the endpoints of the line segment, we cannot remove either end point. It's likely at this point that multiple functions are changing sign along this line segment. This implies that our assumption that the root is contained in the acceptable region is false. Therefore, we pursue a relaxation of the acceptable region, expanding its size to encompass the root once again. As shown in the example in figure 2.13, the sign configuration of the midpoint is matched with the (unique) corner point of the acceptable region which has the same sign configuration. Then the acceptable region is expanded by "pushing" the midpoint away

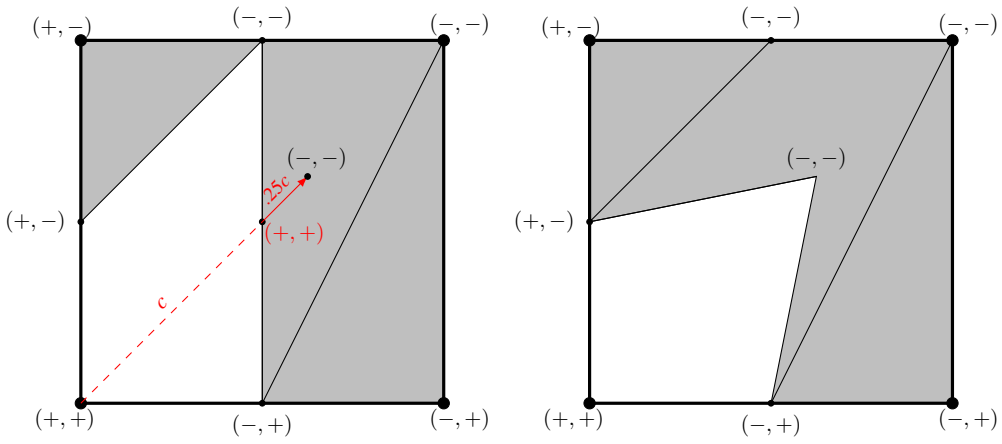


Figure 2.13: Relaxation in Generalized Bisection.

from this corner point. A straight line is drawn from the matching corner point to the tested midpoint. A new point is created along this line 25% further away from the matching corner point. The new point is tested and if found to be matching one of the endpoints, it replaces the matching corner point and Generalized Bisection continues as before. If the new point

still does not match one of the endpoints in sign configuration, the point is further moved further along the line by 50% and tested again. The point is moved in this direction until its sign configuration matches an adjacent corner point and Generalized Bisection continues with the next line segment.

2.9 – Initial Vector

The importance of the initial guess for derivative methods cannot be understated. An initial vector near the solution will allow for very fast convergence, while starting points further away can lead to very slow convergence or even Charge Sloshing. The Generalized Bisection method is also sensitive to the initial point. I modified the algorithm to try to create an initial box in the domain which is as small as possible. It takes the initial point and adds a small $\delta = 10^{-3}$ to each coordinate. Then the corner points of this region are tested to verify that the signs match the required signs for Generalized Bisection. If successful, the algorithm proceeds with this small box. If unsuccessful, the box is enlarged by a factor of 10 until the required signs are found. Therefore, either type of method can be greatly aided by a wise choice of initial vector.

Initially, the program simply sets the initial guess to be the number of valence electrons on the neutral atom; i.e., 3 for Gallium, 5 for Nitrogen, etc. This method is sufficiently close to the solution for most homogeneous systems so as to provide an excellent initial vector. However, with heterogeneous systems charge transfer in the SC solution is a key feature of molecular bonding. Therefore self-consistency is essential to calculate accurate solutions. Hence using the electrons count for neutral atoms is unlikely to yield an initial vector that is near the self-consistent solution for heterogeneous systems.

The first step in improving the initial guess requires a good understanding of the algorithm used in fitting. In figure 2.14, the hierarchical structure of the fitting procedure is outlined. The parameters for the SCED Hamiltonian are fit to a set of reference values using a gradient method. The reference values are a list of geometry and energy values obtained for

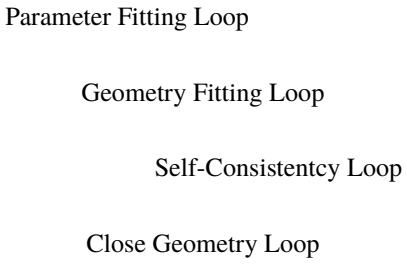


Figure 2.14: Structure of Fitting Procedure.

clusters using DFT methods, ab initio, experimental, or other methods. They also include information for homogeneous bulk structures, such as relaxation curves and band energy data. For each reference value, the fitting procedure uses the current set of parameters to reproduce the value.

For a specific cluster, the geometry is varied to find the set of geometric values that provide the minimum energy (produced with SCED and the current parameter set). Once the minimum energy for the cluster is determined, the geometric values and energy value are compared with the reference values and a weighted "residual" is calculated for each reference value. This residual represents a measure of the percentage difference between the reference value and the calculated value. All of these "local residuals" are averaged together to produced a "global residual" for that parameter set using a square root of the sum of the squares. The fitting procedure can be viewed as a functional operating on the system to produce a single positive value which represents the error in the parameter set; i.e., the system residual becomes a functional $R : \mathbb{R}^x \rightarrow \mathbb{R}^{1,+}$, where x is the number of parameters, typically 24. The parameters are fit to the reference values in such a way as to minimize this residual.

As can be seen from the outline in figure 2.14, the self-consistency procedure is performed very many times for even a single set of parameters. For this reason, the self-consistent solution needs to be produced in as few steps as possible. Because the solutions are being used for two additional levels of optimization, accuracy cannot be sacrificed to

Initial Vector	SC Steps for 1 Parameter Loop
Neutral Atom	824,197
Previous Solution	63,476
Known Solution	59,966

Table 2.1: Comparison of Initial Vectors.

speed the calculations. Attempts to raise the accuracy threshold above 10^{-10} caused the geometry optimization to fail.

However, the hierarchy of optimization can be exploited to garner initial guesses very near to the solution. The geometry optimization uses a gradient method which calculates the derivative (or Jacobian) for the geometry values using a double difference method. The very small changes in geometry used to calculate the derivatives have very little effect on the self-consistent solution. Therefore, for any given cluster, we use the previous SC solution as the initial value for the current calculations. This resulted in huge increases in efficiency, as can be seen from table 2.1. This table lists the resulting number of SC steps used during one parameter loop for a typical fitting procedure. Here we are fitting 25 parameters to 132 reference values for a heterogeneous set of clusters including Boron, Carbon, and Nitrogen. Using the previous SC solution for each cluster results in a 92% reduction in the number of SC steps used. This situation still uses the neutral atom electron count for the initial vector each time a new geometry optimization is begun. Replacing this with a known solution for each cluster (with similar, but not identical parameters) results in a further 1% reduction in steps, as shown in the third row of table 2.1. The reductions shown are for a combination of Broyden's Method and the Generalized Bisection Method described below.

2.10 – Comparison of Methods

Three methods were employed to find the root in the SCED-LCAO self-consistency problem. The initial method was a simple Newton method using the inverse of the Jacobian matrix. This was replaced with the faster and more accurate Broyden's method. Broyden's method and the Generalized Bisection method were compared, as well as a hybrid method

which made use of both.

We note with some displeasure that the guaranteed convergence seen in the one-dimensional bisection method is lost when moving to the Generalized Bisection method. In this regard, Generalized Bisection is equal to Broyden's Method, which also does not guarantee a solution. Given the much less stringent continuity requirements, Generalized Bisection often converges when Broyden's Method will not. However, Generalized Bisection is much slower than derivative methods. While Broyden's method can converge on the order of $2n$, Generalized Bisection converges on the order of 2^n, where n is the size of the system under consideration (the length of $\vec{N}$ for SCED-LCAO self-consistency). This makes Generalized Bisection a particularly poor choice for systems with higher self-consistency degrees of freedom. This suggests that we try Broyden's Method first and then switch to Generalized Bisection should Charge Sloshing be detected. This method was employed for the final version of the SC algorithm.

CHAPTER 3

GALLIUM NITRIDE

SCED-LCAO is a semi-empirical method, meaning that its predictive power relies on a set of parameters that are acquired from empirical data. In the case of SCED-LCAO we use the fitting procedure described previously in this work to develop these parameters. This fitting procedure is based on a set of test clusters, which are described by their properties. These properties include the geometrical degrees of freedom within the given symmetry and the total energy of the cluster. In addition to these clusters, we also use data from bulk phases and the dimer energy curve.

Unfortunately, very little experimental data exists for small clusters of atoms beyond the dimer so we will use a very accurate ab initio calculation to obtain these properties. We have chosen Gaussian™ to perform calculations in the Density Functional Theory (DFT) formalism. In DFT, one of the most common forms of the exchange-correlation energy functional is the Becke, 3-parameter, Lee-Yang-Parr (B3LYP) treatment. This hybrid approach was introduced by Axel Becke in 1993 [24] and incorporates a portion of the exact exchange from Hartree-Fock theory with the exchange and correlation from other sources (ab initio or empirical). The result is an improvement in many of the predicted molecular properties such as atomization energies, bond lengths, and vibration frequencies.

The form of the functional must be paired with a set of basis functions to allow for computation. Gaussian™ provides an array of different basis sets, although not all of them are compatible with Gallium. We tried several basis sets, but eventually settled on the Dunning correlation-consistent triple zeta basis set **cc-pVTZ** [25]. This basis set "has had redundant functions removed and has been rotated in order to increase computational efficiency." [26] It also includes polarization functions, which are additional functions added to the minimal

basis set to model unfilled atomic orbitals. For instance, for Gallium the **cc-pVTZ** basis set includes six *s*-type orbital functions, five *p*-type orbital functions, three *d*-type orbital functions, and one *f*-type orbital function for polarization. We also decided to include diffuse functions, which aid in calculating long-range interactions, such as Van der Waals forces. In Gaussian™, the addition of diffuse functions is denoted by the **aug** prefix. All calculations included in the database were thus performed with **aug-cc-pVTZ** basis sets for consistency. This represents the current state of the art in DFT calculations.

Typically when the geometry of the structure was unknown, we would employ a "lighter," and therefore faster computationally, basis set such as **LanL2DZ** or **STO-3G** to arrive at an approximate set of properties and then use **aug-cc-pVTZ** to verify the stability of the structure and fine-tune the properties. This provides a catalog of cluster geometries and energies against which we fit the SCED parameters.

Bulk data often exists for many forms of periodic systems. For homogeneous systems we will also leverage this crystal structure information to improve our parameters. We use the Vienna Ab-Initio Simulation Package (VASP™) to calculate energy curves as the geometry is varied.

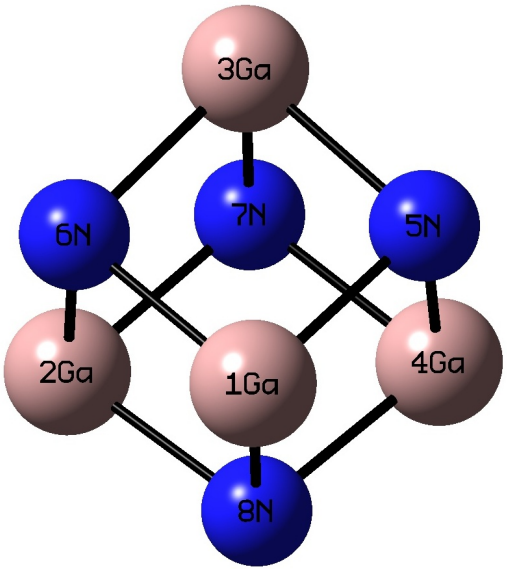


Figure 3.1: Ga_4N_4 D_3d cluster found with Gaussian™.

Residual Scaling

When searching for an optimal set of parameters, we need to have a measure of how well the current parameter set results in calculations which match the reference values. This is accomplished through the use of a simple residual. The difference between the calculated value T^C and the reference value T^R is calculated for each fitting property and the residual R is given for N atoms as

$$R^2 = \frac{1}{N}\sum_{i=1}^{N}(T_i^C - T_i^R)^2.$$

Unfortunately, this formula will result in weighting all of the reference values equally. Since a typical set of reference values consists of both geometries and energies which are on vastly different scales, this will result in some reference values becoming insignificant in the residual calculation. A possible solution would be to divide the differences by the reference values to provide a percentage difference in each value, but a more general approach is to introduce a scaling factor S_i as discussed in [27] for each reference value,

$$R^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{T_i^C - T_i^R}{S_i}\right)^2.$$

Now different types of data can be scaled to allow for equal weighting or perhaps some other weighting system better suited to the system.

One additional problem encountered with the geometry is that the choice of coordinate system can greatly affect the residual. Consider the simple cluster with three atoms and $D_{\infty h}$ symmetry shown in figure 3.2.

If we set the origin of the coordinate system to be at an end atom, the two geometric degrees of freedom will be different from a system with the origin at the center atom. This is represented in the figure by the solid and dashed arrows. If we take the residual to be percentage changes, we obtain two different results for the residual. Assume that the measurements are as in the figure and that atom 3 is calculated to be at 3.2 (with reference value

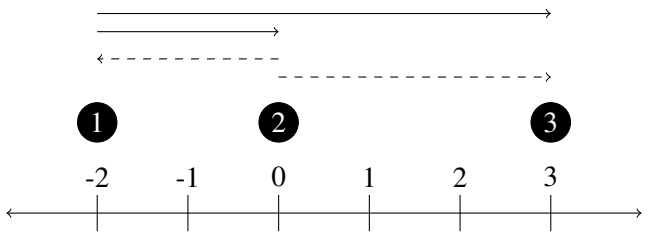


Figure 3.2: Three-atom cluster with $D_{\infty h}$ symmetry.

3.0). The residual for the position of atom 3 in the dashed coordinate system is

$$R^2 = \left(\frac{T_i^C - T_i^R}{S_i}\right)^2 = \left(\frac{3.2 - 3.0}{3.0}\right)^2 = 0.00\overline{4}$$

while the residual in the solid coordinate system is

$$R^2 = \left(\frac{T_i^C - T_i^R}{S_i}\right)^2 = \left(\frac{5.2 - 5.0}{5.0}\right)^2 = 0.00016$$

Certainly we want the residual to be based on the difference in the position, but independent of the coordinate system used.

This problem is avoided with the energies since the energies are all relative to some fixed value, typically the ground state energy of a single atom. So our first inclination is to use one standard scaling factor for all reference geometries. This would give equal residuals in the example above. The most obvious choice for a scaling factor would be the bond length of the dimer (for homogeneous calculations).

However, given the physics of the situation we may want to weight some geometric references more than others. For instance, we may desire to capture the angular flavor of p-bonding or perhaps focus only on inter-atomic distances. We would then choose to weight differences in directions that are tangential to the inter-atomic distances more heavily than in differences in the distances between the atoms. A reasonable compromise is to use some set reference value for each type of distance. After careful consideration, we chose five reference values for five types of measurements. These are given for each element discussed below.

The residual is then multiplied by 10 to yield a scale that is more readable.

3.1 – Gallium

Neutral Gallium atoms contain 31 electrons, making it very expensive computationally to form clusters. A typical calculation of a high-symmetry cluster consisting of eight Gallium atoms will take approximately a week of computer time. For this reason, we created our database of fitting properties using smaller clusters and supplemented this with energy curve data from the dimer and bulk phases.

Database Generation

The energy curve for the Ga_2 dimer is shown in figure 3.3. This was created by varying the separation distance of the two atoms using a small mesh of 0.1 Angstroms and calculating the resulting energy in Gaussian™.

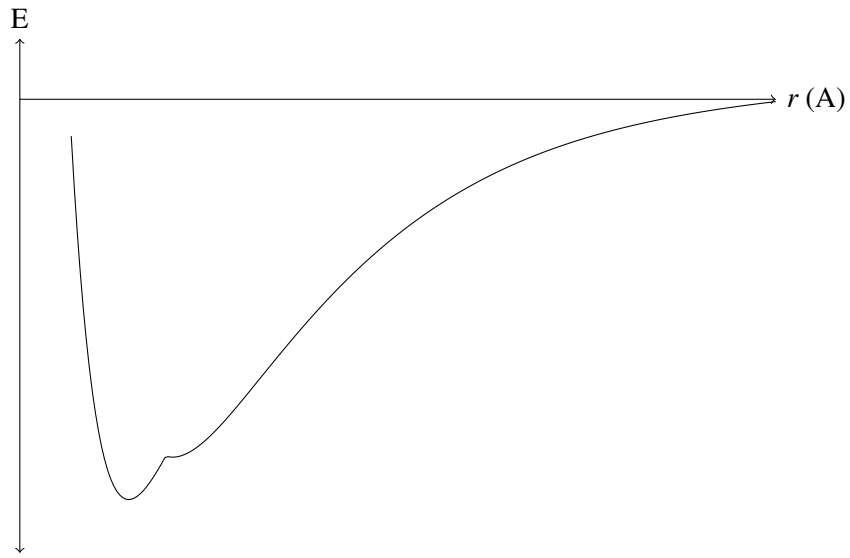


Figure 3.3: Gallium dimer energy versus atomic separation.

Gallium bulk data was included for both the alpha α and beta β phases. The most common solid form of gallium is the α-gallium phase. The fitting program allows for the inclusion of a simple expansion ("breathing") mode energy curve where all degrees of freedom in the lattice are increased by the same scaling factor. The structure for the unit scaling

(scale=1.00) was based upon the experimentally-observed lattice vectors. This was scaled by multiplying all three dimensions of the unit cell vectors by 0.98, 0.99, 1.01, 1.02, etc. We then fit the parameters for gallium against the resulting energies from these VASP™ calculations. This energy curve from VASP™ is given in figure 3.4, along with the results using the final SCED parameters for gallium.

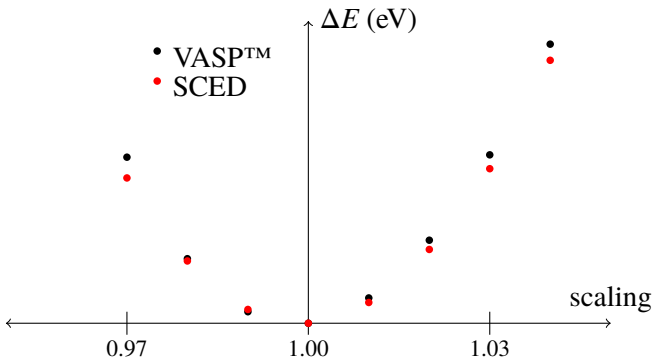


Figure 3.4: β-gallium energy versus lattice scaling.

For clusters, a thorough search of every symmetry was performed for clusters consisting of up to four gallium atoms. Convergence was tested for each degree of freedom by testing the cluster in Gaussian™ starting both above and below the convergence value. Since Gaussian™ forces clusters to retain the input symmetry, test clusters were also converged after the symmetry was broken by moving an atom slightly out of position. For example, in figure 3.5 there are two degrees of freedom in the Ga_3 C_{2v}a cluster, namely the distance from 1Ga to 3Ga and the horizontal distance from 2Ga to the line connecting 1Ga and 3Ga. Each of these was tested above and below the equilibrium value.

Stable clusters ranging in size from five to seven atoms were found using high-symmetry geometrical configurations. Others were suggested by literature [28]–[31]. One cluster of eight gallium atoms was considered, the D_{2h} structure in figure 3.6. This structure has only three degrees of freedom and will therefore converge quickly enough to allow for parameter fitting.

Once a collection of stable structures was generated, a subset was selected to comprise

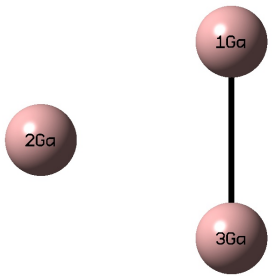


Figure 3.5: Ga_3 C_{2v}a

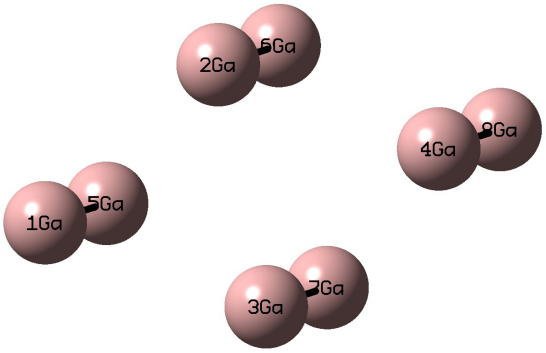


Figure 3.6: Ga_8 D_{2h}

the database properties. Preference was given to the lowest energy clusters for each stoichiometry (the number of each element present), but others were also included. Each cluster was tested at the three lowest spin multiplicity states and the lowest energy of those three was used. Since SCED-LCAO does not incorporate spin in this version, we included only those clusters with minimal spin contamination.

Fitting Procedure

The choice of initial parameters is extremely important. We are attempting to perform a global optimization on a function of 24 variables, i.e. we wish to find the values of the 24 parameters that result in a minimum residual. The domain of some variables is limited, e.g. we expect a_K to be positive, but less than one half. Other parameters have a much wider

range of possible values. No method exists to arrive quickly at the global minimum residual, so we are employing a random search pattern (see [27]). A poor choice of initial parameters could result in years of additional calculations.

The initial gallium parameters were chosen as shown in table 3.1. All units are in angstroms and electron volts. The initial overlap parameters are generated by producing overlap curves in Gaussian™ for the Ga_2 dimer and using a fitting algorithm to arrive at parameters to match these curves with the form of our overlap functions. We used the freely available plotting program XMGrace™ to calculate these fits. The values for ε_s and ε_p are standard results for the Hartree-Fock atomic orbital energies for the 4*s* and 4*p* valence orbitals from *Atomic Structure Calculations* by Joseph B. Mann [32]. These were simply doubled for the initial ε' values. It is important that these be set at least 50% below the values for ε since it is the difference between them that allows for bonding in SCED-LCAO. The value for A_K will typically vary from -0.5 to 0.5, so any value near zero will suffice. The inter-atomic Coulomb repulsion term U is given by Walter A. Harrison in *Elementary Electronic Structure* [33]. This value was fixed initially, but allowed to vary slightly once the residual was less than 80. The V_N and $_Z$ terms were initially set to zero. However, if values from another column **V** element were available, they would be a better choice for the starting parameters. The distance parameter d_N was set to the dimer bond length, as were the reciprocal length parameters w_{es} and w_{ep} since we expect the effect to occur on that distance scale. The other parameters for the W term were initially fixed at zero and then allowed to vary once the residual was below 100 since we expect them to be fine-tuning values. Finally, the energy constant value is fixed at *E_const* to allow the reference properties to be given in Hartrees as output from Gaussian™ while the SCED-LCAO parameters and output will be in electron volts.

We decided to use the scaling weights for homogeneous Gallium shown in table 3.2. The energies for the reference gallium clusters vary from -0.024 for the dimer to -0.061 for the Ga_8 D_{4h} cluster. Therefore, a scale value of 0.045 is reasonable. The bulk energy values vary across a large range, with the minimum set to exactly zero. They also don't vary evenly

Name	Value	Source
$A_{ss\sigma}$	1	XMGrace fit to dimer overlap; fixed.
$B_{ss\sigma}$	-0.0798843	XMGrace fit to dimer overlap.
$\alpha_{ss\sigma}$	1.472260	XMGrace fit to dimer overlap.
$d_{ss\sigma}$	1.977800	XMGrace fit to dimer overlap.
$A_{sp\sigma}$	0	XMGrace fit to dimer overlap; fixed
$B_{sp\sigma}$	0.484434	XMGrace fit to dimer overlap.
$\alpha_{sp\sigma}$	1.719130	XMGrace fit to dimer overlap.
$d_{sp\sigma}$	1.906810	XMGrace fit to dimer overlap.
$A_{pp\sigma}$	1	XMGrace fit to dimer overlap; fixed.
$B_{pp\sigma}$	-0.788506	XMGrace fit to dimer overlap.
$\alpha_{pp\sigma}$	2.112320	XMGrace fit to dimer overlap.
$d_{pp\sigma}$	2.337990	XMGrace fit to dimer overlap.
$A_{pp\pi}$	1	XMGrace fit to dimer overlap; fixed.
$B_{pp\pi}$	0.0566966	XMGrace fit to dimer overlap.
$\alpha_{pp\pi}$	1.683180	XMGrace fit to dimer overlap.
$d_{pp\pi}$	1.27176	XMGrace fit to dimer overlap.
ε_s	-11.55396	*Atomic Structure Calculations* [32]; fixed.
ε_p	-5.673576	*Atomic Structure Calculations* [32]; fixed.
ε'_s	-23.10792	twice ε_s.
ε'_p	-11.34715	twice ε_p.
A_K	0.000001	≈ 0.
U	6.701734	*Elementary Electronic Structure* [33]; fixed.
B_Z	0	Start at zero.
A_N	0	Start at zero.
B_N	0	Start at zero.
α_N	0	Start at zero.
d_N	2.52000	Set to dimer bond length.
w_{as}	0	Start at zero.
w_{es}	2.52000	Set to dimer bond length.
w_{ap}	0	Start at zero.
w_{ep}	2.52000	Set to dimer bond length.
E_const	14.3996	For Angstroms and electron volts; fixed.

Table 3.1: Initial SCED-LCAO parameters for gallium.

Name	Value	Source
Bond	2.6 Å	Bond length
Half-bond	1.3 Å	Half bond length
Angular	2.0 Å	Lengths directly affecting bond angle.
Cluster Energy	0.045 Har.	Energy per atom of clusters
Bulk Energy	0.002 Har.	Energy per atom

Table 3.2: Scale weight factors for gallium.

due to the fact that the length scale was varied in DFT, but the property scale is based on the volume. Therefore, we use the DFT value for scaling the bulk, i.e. we scale by percentage.

The dimer energy curve and the dimer cluster property were used to begin the parameter search. However, once a rough estimate was achieved, these were removed from the reference properties. The dimer curve contains a cusp-like (non-differentiable) point where there is a transition in the lowest energy eigenvalue curve from DFT. The Ga_2 dimer cluster was also shown to have a high degree of spin contamination. It was decided that these features were not something we desired to capture in our model. The ultimate goal of SCED-LCAO is to quickly model systems of many atoms, so the behavior of a naked dimer is not something upon which we will focus.

Initially, the overlap curves were fixed, along with U and the W terms. Once the dimer had provided a rough estimate of the best parameters, we gradually added in additional clusters starting with two of the largest and a second small cluster. The goal is to allow the fitting code to run quickly by only computing a small number of clusters while capturing the behavior of the range of clusters in the database. Once the residual had reached a relatively low value of 100 and most of the clusters were included, we began to optimize the overlap curves, U, and W as well. This process was also expedited by optimizing only the overlap parameters or the only other half of the parameters at any given time, e.g. we would first optimize the non-overlap parameters, second optimize the overlap parameters, then repeat. Eventually, we arrived at a residual of 71.2. The resulting parameters for both Gallium and Nitrogen are given in table 3.3. The residuals for each fitting property can be found in the

Name	Gallium	Nitrogen
ε_s	-11.55396000	-26.23360000
ε_p	-5.67357600	-13.84240000
ε'_s	-15.89942562	-34.67487053
ε'_p	-9.78502345	-20.28141541
α_K	0.09070368	0.25788710
U	13.66059628	15.88403566
B_Z	1.57855854	3.84869042
A_N	-1.40923471	-3.50765478
B_N	-1.74623609	1.78176908
α_N	2.38478380	3.94697772
d_N	-0.04540130	0.57207437
w_{as}	-0.00445490	0.24230389
w_{es}	1.32305085	1.38831534
w_{ap}	-0.10498407	-0.40115070
w_{ep}	1.39086822	1.45510897
α mixing	0.63574873	
$B_{ss\sigma}$	-0.17126234	0.67554176
$\alpha_{ss\sigma}$	1.71417826	1.72525441
$d_{ss\sigma}$	1.66785074	0.24987968
$B_{sp\sigma}$	0.49124563	0.47210819
$\alpha_{sp\sigma}$	2.04314742	3.08262899
$d_{sp\sigma}$	1.68594570	1.27079590
$B_{pp\sigma}$	-0.61936298	-1.52123746
$\alpha_{pp\sigma}$	2.57254048	3.29562875
$d_{pp\sigma}$	2.36229832	0.96429255
$B_{pp\pi}$	-0.11258717	0.03734260
$\alpha_{pp\pi}$	1.69590206	3.69029550
$d_{pp\pi}$	1.29280440	0.74092947

Table 3.3: Final SCED-LCAO parameters.

appendices. In total, there were 35 gallium cluster properties and 16 bulk properties used.

During this fitting process, it is important to maintain a minimum gap between ε'_s and ε'_p. Since the difference between ε_s and ε_p is approximately 6 eV, we chose to enforce a minimum ε' gap of 6 eV while both values were allowed to vary.

3.2 – Nitrogen

Nitrogen poses a much more difficult problem. Nitrogen forms no known homogeneous molecular structures beyond the dimer N_2. Rather than try to form artificial clusters in DFT

– a task that is likely to fail – we chose to generate the Nitrogen parameters using a database of heterogeneous clusters containing Nitrogen atoms. In addition, the version of the fitting code used for this work does not allow for the inclusion of heterogeneous bulk in the database. Such heterogeneous bulk structures require a formulation that can capture long-range interactions. This is typically done through the use of an Ewald summation, which uses a Fourier transform to perform the calculations for long-range periodic forces. The version of the fitting code used here does not incorporate Ewald sums.

Since the ultimate goal is to model GaN systems, we created a database of clusters containing both gallium and nitrogen. For this nitrogen parameter generation, the gallium parameters are treated as being fixed. They are not allowed to vary from the values found in the previous section.

While nitrogen is a much lighter atom than gallium with only seven electrons in the neutral atom, there are still limitations on the size of the clusters that can be included in the database. Stoichiometries with many more nitrogen atoms than gallium tend to separate into individual N_2 pieces. Elemental formulas with many more gallium atoms than nitrogen would tend to over emphasize the gallium parameters, leading to little benefit for nitrogen fitting. Therefore, we chose a database that emphasizes stoichiometries with approximately the same number of gallium and nitrogen atoms. In this situation, computation time limits the size of the clusters to approximately twelve atoms – six nitrogen and six gallium.

Again, a thorough search of every symmetry was performed for clusters consisting of up to four atoms. Convergence was rigorously tested for each degree of freedom. The clusters found were also tested for convergence after the symmetry was broken. Some clusters ranging in size from five to ten atoms were found using high-symmetry geometrical configurations. Others were suggested by previous work [34]–[46].

In addition, several clusters were attempted by modeling a small chunk of the hexagonal gallium nitrogen bulk structure and allowing it to relax to a new geometry. Using this method, only the Ga_6N_6 D_{3d} cluster shown in figure 3.7 was found to be stable.

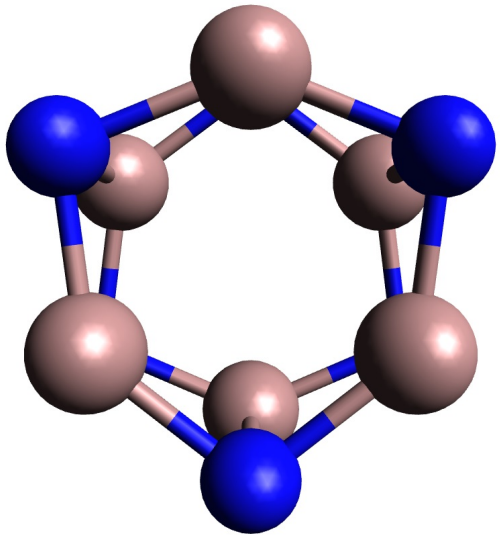


Figure 3.7: Ga_6N_6 D_{3d}

Once a collection of stable structures was generated, a subset was again selected to comprise the database properties. Preference was given to the lowest energy clusters for each elemental formula, but others were also included. We included more clusters with approximately equal numbers of gallium and nitrogen atoms in an attempt to focus on the interaction between the two atomic species. Each cluster was tested at the three lowest spin multiplicity states and the lowest energy of those three was used. Once again, we included only those clusters with minimal spin contamination.

Fitting Procedure

The initial nitrogen parameters were chosen in the same way as for gallium. The initial overlap parameters were generated by producing N_2 dimer overlap curves in Gaussian™ and using XMGrace™ to create parameters to match these curves with the form of the SCED overlap functions. The values for ε_s = -26.2336 eV and ε_p = -13.8424 eV were again pulled from Mann [32] and doubled for the initial ε' values. The inter-atomic Coulomb repulsion term U = 13.15 eV is given by Harrison [33]. This value was fixed initially, but allowed to vary slightly once the residual was less than 150. The V_N and V_Z terms were initially copied from the carbon values given by Yu [4].

Name	Value	Source
Ga-Ga bond	2.8 Å	Ga_2 bond length
Ga-N bond	2.0 Å	GaN dimer bond length
N-N bond	1.1 Å	N_2 bond length
Angular	2.0 Å	Lengths directly affecting bond angle.
Cluster Energy	0.10 Har.	Energy per atom of clusters

Table 3.4: Scale weight factors for gallium nitride.

There is more to consider when deciding on a set of scaling weights for gallium nitride systems. For example, the GaN dimer characteristic bond length is significantly different from the characteristic N_2 dimer bond length. Most of the scaling weights were set as the characteristic bond length for that interaction. However, the ring structures Ga_4N_4 D_{4h} and Ga_5N_5 D_{5h} are described by the ring radii. Therefore, the weights for those were set at the reference values for the radii. A list of the weights is given in table 3.4.

The N_2 dimer energy curve and the dimer cluster properties were not used in the parameter search. The lowest energy dimer occurs for multiplicity three and has a high degree of spin contamination. The same is true for the GaN dimer cluster.

We began by fitting to the largest cluster Ga_6N_6 D_{3d} and the smallest cluster Ga_3N_1 while holding the overlap fixed. Experience has shown that fitting to only a handful of clusters can lead to an unproductive search once more clusters are added. So we included every GaN cluster that would converge quickly.We allowed ε', A_K, B_Z, A_N, B_N, α_N, and d_N to vary, along with the mixing parameter for Gallium and Nitrogen $\alpha_{\text{Ga-N}}$.Then we gradually began filling in additional Gallium-Nitrogen clusters as the residual improved. Once most of the GaN clusters were included, we began to optimize the overlap parameters simultaneously with the V_N and V_Z parameters until all GaN clusters in the database were included. Finally, we also allowed the W terms to vary, as well as the U term.

Eventually, we arrived at a residual of 93.19. The resulting parameters and residuals for each fitting property are given in the appendices.

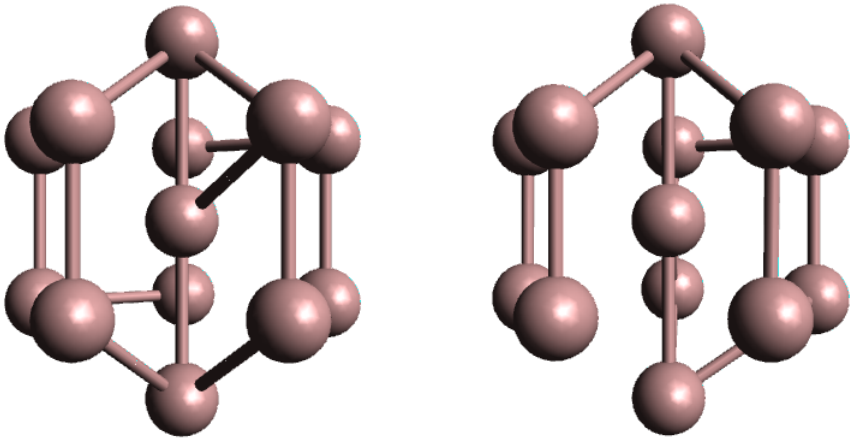
3.3 – Gallium Validation

Care must be taken to ensure that the generated parameters indeed have predictive power for further systems. The goal of SCED-LCAO is to quickly calculate previously unknown atomic configurations.

Therefore, two large homogeneous gallium clusters were chosen to validate the gallium SCED-LCAO parameters. The first is a nearly symmetric 13-atom ball. The D_{5h} symmetry is slightly broken due to the Jahn-Teller effect. H.A. Jahn and E. Teller used group theory to analyze the perturbation calculation on the position of the nucleus to prove that "All non-linear nuclear configurations are therefore unstable for an orbitally degenerate electronic state [47]." The addition of an electron to the system results in the perfectly symmetrical Ga_{13}^{-} D_{5h} ball [48]. However, we modeled the neutral off-symmetric structure.

An initial structure was generated by guessing the necessary bond lengths and allowed to relax using the VASP™ program with the GGA gallium potential. The structure that was produced nearly possesses the D_{5h} symmetry as predicted by Drebov, et.al. [48]. This geometry was then introduced into the SCED-LCAO molecular dynamics routine using the gallium parameters previously produced. After 50,000 time steps, the structure has completely relaxed into a structure that matches the geometry of the VASP™ calculations, as seen in figure 3.8. Note that the lines joining the atoms are generated by the visualization software and do not indicate bonding or lack of bonding, but rather are based on pair distance alone.

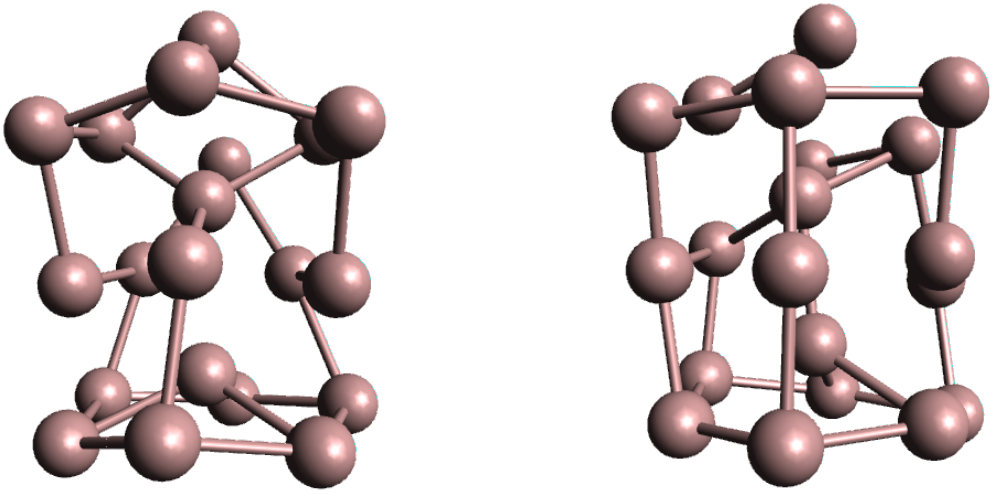
The second validation cluster is a 20-atom with no discernible symmetry. An initial structure was again generated and allowed to relax using the VASP™ program with the GGA gallium potential. However, the symmetry of the initial guess was retained in this calculation, so the cluster was first relaxed using the SCED-LCAO molecular dynamics routine using the gallium parameters. Again we used 50,000 time steps and found that the structure was completely relaxed into a structure that matches the structure pictured in Drebov, et.al. [48]. This non-symmetrical structure was further relaxed in the VASP™ program and pro-



VASP™ output SCED-LCAO output

Figure 3.8: Ga_{13} Validation

duced a structure very similar to the SCED-LCAO prediction, as seen in figure 3.9.



VASP™ output SCED-LCAO output

Figure 3.9: Ga_{20} Validation

Both of these validation clusters were further analyzed by plotting the pair-distribution function and the angle-distribution function for both the SCED and VASP™ results. The plots are given in figure 3.12.

Given the good agreement in both validation clusters, we are confident that the gallium parameters have the predictive power to correctly model other homogeneous gallium structures. Therefore, these gallium parameters also form a solid base upon which to build Nitrogen parameters, fitting them against gallium nitride clusters.

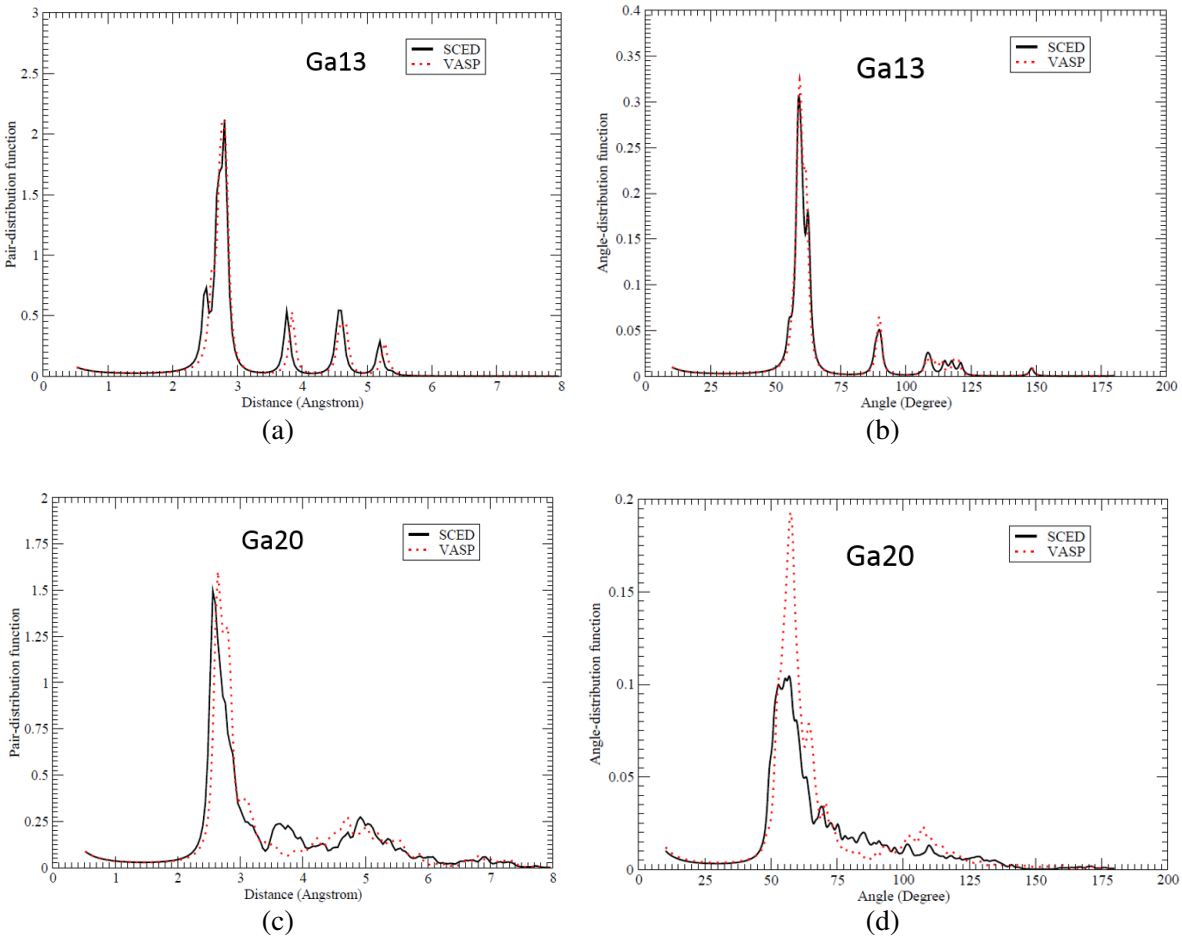


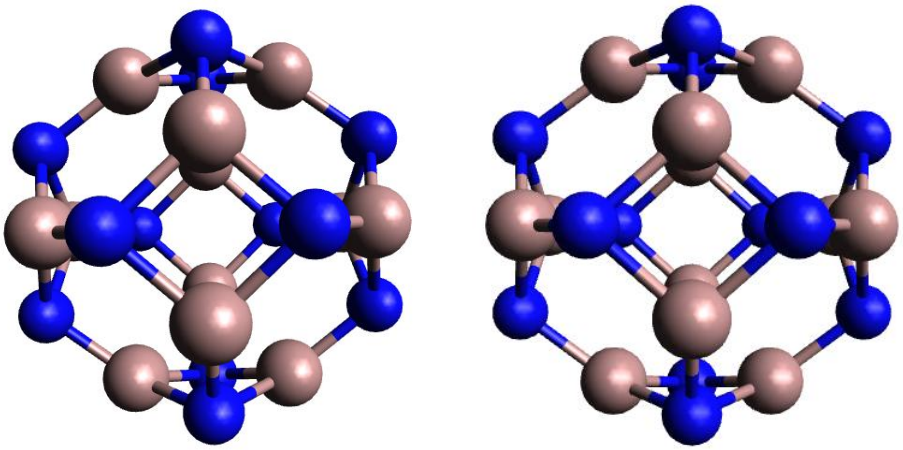
Figure 3.10: Pair-Distribution functions for Ga_{13} (a) and Ga_{20} (c) and angle-distribution functions for Ga_{13} (b) and Ga_{20} (d).

3.4 – Nitrogen Validation

While matching the properties in the database for GaN structures is extremely important for forming a useful set of SCED-LCAO parameters, SCED-LCAO should be able to accurately predict results for larger systems. To demonstrate that our results provide this pre-

dictive power, we will use the SCED-LCAO formalism and the parameters we calculated to predict properties for a larger symmetric cluster and a large periodic system.

For the larger cluster, we chose a highly symmetric 24-atom cluster that was first modeled in [34]. Again, an initial structure was created using approximate GaN bond lengths and then relaxed in both VASP™ and SCED-LCAO. The resulting structures are given in figure 3.11. The pair-distribution and angle-distribution functions are given in **??**.



VASP™ output SCED-LCAO output

Figure 3.11: $Ga_{12}N_{12}$ Validation

There is very good agreement between the two results, although the SCED results have a slightly larger bond lengths.

We chose to test our SCED-LCAO parameters with the most common form of GaN bulk crystal structure. While the cubic zinc-blende is also discussed in the literature,the hexagonal wurtzite most commonly occurs in nature and is the form used in semiconductor manufacturing. Both structures form planar hexagonal layers of gallium with a closely bonded hexagonal layer of nitrogen at the triangle centers and offset by a small amount in the direction of the z [001] axis. This pair of layers is repeated with a larger separation between layer pairs. The only difference between zinc-blende and wurzite is in the layer packing. The zinc-blende structure has layers organized as AaBbCc where the capital letters represent gallium layers and the lowercase represent the nitrogen layers. In contrast, the wurzite

structure has layers stacked as AaBbAa, where the every other layer is identical in the *xy* plane. An illustration is given in figure 3.13.

We will predict the geometrical configuration for minimum energy. For the wurtzite structure, three parameters are needed to completely describe the crystal lattice. We will use the ABA layer distance c between the two aligned gallium layers, the Aa layer separation distance d between a gallium layer and its nearest nitrogen layer, and the planar hexagon side length a. These have known values of 5.185 Å, 0.643 Å, and 3.189 Å respectively [Vurgaftman2003, Bernardini1997].

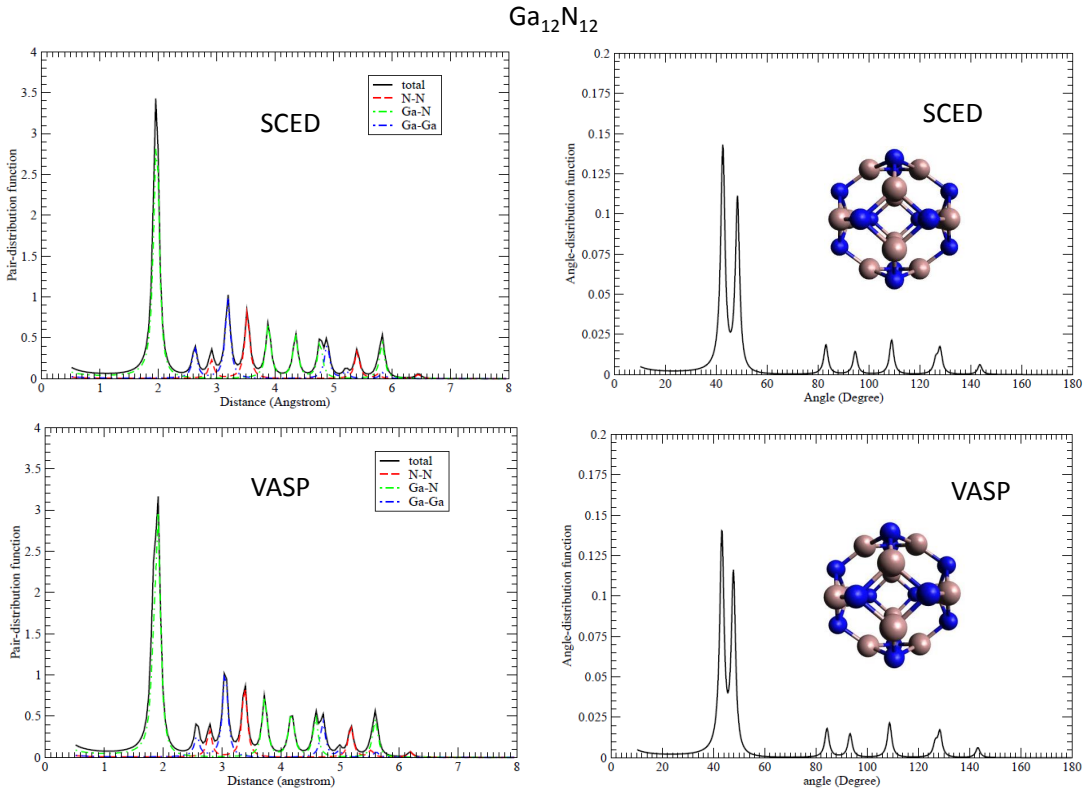


Figure 3.12: Pair-Distribution and Angle-Distribution functions for $Ga_{12}N_{12}$.

Our SCED-LCAO predictions were made with the molecular dynamics program developed in [3]. We began with the known geometries for a block of GaN bulk as shown in figure 3.14. We applied periodic boundary conditions and used the parameters for gallium and nitrogen that we found earlier. The structure was allowed to completely relax to its

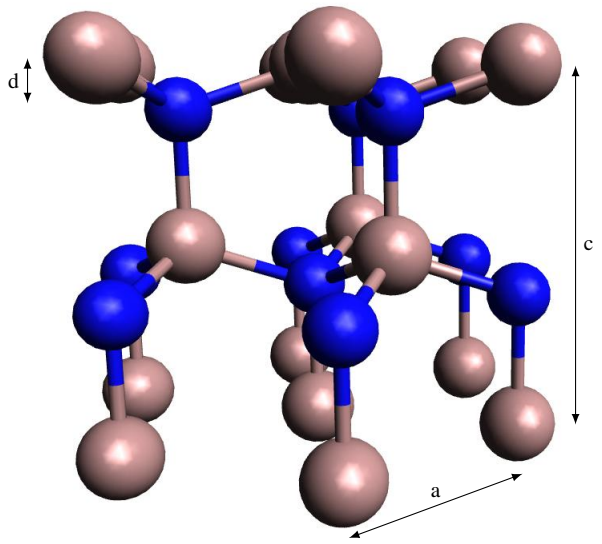


Figure 3.13: Hexagonal wurtzite structure for gallium nitride.

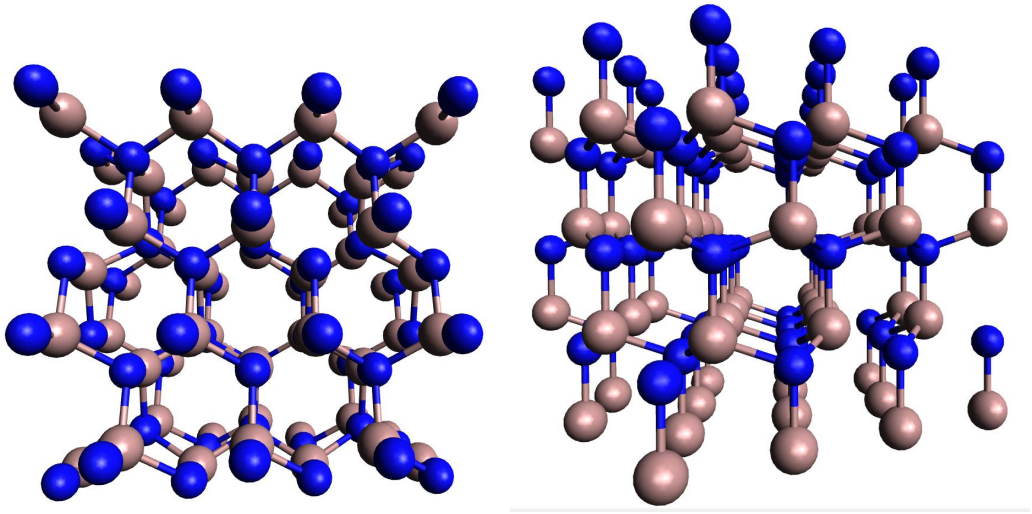


Figure 3.14: Block of gallium nitride.

stable structure using the powerquenching technique. As before, a breathing mode was accomplished by scaling all three unit cell vectors and finding the energy for each scaling. The same procedure was performed using the VASP™ program for comparison. These results are plotted in figure 3.15.

While the minimum energy for the SCED-LCAO results occurs at a scaling factor of 1.02, the 2% error is considered well within the acceptable range.

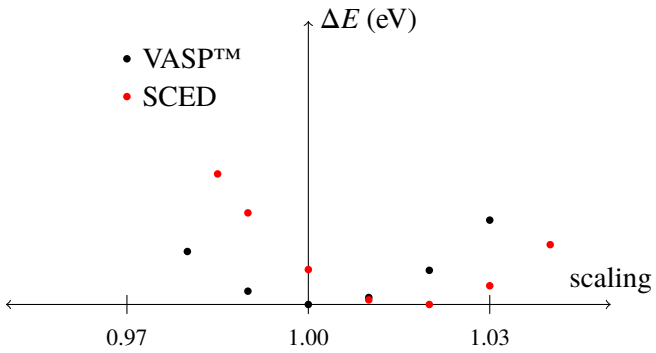


Figure 3.15: GaN bulk energy versus lattice scaling.

Thus, there is excellent agreement for both the $Ga_{12}N_{12}$ cluster and the bulk properties. This demonstrates that SCED-LCAO can accurately predict large GaN structures with the parameters given. We can now be confident that SCED-LCAO with the parameters produced here can accurately predict structures involving Gallium and Nitrogen.

CHAPTER 4

PREDICTION OF GALLIUM NITRIDE STRUCTURES

Having validated the Gallium and Nitrogen SCED-LCAO parameters, we will now turn our attention to the prediction of larger structures. The true power of SCED-LCAO lies in its ability to accurately predict structures and properties where ab-initio methods are too costly computationally. SCED-LCAO is able to give accurate predictions in situations where the multi-center interactions are important.

We continue our study of Gallium Nitride cage structures by considering two additional clusters – $Ga_{16}N_{16}$ and $Ga_{24}N_{24}$. These were studied in depth by Brena and Ojamae [46] using Gaussian™ with a light basis set. We extend our study beyond cluster geometries to also include electronic structure. These clusters were allowed to relax in the SCED-LCAO molecular dynamics simulation to find the lowest energy state. The Pair-Distribution functions and Angle-Distribution functions are given for $Ga_{16}N_{16}$ and $Ga_{24}N_{24}$ in figure 4.1 and figure 4.2, respectively. As expected, the results from SCED are a very close match with those of VASP™ for $Ga_{16}N_{16}$. The $Ga_{24}N_{24}$ was too large to model in VASP™ with our current machines. This confirms that the current parameters in SCED-LCAO provide a firm foundation for additional studies.

We also explored the electronic structure of $Ga_{12}N_{12}$, $Ga_{16}N_{16}$, and $Ga_{24}N_{24}$. The electronic density of states (DoS) was calculated and plotted for each, along with the DoS for GaN bulk. These results are summarized in figure 4.3.

One quatity of interest is the band gap for these structures. This is the gap in the energy of the highest occupied molecular orbital (HOMO) and the lowest unoccupied molecular orbital (LUMO). For semi-conductors such as GaN, we expect a small, but distinct gap. This gap allows for the electrical behavior that makes transistors possible. As seen in figure 4.3, in

each case we found such a gap. These gaps also matched very closely with the results from Brena and Ojamae [46] using the 6-31G(d,p) basis sets in Gaussian™. These results are plotted in figure 4.4. The gap for $Ga_{24}N_{24}$ is significantly different since the SCED cluster relaxed into a non-symmetric structure, while the Gaussian™ result maintained symmetry.

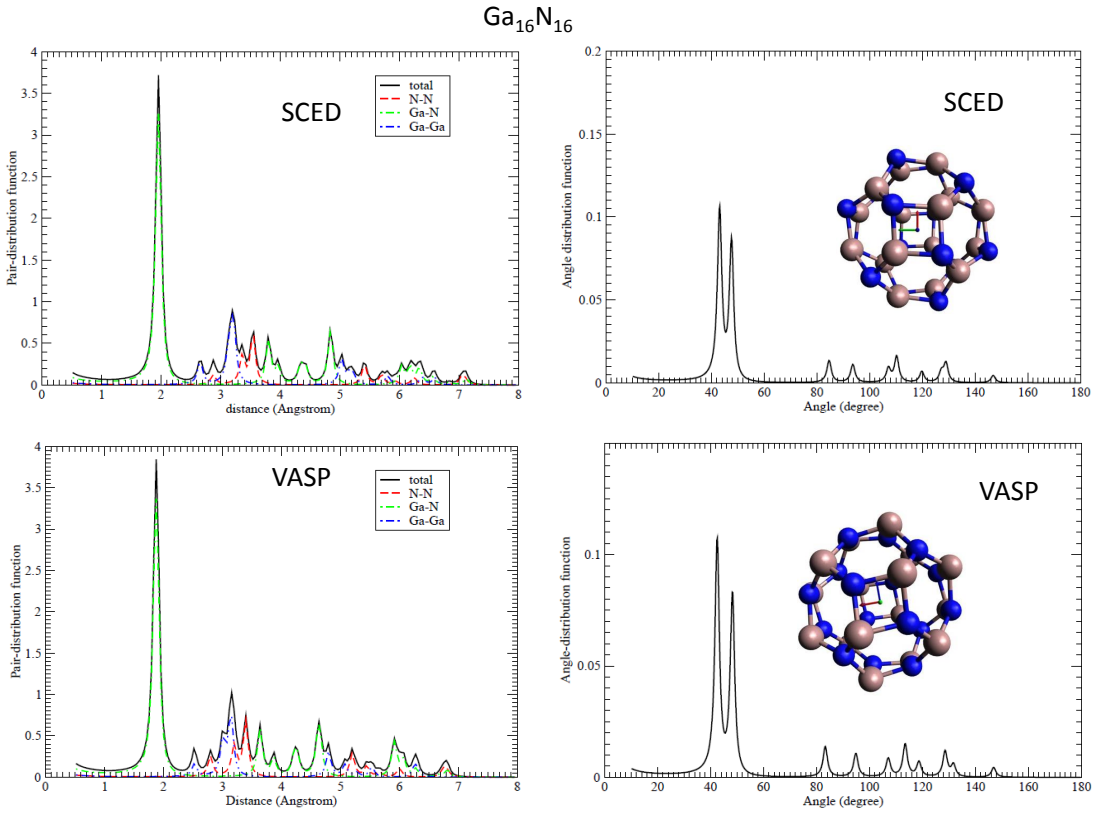


Figure 4.1: Pair-Distribution and Angle-Distribution functions for $Ga_{16}N_{16}$.

These results are encouraging and are but a glimpse of the power of SCED-LCAO.

$Ga_{24}N_{24}$

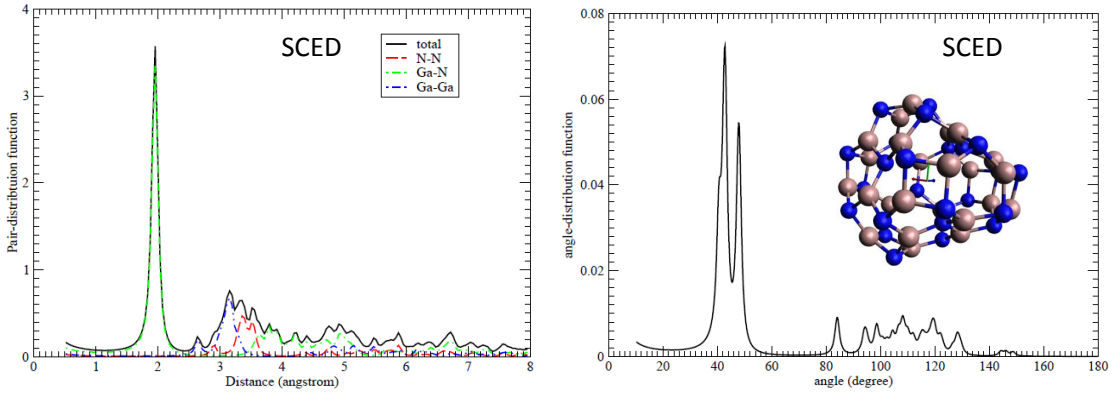


Figure 4.2: Pair-Distribution and Angle-Distribution functions for $Ga_{24}N_{24}$.

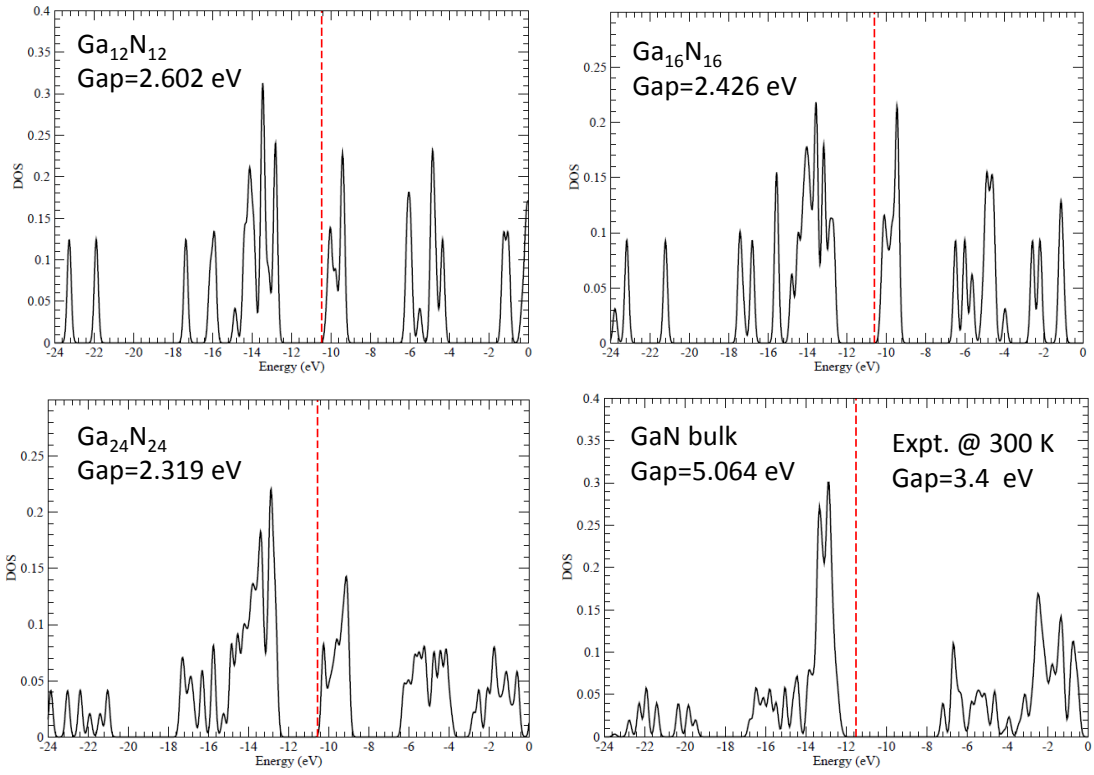


Figure 4.3: Density of States results for SCED results.

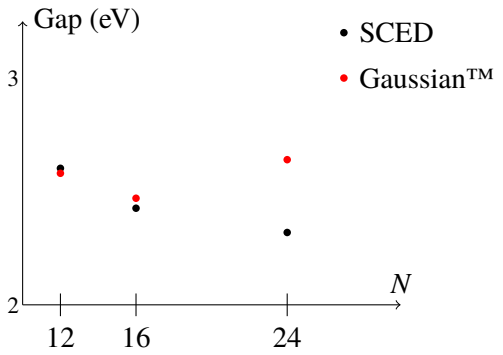


Figure 4.4: GaN band gap versus number of GaN pairs N.

CHAPTER 5

CONCLUSION

5.1 – Comparison of Methods

The self-consistency problem provides an unusual challenge for computational solution. Within the framework of SCED-LCAO, the SC problem is often highly non-linear in multiple dimensions, especially surrounding the root – a veritable nightmare scenario for root-finding algorithms. The Generalized Bisection method provides solutions in many cases where derivative-based methods fail due to charge sloshing. This is tremendously important for the task of generating parameters for SCED-LCAO. During this fitting procedure, failure to converge to a root can be catastrophic. The only way to deal with a self-consistency failure in the fitting procedure is to abandon the entire set of parameters. While the optimal parameter set usually converges to a root with derivative methods, the process of arriving at this optimal set often requires the calculation of many self-consistent systems with the Generalized Bisection method.

The Generalized Bisection method will often provide a solution when the Broyden method fails, but it is comparatively very slow in its convergence to the root. Our goal is to decrease the computational time required for generating parameters, so we need to minimize number of steps in this process. Given a series of timing runs, it was determined that the optimal way to find roots was to employ the gradient methods whenever possible and to switch to the Generalized Bisection method when the derivative method fails. Since even one root-finding failure can result in discarding hundreds of thousands of successful solutions, the added expense of the Generalized Bisection method is more than justified in this case.

5.2 – Gallium Nitride

Gallium and nitrogen form strong bonds with significant electron transfer in SCED-LCAO. For this heterogeneous system, an improvement to the self-consistency algorithm was necessary to find parameters to describe this bonding. In addition, our work represents the first attempt to fit parameters for an entire elemental species using only heterogeneous properties. Since heterogeneous nitrogen forms only dimers (N_2) under most natural conditions, it was not profitable to attempt to create a homogeneous nitrogen database of clusters and bulk. Therefore, we used only heterogeneous clusters to fill out the property database (along with the nitrogen dimer), namely those clusters consisting of gallium and nitrogen, carbon and nitrogen, and boron and nitrogen. Given this situation, the burden on the self-consistency algorithm was especially high, increasing the likelihood of failures. The inclusion of the Generalized Bisection method as a "safety net" for when the Broyden method failed was essential to the successful generation of SCED-LCAO parameters for nitrogen.

5.3 – Direction for Future Work

While the Generalized Bisection method allows for successful root finding in most SC problems, it does not always converge. The particularly devilish nature of the SCED-LCAO self-consistency problem will occasionally lead to failure. Given the importance of finding the solution for every case in the fitting procedure, more work should be done to eliminate this small percentage of failures. One possible route of investigation would be to improve the Generalized Bisection method by optimizing the size of the relaxation factor after a corner sign failure. Perhaps even a randomly generated relaxation factor would prevent Generalized Bisection failures. Hybrid methods combining Newton's method and the bisection method are easy to envision in one dimension. Perhaps these can be generalized to higher dimensions using techniques from Generalized Bisection.

Gallium nitride has been a hot topic for research in recent years. The crystal properties of gallium nitride have been shown to be superior to other semi-conductors in heat and

radiation sensitivity, voltage handling, and in the production of violet laser diodes. Recent research has been shifting to the nano-scale. Gallium nitride has been shown to form nanowires and nanotubes, which have the potential to provide next-generation semiconductors and optical devices.

The inclusion of the Generalized Bisection method opens the door for a wide array of other studies with SCED-LCAO. Parameters can be generated for many additional elements. As demonstrated in this work, elemental species need not form their own homogeneous crystal structure or small clusters to provide a sufficient database of properties. Since boron has already been studied, boron nitride can be quickly investigated. Perhaps it too will form nanowires with interesting properties.

REFERENCES

[1] S. Konyukhov, I. Kupchenko, A. Moskovsky, A. Nemukhin, A. Akimov, and A. Kolomeisky, "Rigid-body molecular dynamics of the fullerene-based nanocars on the metallic surfaces," *J. Chem. Theor. Comp.*, vol. 6, pp. 2581–2590, 2010. DOI: 10.1021/ct100101y. URL: http://python.rice.edu/~kolomeisky/nanocar.htm.

[2] S. M. Lee, Y. H. Lee, Y. G. Hwang, J. Elsner, D. Porezag, and T. Frauenheim, "Stability and electronic structure of GaN nanotubes from density-functional calculations," *Phys. Rev. B*, vol. 60, pp. 7788–7791, 11 1999-09. DOI: 10.1103/PhysRevB.60.7788. URL: http://link.aps.org/doi/10.1103/PhysRevB.60.7788.

[3] C. R. Leahy, M. Yu, C. S. Jayanthi, and S. Y. Wu, "Coherent treatment of the self-consistency and the environment-dependency in a semi-empirical hamiltonian: applications to bulk silicon, silicon surfaces, and silicon clusters," *Phys. Rev. B*, vol. 74, no. 15, 155408:1–13, 2006. DOI: 10.1103/PhysRevB.74.155408.

[4] M. Yu, I. Chaudhuri, C. Leahy, S. Y. Wu, and C. S. Jayanthi, "Energetics, relative stabilities, and size-dependent properties of nanosized carbon clusters of different families: fullerenes, bucky-diamond, icosahedral, and bulk-truncated structures," *The Journal of Chemical Physics*, vol. 130, no. 18, 184708, p. 184 708, 2009. DOI: 10.1063/1.3124827. URL: http://link.aip.org/link/?JCP/130/184708/1.

[5] W. Q. Tian, M. Yu, C. Leahy, C. S. Jayanthi, and S.-Y. Wu, "The self-consistent and environment-dependent hamiltonian and its application to carbon nanoparticles," *Journal of Computational and Theoretical Nanoscience*, vol. 6, no. 2, pp. 390–396, 2009. DOI: doi:10.1166/jctn.2009.1048. URL: http://www.ingentaconnect.com/content/asp/jctn/2009/00000006/00000002/art00016.

[6] M. Yu, S. Wu, and C. Jayanthi, "A self-consistent and environment-dependent hamiltonian for large-scale simulations of complex nanostructures," *Physica E: Low-dimensional Systems and Nanostructures*, vol. 42, no. 1, pp. 1–16, 2009, ISSN: 1386-9477. DOI: 10.1016/j.physe.2009.08.024. URL: http://www.sciencedirect.com/science/article/pii/S1386947709003208.

[7] M. Yu, C. S. Jayanthi, and S. Y. Wu, "Geometric and electronic structures of graphitic-like and tubular silicon carbides: *Ab-initio* studies," *Phys. Rev. B*, vol. 82, p. 075 407, 7 2010-08. DOI: 10.1103/PhysRevB.82.075407. URL: http://link.aps.org/doi/10.1103/PhysRevB.82.075407.

[8] M. Yu, C. S. Jayanthi, and S. Y. Wu, "Theoretical predictions of a bucky-diamond sic cluster," *Nanotechnology*, vol. 23, no. 23, p. 235 705, 2012. URL: http://stacks.iop.org/0957-4484/23/i=23/a=235705.

[9] M. Yu, C. Jayanthi, and S. Wu, "Size-, shape-, and orientation-dependent properties of sic nanowires of selected bulk polytypes," *Journal of Materials Research*, vol. 28, pp. 57–67, 01 2013-1, ISSN: 2044-5326. DOI: 10.1557/jmr.2012.237. URL: http://journals.cambridge.org/article_S0884291412002373.

[10] P. Tandy, M. Yu, C. Leahy, C. S. Jayanthi, and S. Y. Wu, "Next generation of the self-consistent and environment-dependent hamiltonian: applications to various boron allotropes from zero- to three-dimensional structures," *The Journal of Chemical Physics*, vol. 142, no. 12, 2015. DOI: http://dx.doi.org/10.1063/1.4916069. URL: http://scitation.aip.org/content/aip/journal/jcp/142/12/10.1063/1.4916069.

[11] D. D. Johnson, F. J. Pinski, and G. M. Stocks, "Fast method for calculating the self-consistent electronic structure of random alloys," *Physical Review B*, vol. 30, no. 10, pp. 5508–5515, 1984.

[12] D. D. Johnson, "Modified broyden's method for accelerating convergence in self-consistent calculations," *Physical Review B: Condensed Matter*, vol. 38, no. 18, pp. 807–813, 1988.

[13] J. E. Dennis Jr and J. J. Moree, "Quasi-newton methods, motivation and theory," *SIAM Review*, vol. 19, no. 1, pp. 46–89, 1977.

[14] R. H. Byrd, J. Nocedal, and R. B. Schnabel, "Representation of quasi-newton matrices and their use in limited memory methods," pp. 1–30, 1996.

[15] M. Kawata, C. M. Cortis, and R. A. Friesner, "Efficient recursive implementation of the modified broyden method and the direct inversion in the iterative subspace method: acceleration of self-consistent calculations," *Journal of Chemical Physics*, vol. 108, no. 11, pp. 4426–4438, 1998.

[16] G. P. Srivastava, "Broyden's method for self -consistent field convergence acceleration," *J. Phys. A: Math. Gen*, vol. 17, pp. L317–L321, 1984.

[17] B. Kearfott, "An efficient degree-computation method for a generalized method of bisection," *Numer. Math*, vol. 32, pp. 109–127, 1979.

[18] M. N. Vrahatis and K. I. Iordanidis, "A rapid generalized method of bisection for solving systems of nonlinear equations," *Numer. Math.*, vol. 49, no. 2-3, pp. 123–138, 1986, ISSN: 0029-599X. DOI: 10.1007/BF01389620. URL: http://dx.doi.org/10.1007/BF01389620.

[19] M. N. Vrahatis, I. Z. Emiris, and B. Mourrain, "Sign methods for counting and computing real roots of algebraic systems," 1999.

[20] G. R. Wood, "The bisection method in higher dimensions," *Math. Programming*, vol. 55, no. 3, Ser. A, pp. 319–337, 1992, ISSN: 0025-5610. DOI: 10.1007/BF01581205. URL: http://dx.doi.org/10.1007/BF01581205.

[21] M. N. Vrahatis, G. Servizi, and T. Boutis, "A procedure to compute the fixed points and visualize the orbits ofa 2d map," 1993.

[22] M. N. Vrahatis, "An efficient method for locating and computing periodic orbits of nonlinear mappings," *Journal of Computational Physics*, vol. 119, pp. 105–119, 1995.

[23] M. N. Vrahatis, A. E. Perdiou, V. S. Kalantonis, E. A. Perdios, K. Papadakis, R. Prosmiti, and S. C. Farantos, "Application of the characteristic bisection method for locating and computing periodic orbits in molecular systems," 2001.

[24] A. D. Becke, "A new mixing of hartree–fock and local density-functional theories," *The Journal of Chemical Physics*, vol. 98, no. 2, pp. 1372–1377, 1993. DOI: 10.1063/1.464304. URL: http://link.aip.org/link/?JCP/98/1372/1.

[25] D. E. Woon and J. Thom H. Dunning, "Gaussian basis sets for use in correlated molecular calculations. iii. the atoms aluminum through argon," *The Journal of Chemical Physics*, vol. 98, no. 2, pp. 1358–1371, 1993. DOI: 10.1063/1.464303. URL: http://link.aip.org/link/?JCP/98/1358/1.

[26] E. R. Davidson, "Comment on "comment on dunning's correlation-consistent basis sets"," *Chemical Physics Letters*, vol. 260, no. 3–4, pp. 514–518, 1996, ISSN: 0009-2614. DOI: 10.1016/0009-2614(96)00917-7. URL: http://www.sciencedirect.com/science/article/pii/0009261496009177.

[27] C. Leahy, "Self-consistent and environment-dependent hamiltonian for quantum-mechanics materials simulations," *PhD dissertation*, 2007.

[28] B. Song and P. Cao, "Evolution of the geometrical and electronic structures of Ga_n (n=2-26) clusters: a density-functional theory study," *The Journal of chemical physics*, vol. 123, no. 144312, 2005. DOI: 10.1063/1.2047527.

[29] X. G. Gong and E. Tosatti, "Structure of small gallium clusters," *Physics letters A*, vol. 166, no. 5,6, pp. 369–372, 1992, ISSN: 0375-9601. DOI: http://dx.doi.org/10.1016/0375-9601(92)90725-2. URL: http://www.sciencedirect.com/science/article/pii/0375960192907252.

[30] Y. Zhao, W. Xu, Q. Li, Y. Xie, and H. F. Schaefer III, "Gallium clusters Ga_n (n=1-6): structures, thermochemistry, and electron affinities," *J. Phys. Chem. A*, vol. 108, no. 36, pp. 7448–7459, 2004. DOI: 10.1021/jp0402784.

[31] J. Moc, "Can gallium dimer react effectively with three H_2 molecules to form digallane?" *Chemical Physics*, vol. 313, pp. 93–100, 2005. DOI: 10.1016/j.chemphys.2004.12.018.

[32] J. B. Mann, "Atomic structure calculations i. hartree-fock energy results for the elements report la-3690," 1967.

[33] W. Harrison, *Elementary Electronic Structure*. World Scientific Publishing Company Incorporated, 2004, ISBN: 9789812387080. URL: http://books.google.com/books?id=yZrkcSwlr2YC.

[34] J. Zhao, B. Wang, X. Zhou, Z. Chen, and W. Lu, "Structure and electronic properties of medium-sized Ga_nN_n clusters (n=4–12)," *Chemical Physics Letter*, vol. 422, pp. 170–173, 2006.

[35] P. S. Yadav, R. K. Yadav, and A. B. K., "Structural, electronic and vibrational properties of small Ga_xN_y (x+y=2–5) nanoclusters: a b3lyp-dft study," *Journal of Physics:Condensed Matter*, vol. 19, no. 076209, pp. 1–28, 2007. DOI: 10.1088/0953-8984/19/7/076209.

[36] A. C. Pineda and S. P. Karan, "(hyper)polarizabilities of isolated GaN nanoclusters," *Chemical Physics Letters*, vol. 429, pp. 169–173, 2006. DOI: 10.1016/j.cplett.2006.07.067.

[37] Z. Hao-Ping and H. Jing-An, "Ab initio study of the electronic properties of the planar Ga_5N_5 cluster," *Chinese Physics*, vol. 14, no. 3, pp. 529–532, 2005.

[38] A. K. Kandalam, M. A. Blanco, and R. Pandey, "Theoretical study of Al_nN_n, Ga_nN_n, and In_nN_n (n=4, 5, 6) clusters," *J. Phys. Chem. B*, vol. 6, no. 8, pp. 1945–1953, 2002. DOI: 10.1021/jp0140062.

[39] A. Costales, M. A. Blanco, A. M. Pendas, A. K. Kandalam, and R. Pandey, "Chemical bonding in group III nitrides," *J. Am. Chem. Soc.*, vol. 124, no. 15, pp. 4116–4123, 2002. DOI: 10.1021/ja017380o.

[40] M. Zhou and L. Andrews, "Reactions of laser-ablated Ga, In, and Tl atoms with nitrogen atoms and molecules. infrared spectra and density functional calculations of GaN, NGaN, NInN, and the M_3N and MN_3 molecules," *J. Phys. Chem. A*, vol. 104, no. 8, pp. 1648–1655, 2000. DOI: 10.1021/jp993429p.

[41] E. P. F. Lee and J. M. Dyke, "An ab initio study of the low-lying doublet states of linear and t-shaped Ga-N_2," *J. Phys. Chem. A*, vol. 104, no. 50, pp. 11 810–11 815, 2000. DOI: 10.1021/jp002869+.

[42] J. J. Belbruno, "The structure of small galium nitride clusters," *Heteroatom Chemistry*, vol. 11, no. 4, pp. 281–286, 2000.

[43] B. Song, P. Cao, and B. Li, "Theoretical study of the Ga_6N_6 cluster," *Physics Letters A*, vol. 315, pp. 308–312, 2003. DOI: 10.1016/S0375-9601(03)01035-1.

[44] A. K. Kandalam, M. A. Blanco, and R. Pandey, "Theoretical study of structural and vibrational properties of Al_3N_3, Ga_3N_3, and In_3N_3," *J. Phys. Chem. B*, vol. 105, no. 26, pp. 6080–6084, 2001. DOI: 10.1021/jp004404p.

[45] A. Costales and R. Pandey, "Density functional calculations of small anionic clusters of group III nitrides," *J. Phys. Chem. A*, vol. 107, no. 1, pp. 191–197, 2003. DOI: 10.1021/jp022202i.

[46] B. Brena and L. Ojamae, "Effects and quantum confinement in nanosized GaN clusters: theoretical predictions," *J. Phys. Chem. C*, vol. 112, no. 35, pp. 13 516–13 523, 2008. DOI: 10.1021/jp8048179.

[47] H. A. Jahn and E. Teller, "Stability of polyatomic molecules in degenerate electronic states. i. orbital degeneracy," *Proceedings of the Royal Society A*, vol. 161, no. 905, pp. 220–235, 1937. DOI: 10.1098/rspa.1937.0142.

[48] N. Drebov, F. Weigend, and R. Ahlrichs, "Structures and properties of neutral gallium clusters: a theoretical investigation," *The Journal of Chemical Physics*, vol. 135, no. 4, 044314, 2011. DOI: http://dx.doi.org/10.1063/1.3615501. URL: http://scitation.aip.org/content/aip/journal/jcp/135/4/10.1063/1.3615501.

APPENDIX A

GALLIUM CLUSTER DATABASE

The following is a list of clusters used to create the Gallium database. The geometries and cluster energies were produced using Guassian™.

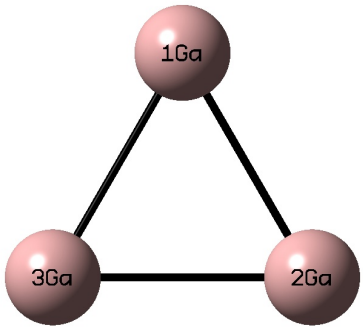


Figure A.1: Ga_3 D_3h

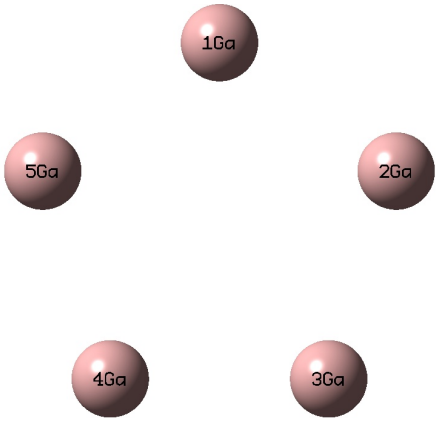


Figure A.2: Ga_5 D_5h

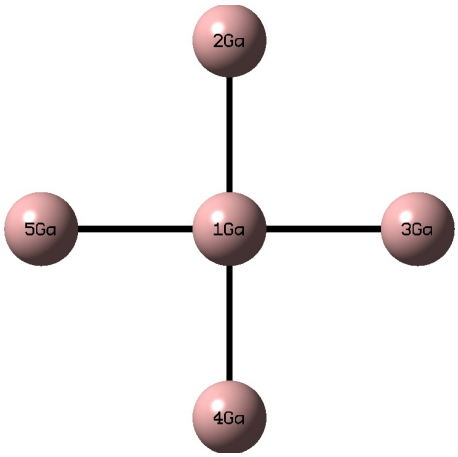


Figure A.3: Ga_5 D_4h

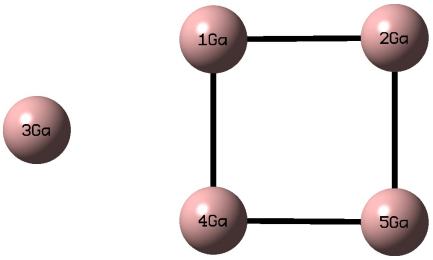


Figure A.4: Ga_5 C_2v

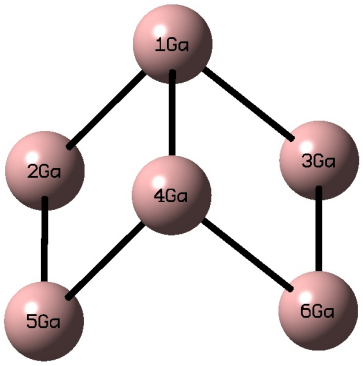


Figure A.5: Ga_6 C_2v a

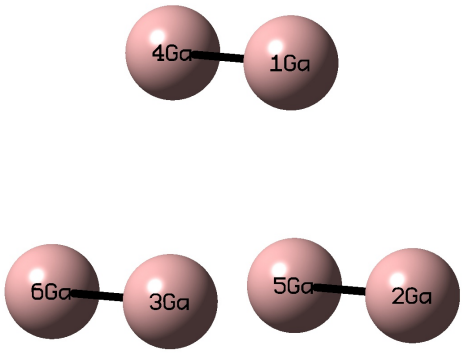


Figure A.6: Ga_6 D_3h

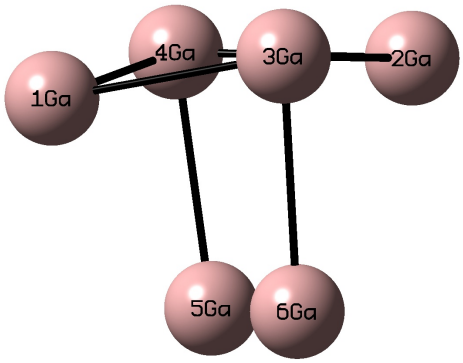


Figure A.7: Ga_6 C_2v b

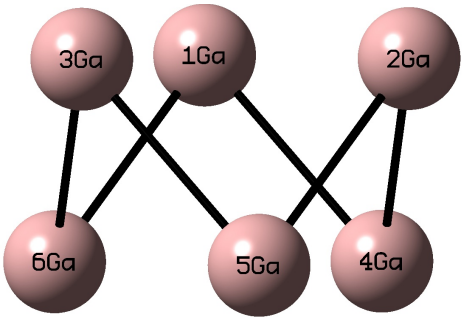


Figure A.8: Ga_6 D_3d

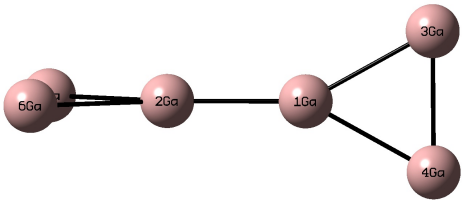


Figure A.9: Ga_5 D_2d

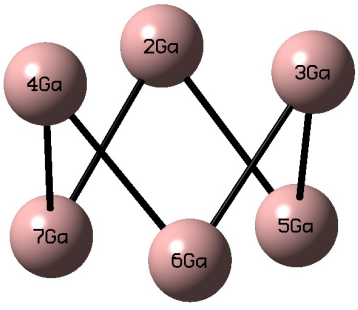


Figure A.10: Ga_7 C_3v

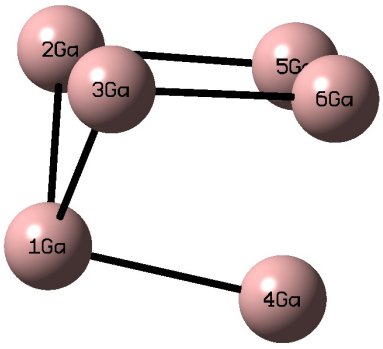


Figure A.11: Ga_7 Cs

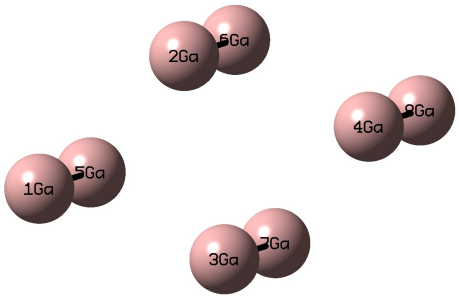


Figure A.12: Ga_8 D_2h

APPENDIX B

NITROGEN CLUSTER DATABASE

The Nitrogen SCED-LCAO parameters were found by fitting against the following heterogeneous clusters, pairing Nitrogen with Gallium.

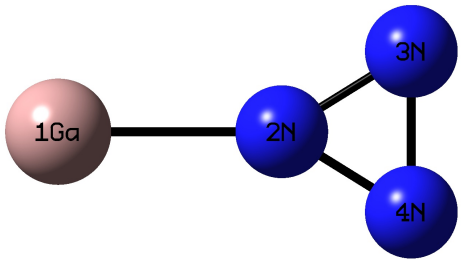


Figure B.1: Ga_1N_3 C_2v b

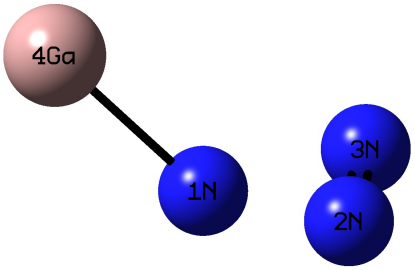


Figure B.2: Ga_1N_3 Pyramidal

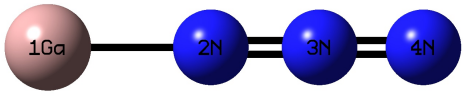


Figure B.3: Ga_1N_3 $C_\infty h$

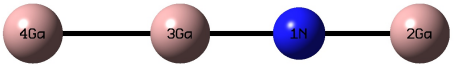


Figure B.4: Ga_3N_1 $C_\infty h$

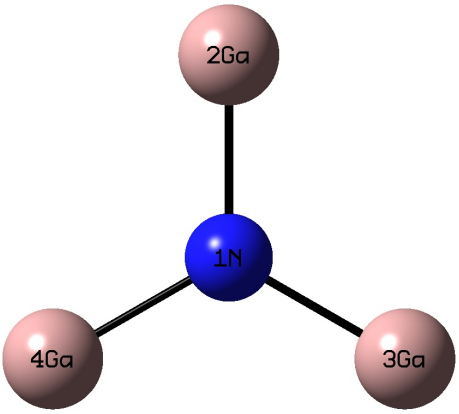


Figure B.5: Ga_3N_1 D_3h

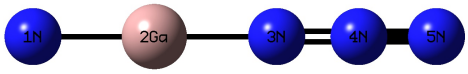


Figure B.6: Ga_1N_4 $C_\infty h$

Figure B.7: Ga_2N_3 $C_\infty h$

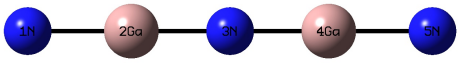


Figure B.8: Ga_2N_3 $D_\infty h$

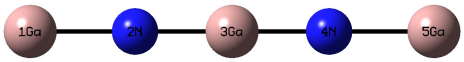


Figure B.9: Ga_3N_2 $D_\infty h$

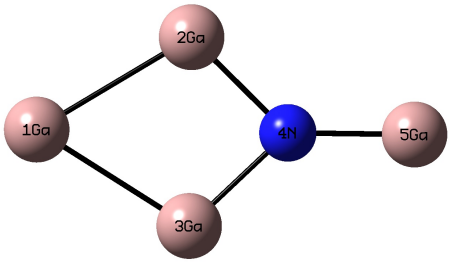


Figure B.10: Ga_4N_1 $C_2 v$

Figure B.11: Ga_4N_1 $C_\infty h$

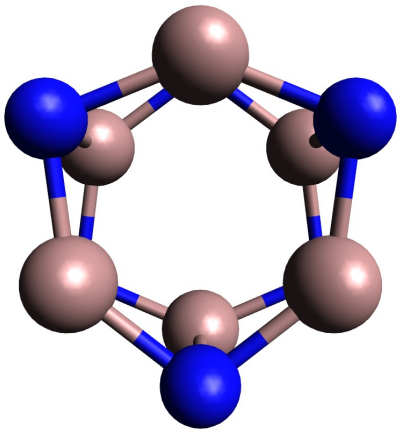


Figure B.12: Ga_6N_6 $D_3 d$

APPENDIX C

GENERALIZED BISECTION COMPUTER CODE

The Self-Consistency routines were written into the pre-existing structure of the fitting code. The fitting code executes the many fitting loops using a reverse communication technique. Flags are used in the code to determine which parts need to be run. This allows the program to determine optimization controls without knowing the function that is being optimized.

Total Cluster

```
!<>================================
!<>
!<>  Fortran source code subroutines.
!<>
!<>  Filename: total_cluster.F
!<>
!<>  Description: <fitting-doc-tag-short> <need-file-description>
!<>
!<>  Documentation: <fitting-doc-tag-short> <need-doc-description>
!<>
!<>  Arranged by: Lyle Smith   rev. 1   July  1, 2010
!<>                            rev. 2   Jan. 19, 2011
!<>                            rev. 2.1 Jan. 31, 2011
!<>                         rev. 2.2 Mar. 21, 2011 - improved error output.
!<>                       rev. 2.3 May  24, 2011 - fixed a bug in error output.
!<>                         rev. 2.3.1 July 12, 2011 - include cheats for all.
!<>
!<>  There are 2 self-consistency schemes included in this version:
!<>  1. the Broyden Method
!<>  2. Generalized Bisection (limited to 8 or less degrees of freedom)
!<>
!<>  It was determined that greatest efficiency was achieved by using the Broyden
!<>  method.  This is true for large systems (5+ DoFs) even when a large amount
!<>  of charge sloshing occured.  Therefore, the Broyden method is always used
!<>  initially.  If the Broyden method exceeds the maximum number of steps, then
!<>  the routine will switch to the Generalized Bisection method.  Preliminary
!<>  tests show this to find the root in excess of 99.999% of systems.
!<>
!<>================================
!<>
!<>
!<>
!<>
!<>================================  total_cluster
!<>
!<>  <fitting-doc-tag-short> <need-sub-description>
!<>
!<>  <fitting-doc-tag-short> <unsupported-sub-declare>
subroutine total_cluster( E_tot, size_R, charge_config, spin_config,&
  elem_R, sites_X_R,Free_R, size_MP, curr_EP, err_tot )
!<>
!<>  <fitting-doc-tag-begin> <supported-use-block>
```

```
!<>
use alloc_module
!<>
use global_module
!<>
use clust_wrap_module
!<>
!<>  <fitting-doc-tag-end> <supported-use-block>
!<>
!<>================================  declarations
!<>
!<>================  arguments
!<>
!<>--------  input
!<>
!<>  <fitting-doc-tag-begin> <unsupported-arg-block>
!<>
integer( kind = 4 ) :: old_dec_EIG
integer( kind = 4 ) :: old_dec_HS
integer( kind = 4 ) :: size_R
character( len = length__elem ) :: elem_R( size_R )
real( kind = 8 ) :: sites_X_R( fix__X, size_R )
integer( kind = 4 ) :: spin_config
integer( kind = 4 ) :: charge_config
integer( kind = 4 ) :: Free_R( size_R )
integer( kind = 4 ) :: old_short_R_R( alloc__R, alloc__R )
integer( kind = 4 ) :: size_MP
real( kind = 8 ) :: curr_EP( size_MP )
!<>
!<>  <fitting-doc-tag-end> <unsupported-arg-block>
!<>
!<>--------  output
!<>
!<>  <fitting-doc-tag-begin> <unsupported-arg-block>
!<>
real( kind = 8 ) :: E_tot
integer( kind = 4 ) :: old_size_EIG
real( kind = 8 ) :: old_energy_EIG( alloc__clust_HS )
integer( kind = 4 ) :: old_size_HS
real( kind = 8 ) :: old_C_HS_qHS( alloc__clust_HS, alloc__clust_HS )
character( len = * ) :: err_tot
!<>
!<>  <fitting-doc-tag-end> <unsupported-arg-block>
!<>
!<>================  local variables
!<>
!<>  <fitting-doc-tag-begin> <unsupported-local-block>
!<>
integer( kind = 4 ), parameter :: MODEL_neutral_SC = 1
integer( kind = 4 ), parameter :: fix_EL = 4
real( kind = 8 ), parameter :: tune_elec_N_change = 1.0E-2_8
real( kind = 8 ), parameter :: tune_elec_N_toler = 1.0E-12_8
integer( kind = 4 ) :: size_RF, size_RF1, i_EIG
real( kind = 8 ) :: X_eval_RF( alloc__R )
real( kind = 8 ) :: G_eval_RF( alloc__R )
real( kind = 8 ) :: X_best_RF( alloc__R )
real( kind = 8 ) :: G_best_RF( alloc__R )
real( kind = 8 ) :: sites_prev_R( fix__X, alloc__R ); save sites_prev_R
integer( kind = 4 ) :: n_eval
integer( kind = 4 ) :: rev_eval
integer( kind = 4 ) :: save_iso ; save save_iso
real( kind = 8 ) :: X_init_RF( alloc__R ) ; save X_init_RF
real( kind = 8 ) :: X_change
real( kind = 8 ) :: X_scale_RF( alloc__R )
real( kind = 8 ) :: G_scale_RF( alloc__R )
```

```
real( kind = 8 ) :: X_toler
real( kind = 8 ) :: G_toler
real( kind = 8 ) :: G_accur
real( kind = 8 ) :: Ln_this_R( alloc__R )
real( kind = 8 ) :: Ln_next_R( alloc__R )
real( kind = 8 ) :: Lz_this_R( alloc__R )
integer( kind = 4 ) :: act_EIG
real( kind = 8 ) :: energy_E( alloc__clust_HS )
real( kind = 8 ) :: occupy_E( alloc__clust_HS )
real( kind = 8 ) :: E_band
real( kind = 8 ) :: E_cor
real( kind = 8 ) :: R_scale
real( kind = 8 ) :: E_scale
integer( kind = 4 ) :: flag_SC
integer( kind = 4 ) :: i_R, i_RF, n_RF,i_M
real( kind = 8 ) :: X_init, X_scale, G_scale
real( kind = 8 ) :: E_tract, F_set, G_norm
integer( kind = 4 ) :: i_R_RF( alloc__R )
integer( kind = 4 ) :: i_RF_R( alloc__R )
integer( kind = 4 ) :: flag_free, k_R
real( kind = 8 ) :: Ln_allbutlast, float_tot
integer( kind = 4 ) :: int_elec, flag_occ, degen_last
integer( kind = 4 ) :: get_n_val
real( kind = 8 ) :: f_int
real( kind = 8 ) :: f_invert
real( kind = 8 ) :: f_float_accur, max_SC_steps
integer( kind = 4 ) :: index_from_nat
integer( kind = 4 ) :: SC_method, switcher, n_eval_int
integer( kind = 4 ) :: n_eval_tot ; save n_eval_tot
integer( kind = 4 ) :: clust_curr ; save clust_curr
!<>
!<>  <fitting-doc-tag-end> <unsupported-local-block>
!<>
!<>================================  body (total_cluster)
!<>
!<>  <fitting-doc-tag-begin> <unsupported-body-block>
!<>
old_dec_EIG = 0
old_dec_HS = 0
call int_matrix_set( alloc__R, alloc__R, size_R, size_R, old_short_R_R, 1 )
R_scale = global__R_scale
E_scale = global__E_scale
flag_SC = 0
if ( MODEL_neutral_SC /= 0 ) flag_SC = 1
if ( charge_config /= 0 ) flag_SC = 1
n_RF = 0
do i_R = 1, size_R
  flag_free = 1
  do k_R = i_R - 1, 1, -1
    if ( Free_R( k_R ) == Free_R( i_R ) ) then
      flag_free = 0
    end if
  end do
  if ( flag_free == 1 ) then
    n_RF = n_RF + 1
  end if
end do
size_RF = n_RF
do i_RF = 1, size_RF
  i_R_RF( i_RF ) = -1
end do
```

```
float_tot = 0.000_8
degen_last = 0
do i_R = 1, size_R
  i_RF_R( i_R ) = Free_R( i_R )
  if ( i_R_RF( i_RF_R( i_R ) ) == -1 ) then
    i_R_RF( i_RF_R( i_R ) ) = i_R
  end if
  float_tot = float_tot + f_int( get_n_val( elem_R( i_R ) ) )
  if ( Free_R( i_R ) == size_RF ) then
    degen_last = degen_last + 1
  end if
end do
float_tot = float_tot - f_int( charge_config )
size_RF1 = size_RF - 1
if ( size_RF1 == 0 ) flag_SC = 0
X_init = f_int( fix_EL )
X_change = tune_elec_N_change

if (wrap__index_iso /= save_iso) then
  do i_RF = 1, size_RF
    X_init_RF( i_RF ) = f_int( get_n_val( elem_R( i_R_RF( i_RF ) ) ) )
  end do
  if (wrap__index_iso == 82 ) then
    X_init_RF( 1 ) = 3.87721401_8
    X_init_RF( 2 ) = 4.12054316_8
    X_init_RF( 3 ) = 3.89416424_8
    X_init_RF( 4 ) = 3.97334306_8
    X_init_RF( 5 ) = 4.08827692_8
    X_init_RF( 6 ) = 3.6599226_8
  endif
  if (wrap__index_iso == 84 ) then
    X_init_RF( 1 ) = 5.23366689_8
  endif
  if (wrap__index_iso == 85 ) then
    X_init_RF( 1 ) = 5.26165340_8
    X_init_RF( 2 ) = 3.69981565_8
  endif
  if (wrap__index_iso == 87 ) then
    X_init_RF( 1 ) = 5.31224911_8
    X_init_RF( 2 ) = 3.69416810_8
    X_init_RF( 3 ) = 4.03936440_8
  endif
  if (wrap__index_iso == 88 ) then
    X_init_RF( 1 ) = 5.46332360_8
    X_init_RF( 2 ) = 3.04908541_8
  endif
  if (wrap__index_iso == 89 ) then
    X_init_RF( 1 ) = 5.29908537_8
    X_init_RF( 2 ) = 3.66869170_8
    X_init_RF( 3 ) = 3.93112135_8
  endif
  if (wrap__index_iso == 90 ) then
    X_init_RF( 1 ) = 5.06291869_8
    X_init_RF( 2 ) = 4.08676190_8
    X_init_RF( 3 ) = 3.59988425_8
  endif
  if (wrap__index_iso == 91 ) then
    X_init_RF( 1 ) = 4.76712786_8
    X_init_RF( 2 ) = 5.26680042_8
    X_init_RF( 3 ) = 4.07296647_8
    X_init_RF( 4 ) = 3.62357902_8
  endif
  if (wrap__index_iso == 92 ) then
    X_init_RF( 1 ) = 4.32355135_8
```

```
  X_init_RF( 2 ) = 3.55442722_8
endif
if (wrap__index_iso == 95 ) then
  X_init_RF( 1 ) = 2.61471199_8
endif
if (wrap__index_iso == 96 ) then
  X_init_RF( 1 ) = 5.17036085_8
endif
if (wrap__index_iso == 97 ) then
  X_init_RF( 1 ) = 5.08522652_8
  X_init_RF( 2 ) = 5.01177907_8
endif
if (wrap__index_iso == 98 ) then
  X_init_RF( 1 ) = 2.73883116_8
endif
if (wrap__index_iso == 99 ) then
  X_init_RF( 1 ) = 2.67278652_8
endif
if (wrap__index_iso == 101 ) then
  X_init_RF( 1 ) = 5.34017854_8
  X_init_RF( 2 ) = 2.65564190_8
endif
if (wrap__index_iso == 102 ) then
  X_init_RF( 1 ) = 2.56369328_8
  X_init_RF( 2 ) = 5.44609954_8
endif
if (wrap__index_iso == 104 ) then
  X_init_RF( 1 ) = 2.61227530_8
  X_init_RF( 2 ) = 5.90704434_8
  X_init_RF( 3 ) = 3.67571873_8
endif
if (wrap__index_iso == 105 ) then
  X_init_RF( 1 ) = 5.36659982_8
  X_init_RF( 2 ) = 2.71144796_8
  X_init_RF( 3 ) = 4.95702874_8
endif
if (wrap__index_iso == 106 ) then
  X_init_RF( 1 ) = 2.96089655_8
  X_init_RF( 2 ) = 2.78469856_8
endif
if (wrap__index_iso == 109 ) then
  X_init_RF( 1 ) = 2.52233847_8
endif
if (wrap__index_iso == 110 ) then
  X_init_RF( 1 ) = 5.23726625_8
endif
if (wrap__index_iso == 111 ) then
  X_init_RF( 1 ) = 2.86080407_8
endif
if (wrap__index_iso == 112 ) then
  X_init_RF( 1 ) = 5.49707771_8
  X_init_RF( 2 ) = 2.76778807_8
endif
if (wrap__index_iso == 116 ) then
  X_init_RF( 1 ) = 5.11887889_8
  X_init_RF( 2 ) = 5.33782679_8
endif
if (wrap__index_iso == 117 ) then
  X_init_RF( 1 ) = 5.34709904_8
  X_init_RF( 2 ) = 2.63194784_8
endif
if (wrap__index_iso == 118 ) then
  X_init_RF( 1 ) = 5.51914985_8
  X_init_RF( 2 ) = 2.52686358_8
endif
```

```
  if (wrap__index_iso == 119 ) then
    X_init_RF( 1 ) = 2.91638028_8
    X_init_RF( 2 ) = 2.78975179_8
  endif
  if (wrap__index_iso == 120 ) then
    X_init_RF( 1 ) = 2.55051764_8
    X_init_RF( 2 ) = 5.49216826_8
  endif
  if (wrap__index_iso == 122 ) then
    X_init_RF( 1 ) = 2.41893401_8
  endif
  if (wrap__index_iso == 123 ) then
    X_init_RF( 1 ) = 2.46083473_8
  endif
  if (wrap__index_iso == 124 ) then
    X_init_RF( 1 ) = 2.36275454_8
  endif
  if (wrap__index_iso == 125 ) then
    X_init_RF( 1 ) = 2.32790471_8
  endif
  if (wrap__index_iso == 126 ) then
    X_init_RF( 1 ) = 4.51277470_8
    X_init_RF( 2 ) = 3.49856210_8
  endif
  save_iso = wrap__index_iso
endif

X_scale = f_int( fix_EL )
G_scale = tune_elec_N_change * f_int( fix_EL )
do i_RF = 1, size_RF
  X_scale_RF( i_RF ) = f_int( get_n_val( elem_R( i_R_RF( i_RF ) ) ) )
 G_scale_RF(i_RF)=tune_elec_N_change*f_int(get_n_val(elem_R(i_R_RF(i_RF))))
end do
X_toler = tune_elec_N_toler
G_toler = tune_elec_N_toler
G_accur = f_invert( tune_elec_N_change ) * f_float_accur( )
n_eval = 0
n_eval_int = 0
rev_eval = 1
X_change = 0.0_8
sites_prev_R=sites_X_R
!****************** Self Consistency Controls ******************
max_SC_steps = 500*((size_RF1-1)**2)+150
!*************************************************************
switcher =0
SC_method = 1
!!!!!!!!!!!!!!!!!!!! MAIN SC LOOP !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

do while ( rev_eval /= 0 )
!write(*,'(F12.6,F12.6,F12.6,F12.6)') X_eval_RF(1),&
!            X_eval_RF(2),G_eval_RF(1),G_eval_RF(2)
if ( flag_SC /= 0 ) then
  G_norm = 0.0_8
  do i_M=1,size_RF1
    G_norm = G_norm + (1/(1.0_8*size_RF1))*G_eval_RF(i_M)**2
  enddo
  if( size_RF1 <= 8 ) then
    if ( n_eval > max_SC_steps-5 .AND. switcher==0) then
      SC_method = 2  ! switch to Bisection when Broyden struggles.
      write(*,'(A29,I5,A27,I4,A1)') 'Switching to Bisection after ', &
n_eval_int,' Broyden steps on cluster #',wrap__index_iso,'.'
      switcher = 1
```

```
      n_eval_int=0
    endif
  endif
  call root_TEST_REVX( size_RF1, X_eval_RF, G_eval_RF,  &
    X_best_RF, G_best_RF, n_eval_int,rev_eval, err_tot, X_init_RF, &
    SC_method, X_scale, G_scale,G_accur )
  if ( err_tot( 1:1 ) == '-' ) then
   err_tot="--Backout: total_cluster >> << root_multi_rev.@@"<cat>TRIM(err_tot)
   write(*,'(A29,I6,A19,I4,A1)') 'Self-consistency error after ',n_eval_int, &
' steps on cluster #',wrap__index_iso,'.'
    write(*,'(A18)') 'Elem     Positions'
    do i_R = 1, size_R
      write(*,'(A1,A2,F18.12,F18.12,F18.12)') ' ', elem_R(i_R),&
                  sites_X_R(1,i_R),sites_X_R(2,i_R),sites_X_R(3,i_R)
    end do
    return
  end if
end if

if ( .NOT. flag_SC /= 0 ) then
  call float_array_set( size_RF1, X_eval_RF, float_tot / size_R )
  rev_eval = 1
  if ( n_eval + 1 > 1 ) rev_eval = 0
end if

if ( rev_eval /= 0 ) then
  Ln_allbutlast = 0.000_8
  do i_R = 1, size_R
    if ( i_RF_R( i_R ) /= size_RF ) then
      Ln_this_R( i_R ) = X_eval_RF( i_RF_R( i_R ) )
      Ln_allbutlast = Ln_allbutlast + Ln_this_R( i_R )
    end if
  end do
  do i_R = 1, size_R
    if ( i_RF_R( i_R ) == size_RF ) then
      Ln_this_R( i_R ) = ( float_tot - Ln_allbutlast ) / degen_last
    end if
  end do
  do i_R = 1, size_R
    Lz_this_R( i_R ) = f_int( get_n_val( elem_R( i_R ) ) )
  end do

  call model_eig_clust( alloc__clust_HS, act_EIG, energy_E, old_dec_HS, &
            old_size_HS, old_C_HS_qHS, size_R, elem_R, sites_X_R, &
            charge_config, Free_R, alloc__R, old_short_R_R, size_MP,&
            curr_EP, err_tot, n_eval, Ln_next_R, Ln_this_R, Lz_this_R )

  if ( err_tot( 1:1 ) == '-' ) then
    write( *, * ) "err_eig", err_tot
   err_tot="--Backout: total_cluster >> << model_eig_clust.@@"<cat>TRIM(err_tot)
    return
  end if
  int_elec = 0
  do i_R = 1, size_R
    int_elec = int_elec + get_n_val( elem_R( i_R ) )
  end do
  int_elec = int_elec - charge_config
  flag_occ = 1
  if ( int_elec <= 2 ) then
    do i_EIG = 1, act_EIG
      occupy_E( i_EIG ) = 0.000_8
    end do
    occupy_E( 1 ) = int_elec
    flag_occ = 0
  end if
  if ( int_elec >= 2 * ( act_EIG - 1 ) ) then
    do i_EIG = 1, act_EIG
      occupy_E( i_EIG ) = 2.000_8
    end do
```

```
    occupy_E( act_EIG ) = int_elec - 2 * ( act_EIG - 1 )
    flag_occ = 0
  end if
  if ( flag_occ == 1 ) then
    call occ_clust_front( act_EIG, occupy_E, energy_E, size_R, elem_R,&
            charge_config, E_scale, err_tot )
    if ( err_tot( 1:1 ) == '-' ) then
      write( *, * ) "err_occ"
    err_tot="--Backout: total_cluster >> << occ_clust_front.@@"<cat>TRIM(err_tot)
      return
    end if
  end if
  do i_RF = 1, size_RF
   G_eval_RF( i_RF ) = Ln_next_R( i_R_RF( i_RF ) ) - Ln_this_R( i_R_RF( i_RF ) )
  end do
  n_eval = n_eval + 1
  n_eval_int = n_eval_int + 1
  n_eval_tot = n_eval_tot + 1
end if
end do
!!!!!!!!!!!!!!! End of Main SC Loop !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!if (clust_curr .NE. wrap__index_iso) then
!  write(*,*) 'SC finished on cluser #', wrap__index_iso
! write(*,*) 'Used ',n_eval_int, ' steps for a total of ',n_eval_tot, ' steps.'
!  do i_RF = 1, size_RF1
!    write(*,'(F12.8)') X_eval_RF(i_RF)
!  end do
!  if ( wrap__index_iso == 2 ) Pause
!  clust_curr = wrap__index_iso
!endif


X_init_RF=X_eval_RF

call sum_A_A( E_tract, act_EIG, occupy_E, energy_E )

E_band = E_tract / f_int( size_R )

call cor_cluster( E_cor, size_R, elem_R, sites_X_R, alloc__R, old_short_R_R, &
  size_MP, curr_EP, err_tot, Ln_this_R, Lz_this_R )

if ( err_tot( 1:1 ) == '-' ) then
  write( *, * ) "err_cor"
 err_tot = "--Backout: total_cluster >> << cor_cluster.@@" <cat> TRIM( err_tot )
  return
end if

E_tot = E_band + E_cor

if ( old_dec_EIG /= 0 ) then
  call check_index(index_from_nat( act_EIG ),old_dec_EIG,"act_EIG",&
            "total_cluster")
  call float_array_copy( act_EIG, old_energy_EIG, energy_E )
  old_size_EIG = act_EIG
end if

!if (size_RF1 > 0 ) PAUSE

!<>
!<>  <fitting-doc-tag-end> <unsupported-body-block>
!<>
!<>================================  total_cluster
end subroutine
!<>
!<>
!<>
!<>
```

```
!<>================================
!<>
!<>  Fortran source code subroutines.
!<>
!<>  Filename: root_multi.F
!<>
!<>  Description: <SC routines> <need-file-description>
!<>
!<>  Documentation: <fitting-doc-tag-short> <need-doc-description>
!<>
!<>  Arranged by: Lyle Smith   rev. 1   July  1, 2010
!<>                            rev. 2   Jan. 19, 2010
!<>                                 generalize to N dimensions
!<>                       rev. 2.2 Mar. 21, 2011 - replace STOP with RETURN
!<>                     rev. 2.3 May  24, 2011 - fixed a bug in error output.
!<>                         rev. 2.3.1 July 12, 2011 - now runs on the CRC.
!<>
!<>  There are 2 self-consistency schemes included in this version:
!<>  1. the Broyden Method
!<>  2. Generalized Bisection (limited to 8 or less degrees of freedom)
!<>
!<>  Which method is used is controlled by the variable 'SC_method',
!<>  with 1 for Broyden and 2 for Bisection.
!<>
!<>  rev. 2 implements the bisection method up to 8 dimensions.  Higher
!<>  dimensions can be solved by increasing the size of the storage
!<>  matrices.
!<>
!<>================================
!<>
!<>
!<>
!<>
!<>================================  root_TEST_REVX
!<>
!<>  <fitting-doc-tag-short> <need-sub-description>
!<>
!<>  <fitting-doc-tag-short> <unsupported-sub-declare>
subroutine root_TEST_REVX( ord_M, X_eval_M, G_eval_M, X_best_M,&
           G_best_M,n_eval,continue_eval, err_root, X_init_M,&
           SC_method,X_scale, G_scale, G_accur)
!<>
!<>  <fitting-doc-tag-begin> <supported-use-block>
!<>
use global_module
!<>
!<>  <fitting-doc-tag-end> <supported-use-block>
!<>
!<>================================  declarations
!<>
!<>================  arguments
!<>
IMPLICIT NONE
!<>--------  input
!<>
!<>  <fitting-doc-tag-begin> <unsupported-arg-block>
!<>
integer( kind = 4 ) :: ord_M
real( kind = 8 ) :: G_eval_M( ord_M )
integer( kind = 4 ) :: n_eval
real( kind = 8 ) :: X_init_M( ord_M )
integer( kind = 4 ) :: SC_method
real( kind = 8 ) :: X_scale
real( kind = 8 ) :: G_scale
real( kind = 8 ) :: G_accur
!<>
!<>  <fitting-doc-tag-end> <unsupported-arg-block>
!<>
```

```
!<>--------  output
!<>
!<>  <fitting-doc-tag-begin> <unsupported-arg-block>
!<>
real( kind = 8 ) :: X_eval_M( ord_M )
real( kind = 8 ) :: X_best_M( ord_M )
real( kind = 8 ) :: G_best_M( ord_M )
integer( kind = 4 ) :: continue_eval
character( len = * ) :: err_root
!<>
!<>  <fitting-doc-tag-end> <unsupported-arg-block>
!<>
!<>================  static local variables
!<>
!<>  <fitting-doc-tag-begin> <unsupported-local-block>
!<>
integer( kind = 4 ), parameter :: alloc_M = 20
integer( kind = 4 ), parameter :: SPEC_done_Jac = 2
integer( kind = 4 ), parameter :: SPEC_done_Rone = 3
integer( kind = 4 ), parameter :: SPEC_done_dir = 4
integer( kind = 4 ), parameter :: SPEC_done_fwd = 3
integer( kind = 4 ), parameter :: SPEC_done_init = 2
integer( kind = 4 ), parameter :: SPEC_done_none = 1
integer( kind = 4 ), parameter :: SPEC_done_opt = 5
integer( kind = 4 ), parameter :: SPEC_done_rev = 4
integer( kind = 4 ), parameter :: SPEC_need_Jac = 1
integer( kind = 4 ), parameter :: SPEC_need_Rone = 2
integer( kind = 4 ), parameter :: SPEC_need_dir = 3
integer( kind = 4 ), parameter :: SPEC_need_fwd = 2
integer( kind = 4 ), parameter :: SPEC_need_init = 1
integer( kind = 4 ), parameter :: SPEC_need_opt = 4
integer( kind = 4 ), parameter :: SPEC_need_rev = 3
!<>  control parameters for bisection
integer( kind = 4 ), parameter :: SPEC_need_inbx = 1
integer( kind = 4 ), parameter :: SPEC_need_main = 2
integer( kind = 4 ), parameter :: SPEC_need_relx = 3
integer( kind = 4 ), parameter :: SPEC_need_diag = 4
integer( kind = 4 ), parameter :: SPEC_need_side = 5
integer( kind = 4 ), parameter :: SPEC_done_inbx = 1
integer( kind = 4 ), parameter :: SPEC_done_main = 2
integer( kind = 4 ), parameter :: SPEC_done_relx = 3
integer( kind = 4 ), parameter :: SPEC_done_diag = 4
integer( kind = 4 ), parameter :: SPEC_done_side = 5
!<>  saved local variables
integer( kind = 4 ) :: spec_done ; save spec_done
integer( kind = 4 ) :: spec_need ; save spec_need
integer( kind = 4 ) :: flag_Rone ; save flag_Rone
integer( kind = 4 ) :: flag_prev ; save flag_prev
integer( kind = 4 ) :: n_steps ; save n_steps
integer( kind = 4 ) :: n_Rone ; save n_Rone
integer( kind = 4 ) :: n_Rone_reset ; save n_Rone_reset
integer( kind = 4 ) :: n_Jac ; save n_Jac
integer( kind = 4 ) :: continue_Jac ; save continue_Jac
integer( kind = 4 ) :: n_opt ; save n_opt
integer( kind = 4 ) :: continue_opt ; save continue_opt
real( kind = 8 ) :: J_calc_M_M( alloc_M, alloc_M ) ; save J_calc_M_M
real( kind = 8 ) :: J_vtile_M_M( alloc_M, alloc_M ) ; save J_vtile_M_M
real( kind = 8 ) :: J_vttwo_M_M( alloc_M, alloc_M ) ; save J_vttwo_M_M
real( kind = 8 ) :: X_Jac_M( alloc_M ) ; save X_Jac_M
real( kind = 8 ) :: G_Jac_M( alloc_M ) ; save G_Jac_M
real( kind = 8 ) :: X_dir_M( alloc_M ) ; save X_dir_M
```

```
real( kind = 8 ) :: X_opt_M( alloc_M ) ; save X_opt_M
real( kind = 8 ) :: XG_vtile_M( alloc_M ) ; save XG_vtile_M
real( kind = 8 ) :: XG_vttwo_M( alloc_M ) ; save XG_vttwo_M
real( kind = 8 ) :: X_delta_M( alloc_M ) ; save X_delta_M
real( kind = 8 ) :: G_delta_M( alloc_M ) ; save G_delta_M
real( kind = 8 ) :: X_prev_M( alloc_M ) ; save X_prev_M
real( kind = 8 ) :: G_prev_M( alloc_M ) ; save G_prev_M
real( kind = 8 ) :: G_best ; save G_best
real( kind = 8 ) :: G_Jac ; save G_Jac
real( kind = 8 ) :: G_eval ; save G_eval
real( kind = 8 ) :: P_eval ; save P_eval
real( kind = 8 ) :: P_opt, G_opt ; save P_opt, G_opt
real( kind = 8 ) :: P_init ; save P_init
real( kind = 8 ) :: P_change ; save P_change
real( kind = 8 ) :: P_scale ; save P_scale
real( kind = 8 ) :: Jac_scale ; save Jac_scale
real( kind = 8 ) :: P_toler ; save P_toler
real( kind = 8 ) :: X_left ; save X_left
real( kind = 8 ) :: X_right ; save X_right
real( kind = 8 ) :: G_left ; save G_left
real( kind = 8 ) :: X_corners( 256, 8 ) ; save X_corners
real( kind = 8 ) :: G_corners( 256, 8 ) ; save G_corners
integer( kind = 4 ) :: M_corners( 256, 8 ) ; save M_corners
integer( kind = 4 ) :: Ord_corners( 1024, 2 ) ; save Ord_corners
integer( kind = 4 ) :: Ord_diag( 128, 2 ) ; save Ord_diag
integer( kind = 4 ) :: init_corner ; save init_corner
integer( kind = 4 ) :: bigger_box ; save bigger_box
integer( kind = 4 ) :: loop_Rone ; save loop_Rone
integer( kind = 4 ) :: point_Rone ; save point_Rone
integer( kind = 4 ) :: cor_match ; save cor_match
integer( kind = 4 ) :: Ord_curr ; save Ord_curr
integer( kind = 4 ) :: tot_SC_num ; save tot_SC_num
integer( kind = 4 ) :: main_loop_number ; save main_loop_number
integer( kind = 4 ) :: relx_loop_number ; save relx_loop_number
integer( kind = 4 ) :: diag_loop_number ; save diag_loop_number
!<>
!<>  <fitting-doc-tag-end> <unsupported-local-block>
!<>
!<>================  local variables
!<>
!<>  <fitting-doc-tag-begin> <unsupported-local-block>
!<>
integer( kind = 4 ) :: i_M,j_M,k_M,i_dist,Ord_min_diag,Ord_max_diag
integer( kind = 4 ) :: t_M,m_M,good_corner,left_corner,right_corner
integer( kind = 4 ) :: Ord_next, i_Ord, max_SC_steps
real( kind = 8 ) :: G_norm, F_scale, MaxStepSize, JacScale
real( kind = 8 ) :: X_dist( ord_M*2**(ord_M-1) )
real( kind = 8 ) :: X_diag( 2**(ord_M-1) )
integer( kind = 4 ) :: Perm_M( alloc_M )
integer( kind = 4 ) :: LWORK
integer( kind = 4 ) :: WORK ( ord_M*5 )
integer( kind = 4 ) :: errc_DGESV
integer( kind = 4 ) :: errc_DGETRI
integer( kind = 4 ) :: errc_DGETRF
integer( kind = 4 ) :: query_backout
integer( kind = 4 ) :: deb
real( kind = 8 ) :: f_int
real( kind = 8 ) :: f_frac
real( kind = 8 ) :: f_sqrt_f
real( kind = 8 ) :: f_float_accur
real( kind = 8 ) :: SC_min_val, SC_max_val,G_toler
```

```
real( kind = 8 ) :: X_norm,X_relax
real( kind = 8 ) :: X_eval_temp( ord_M )
!<>
!<>  <fitting-doc-tag-end> <unsupported-local-block>
!<>
!<>================================  body (root_TEST_REVX)
!<>
!<>  <fitting-doc-tag-begin> <unsupported-body-block>
!<>
!**************  Initialize Variables to prevent NSLEF **********
!*****  Output Variables *****
G_best_M( ord_M ) = 0.0_8
continue_eval = 0
err_root = '0'
!*****  Saved Local Variables *****
if ( n_eval == 0 ) then
!**********  Integers  ************
  tot_SC_num = tot_SC_num + 1
  spec_done = SPEC_done_none ; spec_need = SPEC_need_Jac
  n_Jac = 0 ; n_opt = 0 ; n_Rone = 0; n_Rone_reset = 7
  flag_Rone = 0 ; flag_prev = 0
  n_steps = 0; M_corners=0; Ord_corners=0
  continue_Jac = 1; continue_opt = 1
  Ord_diag = 0; init_corner = 0; bigger_box = 0
  loop_Rone = 0; point_Rone = 0; cor_match = 0
  Ord_curr = 0; deb = 0; diag_loop_number =0
  main_loop_number = 0; relx_loop_number =0
!************  Reals  *************
  J_calc_M_M = 0.0_8; J_vtile_M_M = 0.0_8
  J_vttwo_M_M = 0.0_8; X_Jac_M = 0.0_8; G_Jac_M = 0.0_8
  X_dir_M = 0.0_8
  X_opt_M = 0.0_8; XG_vtile_M = 0.0_8; XG_vttwo_M = 0.0_8
  X_delta_M = 0.0_8; G_delta_M = 0.0_8; X_prev_M = 0.0_8
  G_prev_M = 0.0_8; X_corners = 0.0_8; G_corners = 0.0_8
  G_best = 0.0_8; G_Jac = 0.0_8; G_eval = 0.0_8; P_eval = 0.0_8
  P_opt = 0.0_8; G_opt = 0.0_8; P_init = 0.0_8
  P_change = 0.0_8; P_scale = 0.0_8; Jac_scale = 0.0_8
  P_toler = 0.0_8; X_left = 0.0_8; X_right = 8.0_8
  G_left = 1.0_8
  call float_array_copy( ord_M, X_eval_M, X_init_M )
  call float_array_copy( ord_M, X_opt_M, X_init_M )
  call float_array_copy( ord_M, X_prev_M, X_init_M )
  call float_array_copy( ord_M, X_best_M, X_init_M )
  Jac_scale = X_scale
end if
!*****  Unsaved Local Variables *****
  Perm_M = 0 ; WORK = 0
  i_M = 0 ; j_M = 0 ; k_M = 0 ; t_M = 0 ; m_M = 0
  i_dist = 0; i_Ord = 0 ; LWORK = 0
  Ord_min_diag = 0 ; Ord_max_diag = 0; Ord_next = 0
  good_corner = 0 ; left_corner = 0 ; right_corner = 0
  errc_DGESV = 0 ; errc_DGETRI = 0 ; errc_DGETRF = 0
  query_backout = 0 ; deb = 0; max_SC_steps = 0
  X_dist = 0.0_8;  X_diag = 0.0_8; X_eval_temp = 0.0_8
  G_norm = 0.0_8; F_scale = 0.0_8; MaxStepSize = 0.0_8
  JacScale = 0.0_8
  SC_min_val = 0.0_8; SC_max_val = 0.0_8
  X_norm = 0.0_8; X_relax = 0.0_8
!**************  All Variables Initialized *************


!****************** Self Consistency Controls ******************
max_SC_steps = 500*((ord_M-1)**2)+700
```

```
G_toler = 8.0_8      ! accuracy = 1/(10^G_toler)
deb = 0              ! set to 1 for verbose output.
SC_min_val = -1.0_8  ! starting boundary for bisection box
                     !   minimum number of electrons on an atom.
SC_max_val = 16.0_8  ! starting boundary for bisection box
                     !   maximum number of electrons on an atom.
!****************************************************************
!if (n_eval >= 885 .AND. tot_SC_num==22 ) deb=1
if ( n_eval + 1 > max_SC_steps ) then
  G_norm = 0.0_8
  do i_M=1,ord_M
    G_norm = G_norm + (1/(1.0_8*ord_M))*G_eval_M(i_M)**2 !
  enddo
  write(*,*)'SC fail on cluster ',tot_SC_num,' with G_norm = ',G_norm
  call term_string_void( "ERR_TEST_steps" )
  err_root = "--Backout: root_TEST_REVX >> << n_steps.@@"<cat>TRIM(err_root)
  return
end if
!if (tot_SC_num==4602) deb =1
call float_array_copy( ord_M, X_eval_temp, X_eval_M )
!****************** One-dimensional Bisection ******************
if ( SC_method == 2 .AND. ord_M == 1 ) then
  if ( n_eval == 1 ) then
    if (G_left*G_eval_M(1) > 0 )then
      X_left = X_eval_M(1)
      X_eval_M(1) = X_eval_M(1) + 0.05
    elseif (G_left*G_eval_M(1) < 0 ) then
      X_right = X_eval_M(1)
      X_eval_M(1) = X_eval_M(1) - 0.05
    else
      X_right = X_eval_M(1) + 0.05
      X_left = X_eval_M(1) - 0.05
    endif
  end if
  if ( n_eval > 1 ) then
    if (G_left*G_eval_M(1) > 0 )then
      X_left = X_eval_M(1)
    else
      X_right = X_eval_M(1)
    endif
      X_eval_M(1) = 0.5_8*(X_left + X_right)
  end if
  G_norm = G_eval_M(1)**2
  if ( n_eval > 0 ) then
    continue_eval = 1
    if ( G_norm < 10**(-2 * G_toler ) ) then
      continue_eval = 0
    end if
  end if
!****************** N-dimensional Bisection ********************
elseif ( SC_method == 2 .AND. ord_M > 1 ) then
  if ( n_eval == 0 ) then   !initial setup
    !if (deb==1) write(*,*)'X_corners Matrix = '
    do j_M=1,2**ord_M
      do i_M=1,ord_M
        M_corners(j_M,i_M)=(-1)**((j_M-1)/(2**(ord_M-i_M))+1)
        if(M_corners(j_M,i_M)<0) then
          X_corners(j_M,i_M)=SC_min_val
          G_corners(j_M,i_M)=-1.0_8 * SC_max_val
        else
          X_corners(j_M,i_M)=SC_max_val
          G_corners(j_M,i_M)=SC_max_val
```

```
      endif
    enddo
   !if (deb==1)  write(*,'(F9.5,F9.5,F9.5,F9.5)') X_corners(j_M,1),&
  !                    X_corners(j_M,2),X_corners(j_M,3),X_corners(j_M,4)
   enddo
   !if (deb==1) then
   !  write(*,*)'M_corners Matrix = '
   !  do j_M=1,2**ord_M
   !    write(*,'(I5,I5,I5,I5)') M_corners(j_M,1),M_corners(j_M,2), &
   !                             M_corners(j_M,3),M_corners(j_M,4)
   !  enddo
   !  write(*,*)'Ord_corners Matrix = '
   !endif
   i_M=0
   do j_M=1,ord_M
     do t_M=1,(2**j_M)-1,2
       do m_M=1,2**(ord_M-j_M)
         i_M=i_M+1
         Ord_corners(i_M,1)=(t_M-1)*(2**(ord_M-j_M))+m_M
         Ord_corners(i_M,2)=Ord_corners(i_M,1)+(2**(ord_M-j_M))
         !if(deb==1)write(*,'(I4.1,I4.1)') Ord_corners(i_M,1),&
           !Ord_corners(i_M,2)
       enddo
     enddo
   enddo
   !if (deb==1) write(*,*)'Ord_diag Matrix = '
   do j_M=1,2**(ord_M-1)
     Ord_diag(j_M,1)=j_M
     Ord_diag(j_M,2)=(2**ord_M)+1-j_M
     !if (deb==1) write(*,'(I4.1,I4.1)') Ord_diag(j_M,1),Ord_diag(j_M,2)
   enddo
   Ord_curr = 0 ;  Ord_next = 1 ; main_loop_number = 0 ; loop_Rone = 2
   cor_match = 0; spec_need = SPEC_need_inbx; spec_done = 0
   relx_loop_number = 0; diag_loop_number = 0
   Ord_min_diag=0;Ord_max_diag=0
   init_corner = 1; good_corner = 0
   bigger_box = 0
   X_dist = 100.0_8
   do i_M=1,ord_M
     X_eval_M(i_M) = X_init_M(i_M) - 0.00001_8*M_corners(1,i_M)
   enddo
endif    !end of initial setup
if ( n_eval > 0 ) then
 !if(spec_need /= SPEC_need_inbx .AND. deb==1 ) then ! write data
 ! PAUSE
 ! write(*,*)
 ! write(*,*)'n_eval = ',n_eval
 ! write(*,'(A15,F15.11,F15.11,F15.11)')'prev X_eval_M = ', X_eval_M
 ! write(*,'(A15,F15.11,F15.11,F15.11)')'     G_eval_M = ', G_eval_M
 ! write(*,'(A12,F9.5,F9.5,F9.5)')'X_corners = ',X_corners(1,1),&
 ! X_corners(1,2),X_corners(1,3)
 ! do i_M=2,2**ord_M
 !  write(*,'(A12,F9.5,F9.5,F9.5)')'            ',X_corners(i_M,1),&
 ! X_corners(i_M,2),X_corners(i_M,3)
 ! enddo
  !write(*,'(A12,F9.5,F9.5,F9.5,F9.5)') 'G_corners = ', G_corners(1,1), &
  !   G_corners(1,2),G_corners(1,3),G_corners(1,4)
  !do i_M=2,2**ord_M
 !  write(*,'(A12,F9.5,F9.5,F9.5,F9.5)')'           ', G_corners(i_M,1), &
  !    G_corners(i_M,2),G_corners(i_M,3),G_corners(i_M,4)
  !enddo
 !endif   ! end of write data
 if ( spec_need == SPEC_need_inbx ) then     ! initial box setup
  ! Test corner against M matrix
 !if (deb==1) write(*,'(A12,F9.5,F9.5,F9.5)') 'X_eval_M  = ',X_eval_M(1),&
```

```
!   X_eval_M(2),X_eval_M(3)
!if (deb==1) write(*,'(A12,F9.5,F9.5,F9.5)') 'G_eval_M  = ',G_eval_M(1),&
!   G_eval_M(2),G_eval_M(3)
!if (deb==1) write(*,'(A12,I5,I5,I5)') 'M_corners = ',  &
!M_corners(init_corner,1),M_corners(init_corner,2),&
!            M_corners(init_corner,3)
good_corner=0
do i_M=1,ord_M
if (G_eval_M(i_M)*M_corners(init_corner,i_M)>0)good_corner=good_corner+1
enddo
if (good_corner == ord_M ) then
  do i_M=1,ord_M
    X_corners(init_corner,i_M) = X_eval_M(i_M)
    G_corners(init_corner,i_M) = G_eval_M(i_M)
  enddo
  init_corner = init_corner + 1
  !bigger_box = 0   !optimize each corner separately.
else
  bigger_box = bigger_box + 1
endif
do i_M=1,ord_M
  if (bigger_box < 6 ) then
    X_eval_M(i_M) = X_init_M(i_M)&
     -(10**(bigger_box))*0.00001_8*M_corners(init_corner,i_M)
  elseif(bigger_box == 6) then
    X_eval_M(i_M) = X_init_M(i_M)-5.0_8*M_corners(init_corner,i_M)
  elseif(bigger_box == 7) then
    X_eval_M(i_M) = X_init_M(i_M)-10.0_8*M_corners(init_corner,i_M)
  elseif(bigger_box > 7 ) then
    if (M_corners(init_corner,i_M) <0) then
      X_eval_M(i_M) = SC_max_val
    else
      X_eval_M(i_M) = SC_min_val
    endif
  endif
enddo
if ( bigger_box > 8 )  then
  write(*,*)'Initial box is not big enough.'
  continue_eval = 0
  return
endif
if ( init_corner > 2**ord_M ) then ! initial box setup is complete
  spec_done = SPEC_done_inbx
  Ord_curr = 1
 X_eval_M = 0.5_8*( X_corners(1,:) + X_corners(5,:) )  ! First bisection
  !if( deb==1 )  then
  !  write(*,*)'steps used for initial box = ',bigger_box,n_eval
 !  write(*,'(A12,F9.5,F9.5,F9.5,F9.5)') 'X_corners = ', X_corners(1,1), &
  !     X_corners(1,2),X_corners(1,3),X_corners(1,4)
  !  do i_M=2,2**ord_M
 !    write(*,'(A12,F9.5,F9.5,F9.5,F9.5)')'         ', X_corners(i_M,1), &
  !      X_corners(i_M,2),X_corners(i_M,3),X_corners(i_M,4)
  !  enddo
  !  write(*,'(A13,I7)') ' spec_need = ', spec_need
  !  write(*,'(A13,I7)') ' spec_done = ', spec_done
  !endif
 endif
endif    ! initial box setup
!  Test box to determine next X_eval_M
if ( spec_need > 1 ) then
  !if ( deb == 1 ) then
  !  write(*,*)'Test current bisection point.'
    !write(*,*)'Ord_curr = ',Ord_curr
    !do j_M=1,ord_M*2**(ord_M-1)
    !  write(*,'(I4.1,I4.1)') Ord_corners(j_M,1),Ord_corners(j_M,2)
```

```
       !enddo
!write(*,*)'Current corners', Ord_corners(Ord_curr,1),Ord_corners(Ord_curr,2)
!write(*,'(A12,I9,I9,I9)')       'M_corner(1)=',&
!            M_corners(Ord_corners(Ord_curr,1),1)&
! ,M_corners(Ord_corners(Ord_curr,1),2),M_corners(Ord_corners(Ord_curr,1),3)
!write(*,'(A12,I9,I9,I9)')       'M_corner(2)=',&
!            M_corners(Ord_corners(Ord_curr,2),1)&
! ,M_corners(Ord_corners(Ord_curr,2),2),M_corners(Ord_corners(Ord_curr,2),3)
!write(*,'(A12,F9.3,F9.3,F9.3)') 'G_eval_M  = ', &
!            G_eval_M(1),G_eval_M(2),G_eval_M(3)
     !endif
     good_corner=0; left_corner = 0
     do i_M=1,ord_M
if(G_eval_M(i_M)*M_corners(Ord_corners(Ord_curr,1),i_M)>0)&
           left_corner=left_corner+1
if(G_eval_M(i_M)*M_corners(Ord_corners(Ord_curr,2),i_M)>0)&
           right_corner=right_corner+1
     enddo
     if (left_corner == ord_M ) then
!************** Signs match first corner in Ord_corners matrix **************
      !if( deb==1 ) write(*,*)'Signs match first corner in Ord_corners matrix'
       cor_match = Ord_corners(Ord_curr,1)
       X_corners(Ord_corners(Ord_curr,1),:) = X_eval_M
       G_corners(Ord_corners(Ord_curr,1),:) = G_eval_M
       spec_need = SPEC_need_main
     elseif (right_corner == ord_M) then
!************** Signs match second corner in Ord_corners matrix ***********
     !if( deb==1 ) write(*,*)'Signs match second corner in Ord_corners matrix'
       X_corners(Ord_corners(Ord_curr,2),:) = X_eval_M
       G_corners(Ord_corners(Ord_curr,2),:) = G_eval_M
       cor_match = Ord_corners(Ord_curr,2)
       spec_need = SPEC_need_main
     else
!************** No sign match - relax needed. *************************
       !if( deb==1 ) write(*,*)'No sign match - relax needed.'
       spec_need = SPEC_need_relx
       if (spec_done /= SPEC_done_relx) relx_loop_number = 0
     endif
     do i_dist=1,ord_M*2**(ord_M-1)
        X_dist(i_dist)=(X_corners(Ord_corners(i_dist,1),1)&
            -X_corners(Ord_corners(i_dist,2),1))**2&
            +(X_corners(Ord_corners(i_dist,1),2)&
            -X_corners(Ord_corners(i_dist,2),2))**2
     enddo
     do i_dist=1,2**(ord_M-1)
        X_diag(i_dist)=(X_corners(Ord_diag(i_dist,1),1)&
            -X_corners(Ord_diag(i_dist,2),1) )**2 &
            +( X_corners(Ord_diag(i_dist,1),2) &
            -X_corners(Ord_diag(i_dist,2),2) )**2
       !X_diag(i_dist)=SQRT( X_diag(i_dist))
     enddo
     !if (maxval(X_diag)/minval(X_diag) > 7 ) then
     !  !spec_need = SPEC_need_diag     ! diagonal fix routine currently off
     !  if (spec_done /= SPEC_done_diag) diag_loop_number = 0
     !endif
     !if( deb==1 )  then
     !  write(*,*)'Distance Ratio = ',maxval(X_dist)/minval(X_dist)
     !  write(*,*)'Distance Matrix = '
     !  write(*,'(F19.15,F19.15,F19.15,F19.15)') &
     !                              X_dist(1),X_dist(2),X_dist(3),X_dist(4)
     !  write(*,'(A19,F19.15,F19.15)') '            ',X_diag(1),X_diag(2)
     !  write(*,*) 'X_corners = ', X_corners(1,:)
```

```
  !  write(*,*) '            ', X_corners(2,:)
  !  write(*,*) '            ', X_corners(3,:)
  !  write(*,*) '            ', X_corners(4,:)
  !  write(*,'(A13,I7)') ' spec_need = ', spec_need
  !  write(*,'(A13,I7)') ' spec_done = ', spec_done
  !  !PAUSE
  !endif
endif

if ( spec_need == SPEC_need_main ) then
  !if( deb==1 ) write(*,*)'Entering main subroutine.'
  main_loop_number = main_loop_number + 1
  Ord_next = mod(main_loop_number,ord_M*2**(ord_M-1)) +1
  !if( deb==1 ) write(*,*)'Main Loop Number = ',main_loop_number
  !if( deb==1 ) write(*,*)'Ord_next = ',Ord_next
  do while ( maxval(X_dist)/X_dist(Ord_next) > 50 )
    !if( deb==1 ) write(*,*)'Skipping simplex #',Ord_next
    main_loop_number = main_loop_number + 1
    Ord_next  = mod(main_loop_number,ord_M*2**(ord_M-1)) +1
  enddo
  Ord_curr = Ord_next
  X_eval_M = 0.5_8*(X_corners(Ord_corners(Ord_next,1),:)&
         +X_corners(Ord_corners(Ord_next,2),:))
  !if( deb==1 )  then
  ! write(*,'(A30,I7,I7,I7)') ' n_eval, Ord_curr, Corner # = ', &
  ! n_eval,Ord_curr,main_loop_number
   !write(*,'(A15,F9.5,F9.5,F9.5)')'G_eval_M = ', G_eval_M
   !write(*,'(A15,F9.5,F9.5,F9.5)')'next X_eval_M = ', X_eval_M
  !endif
  spec_done = SPEC_done_main
endif
if ( spec_need == SPEC_need_relx ) then
  !if( deb==1 )  write(*,*)'Entering relaxation subroutine.'
 !if(relx_loop_number==0)then !find reflection point on first relax only.
    !if( deb==1 )  write(*,*)'Finding new reflection point.'
    do i_Ord=1,2**ord_M
      left_corner =0
      do i_M=1,ord_M
     if (G_eval_M(i_M)*M_corners(i_Ord,i_M)>0) left_corner =left_corner+1
      enddo
      if (left_corner == ord_M) then
        point_Rone = i_Ord
      endif
    enddo
  !endif
  loop_Rone = loop_Rone + 1
  X_relax = 0.5_8
  relx_loop_number = relx_loop_number + 1
  !if( deb==1 )  then
  !  write(*,*)'Reflection point = ',point_Rone
    !write(*,*)'X_relax = ', X_relax
  !  write(*,*)'relx_loop_number = ', relx_loop_number
  !endif
  X_eval_M = X_eval_M + X_relax*(X_eval_M - X_corners(point_Rone,:))
  spec_done = SPEC_done_relx
  if (relx_loop_number > 7 ) then   !skip relax; go to next simplex
    main_loop_number = main_loop_number + 1
    Ord_next = mod(main_loop_number,ord_M*2**(ord_M-1)) +1
    Ord_curr = Ord_next
    X_eval_M = 0.5_8*(X_corners(Ord_corners(Ord_next,1),:)&
         +X_corners(Ord_corners(Ord_next,2),:))
    spec_done = SPEC_done_main
   !if( deb==1 )  write(*,*)'Skipping simplex #',Ord_curr,'.  Bad relax.'
```

```
    endif
   endif
   G_norm = 0.0_8
   do i_M=1,ord_M
     G_norm = G_norm + (1/(1.0_8*ord_M))*G_eval_M(i_M)**2 !
   enddo
   X_norm = 0.0_8
   do i_M=1,2**(ord_M-1)
     X_norm =(1/2**(1.0_8*ord_M-1))*&
          ( (X_corners(Ord_diag(i_M,1),1)-X_corners(Ord_diag(i_M,2),1))**2&
           + (X_corners(Ord_diag(i_M,1),2)&
           -X_corners(Ord_diag(i_M,2),2))**2)
   enddo
   !if(spec_need/=SPEC_need_inbx .AND. deb==1)write(*,*)'G_norm,X_norm = ',&
            G_norm, X_norm
   if ( spec_done == SPEC_done_inbx )   spec_need = SPEC_need_main
   if ( spec_done == SPEC_done_main )   spec_need = SPEC_need_main
  ! if ( spec_done == SPEC_done_relx )   spec_need = SPEC_need_main
  ! if ( spec_done == SPEC_done_side )   spec_need = SPEC_need_main
  endif  ! n_eval > 0  block
  !if (deb == 1 ) then
  !  write (*,*) X_init_M(1)
  !  write(*,'(A22,I7,I7,I7)') ' n_eval, Ord_curr, Corner # = ', n_eval,&
  !           Ord_curr,main_loop_number
  !  write(*,*) 'X_eval_M = ', X_eval_temp
  !  write(*,*) 'G_eval_M = ', G_eval_M
  !  write(*,*) 'X_corners = ', X_corners(1,:)
  !  write(*,*) '             ', X_corners(2,:)
  !  write(*,*) '             ', X_corners(3,:)
  !  write(*,*) '             ', X_corners(4,:)
  !  write(*,*) 'next X_eval_M = ', X_eval_M
  !  write(*,*)n_eval,G_norm,X_norm
  !  write(*,*)'Distance Matrix = '
  !  write(*,'(F18.15,F18.15,F18.15,F18.15)') X_dist(1),X_dist(2),&
  !           X_dist(3),X_dist(4)
  !  write(*,'(A18,F18.15,F18.15)') '             ',X_diag(1),X_diag(2)
  !  write(*,*)'Distance Ratio = ',maxval(X_dist)/minval(X_dist)
  !  PAUSE
  !endif
  if ( n_eval > 0 ) then
   continue_eval = 1
   if ( G_norm < 10**(-2 * G_toler ) ) then
     X_eval_M = X_eval_temp
     continue_eval = 0
     !PAUSE
   end if
  end if


elseif ( SC_method == 1 ) then    ! Broyden ****************************
!*************************************************************
if ( n_eval == 1 ) then
  call float_array_norm( G_norm, ord_M, G_eval_M, 'yes-mean' )
  G_best = G_norm
  call float_array_copy( ord_M, X_best_M, X_eval_M )
  call float_array_copy( ord_M, G_best_M, G_eval_M )
  call float_array_copy( ord_M, G_prev_M, G_eval_M )
end if
if ( n_eval > 1 ) then
  call float_array_norm( G_norm, ord_M, G_eval_M, 'yes-mean' )
  if ( G_norm < G_best ) then
```

```
    G_best = G_norm
    call float_array_copy( ord_M, X_best_M, X_eval_M )
    call float_array_copy( ord_M, G_best_M, G_eval_M )
  end if
end if
if ( spec_need == SPEC_need_Jac ) then
  call float_array_copy( ord_M, X_Jac_M, X_opt_M )
  call Jac_double_REVX(alloc_M,ord_M,X_eval_M,G_eval_M,J_calc_M_M,n_Jac, &
    continue_Jac, G_Jac_M, X_Jac_M, Jac_scale, G_accur )
  n_Jac = n_Jac + 1
  if ( .NOT. continue_Jac /= 0 ) then
    n_Jac = 0
    spec_done = SPEC_done_Jac ; spec_need = SPEC_need_dir
  end if
end if
if ( spec_need == SPEC_need_dir ) then
  call float_matrix_copy(alloc_M,alloc_M,ord_M,ord_M,J_vtile_M_M,J_calc_M_M)
  call float_array_copy( ord_M, XG_vtile_M, G_Jac_M )
  call DGESV( &
    ord_M, 1, &
    J_vtile_M_M, alloc_M, &
    Perm_M, &
    XG_vtile_M, alloc_M, &
    errc_DGESV )
  if ( errc_DGESV /= 0 ) then
   err_root = "--Backout: root_TEST_REVX >> << DGESV.@@" <cat> TRIM( err_root )
    call term_string_void( "ERR_TEST_DGESV" )
    Print*, "Jacobian is singular:"
    return
  end if
  LWORK =5*ord_M
  call DGETRI( ord_M, &
    J_vtile_M_M, alloc_M, Perm_M, &
    WORK, LWORK, &
    errc_DGETRI )
  if ( errc_DGETRI /= 0 ) then
   err_root = "--Backout: root_TEST_REVX >> << DGETRI.@@" <cat> TRIM( err_root )
    call term_string_void( "ERR_TEST_DGETRI" )
    return
  end if
  call float_matrix_copy(alloc_M,alloc_M,ord_M,ord_M,J_calc_M_M,J_vtile_M_M)
  call float_array_copy( ord_M, X_dir_M, XG_vtile_M )
  F_scale = - f_int( 1 )
  call float_array_scale( ord_M, X_dir_M, F_scale )
  spec_done = SPEC_done_dir ; spec_need = SPEC_need_opt
end if
if ( spec_need == SPEC_need_opt ) then
  P_init = global__float_zero
  P_change = f_int( 1 )
  P_scale = f_int( 1 )
  if ( G_norm < 1.0E-9_8 ) P_scale = 0.1d0
  P_toler = f_sqrt_f( f_float_accur( ) )
  G_toler = f_float_accur( )
  call sum_A_A( G_Jac, ord_M, G_Jac_M, G_Jac_M )
  G_Jac = f_frac( 1, 2 ) * G_Jac
  if ( n_opt > 0 ) then
    call sum_A_A( G_eval, ord_M, G_eval_M, G_eval_M )
    G_eval = f_frac( 1, 2 ) * G_eval
  end if
end if
if ( spec_need == SPEC_need_opt ) then
  call opt_Nash_REVX( P_eval, G_eval, P_opt, G_opt, n_opt, continue_opt,&
```

```
    err_root, P_init, P_change, P_scale, G_scale, P_toler, &
    G_toler )
  if ( err_root( 1:1 ) == '-' ) then
    err_root = "--Backout: root_TEST_REVX >> << opt_Nash_REVX.@@" &
            <cat> TRIM( err_root )
    return
  end if
  if ( n_opt == 0 .AND. P_eval /= global__float_zero ) then
    call bailout_string( "n_opt and P_eval in root_TEST_REVX." )
  end if
  if ( n_opt == 1 .AND. P_eval /= f_int( 1 ) ) then
    call bailout_string( "n_opt and P_eval in root_TEST_REVX." )
  end if
  if ( P_eval == global__float_zero ) then
    G_eval = G_Jac
    n_opt = n_opt + 1
    call opt_Nash_REVX( P_eval, G_eval, P_opt, G_opt, n_opt, continue_opt,&
      err_root, P_init, P_change, P_scale, G_scale, P_toler, &
      G_toler )
    if ( err_root( 1:1 ) == '-' ) then
      err_root = "--Backout: root_TEST_REVX >> << opt_Nash_REVX .@@" &
            <cat> TRIM( err_root )
      return
    end if
  end if
end if

if ( spec_need == SPEC_need_opt ) then
  if ( G_eval < 0.990_8 * G_Jac ) then
    continue_opt = 0
    if ( ord_M >=2 ) then
      Jac_scale = SQRT( G_eval )
      if ( Jac_scale < 1.0E-11_8 ) Jac_scale = 1.0E-10_8
    end if
  end if
  do i_M = 1, ord_M
    X_eval_M( i_M ) = X_Jac_M( i_M ) + P_eval * X_dir_M( i_M )
    X_opt_M( i_M ) = X_Jac_M( i_M ) + P_opt * X_dir_M( i_M )
  end do
  n_opt = n_opt + 1
  if ( .NOT. continue_opt /= 0 ) n_opt = 0
  if ( .NOT. continue_opt /= 0 ) then
    spec_done = SPEC_done_opt ; spec_need = SPEC_need_Jac
    if ( ord_M > 1 ) spec_need = SPEC_need_Rone
  end if
  if ( .NOT. continue_opt /= 0 ) then
    n_steps = n_steps + 1
  end if
end if

if ( n_Rone >= n_Rone_reset ) then
  spec_need = SPEC_need_Jac
  n_Rone = 0
end if

if ( spec_need == SPEC_need_Rone ) then
  call float_array_copy( ord_M, X_Jac_M, X_opt_M )
  call float_matrix_copy(alloc_M,alloc_M,ord_M,ord_M,J_vtile_M_M,J_calc_M_M)
  X_delta_M = X_opt_M - X_prev_M
  G_delta_M = G_eval_M-G_prev_M
  call sum_R_A( ord_M, ord_M, XG_vtile_M, J_vtile_M_M, G_delta_M )
  call sum_A_A( JacScale, ord_M, XG_vtile_M, X_delta_M )
  XG_vtile_M = X_delta_M - XG_vtile_M
  call sum_L_A( alloc_M, ord_M, XG_vttwo_M, J_vtile_M_M, X_delta_M )
  call outer_A_A( alloc_M, ord_M, J_vttwo_M_M, XG_vtile_M, XG_vttwo_M )
  J_calc_M_M = J_vtile_M_M + ( 1/JacScale )*J_vttwo_M_M
  call sum_R_A( ord_M, ord_M, XG_vtile_M, J_calc_M_M, G_Jac_M )
```

```
  call float_array_copy( ord_M, X_dir_M, XG_vtile_M )
  F_scale = - f_int( 1 )
  call float_array_scale( ord_M, X_dir_M, F_scale )
  call float_array_copy( ord_M, G_Jac_M, G_eval_M )
  G_prev_M = G_eval_M
  X_prev_M = X_opt_M
  spec_done = SPEC_done_Rone ; spec_need = SPEC_need_opt
  if ( JacScale ==0.0d0 ) spec_need = SPEC_need_Jac
  n_Rone = n_Rone + 1
end if
if ( n_eval > 0 ) then
  continue_eval = 1
  call float_array_norm( G_norm, ord_M, G_eval_M, 'yes-mean' )
  if ( G_norm < 1.0E-8_8 )  then
    continue_eval = 0
  end if
end if
else
  write(*,*) 'no SC method selected.'
  STOP
endif  !************************* not bisection ************************
!***************************************************************************

if ( n_eval == 0 ) then
  continue_eval = 1
end if

!<>
!<>  <fitting-doc-tag-end> <unsupported-body-block>
!<>
!<>============================  root_TEST_REVX ============================
!<>=========================================================================
!<>=========================================================================
end subroutine
!<>
!<>
!<>
!<>
!<>================================  Jac_double_REVX
!<>
!<>  <fitting-doc-tag-short> <need-sub-description>
!<>
!<>  <fitting-doc-tag-short> <unsupported-sub-declare>
subroutine Jac_double_REVX( dec_M, ord_M, X_eval_XM, G_eval_GM,&
             J_calc_GM_XM, n_eval,continue_eval, G_Jac_GM, X_Jac_XM,&
             X_scale, G_accur )
!<>
!<>  <fitting-doc-tag-begin> <supported-use-block>
!<>
use global_module
!<>
!<>  <fitting-doc-tag-end> <supported-use-block>
!<>
!<>================================  declarations
!<>
!<>===============  arguments
!<>
!<>--------  input
!<>
!<>  <fitting-doc-tag-begin> <unsupported-arg-block>
!<>
integer( kind = 4 ) :: dec_M
integer( kind = 4 ) :: ord_M
real( kind = 8 ) :: G_eval_GM( dec_M )
integer( kind = 4 ) :: n_eval
real( kind = 8 ) :: X_Jac_XM( dec_M )
real( kind = 8 ) :: X_scale
real( kind = 8 ) :: G_accur
```

```
!<>
!<>  <fitting-doc-tag-end> <unsupported-arg-block>
!<>
!<>--------  output
!<>
!<>  <fitting-doc-tag-begin> <unsupported-arg-block>
!<>
real( kind = 8 ) :: X_eval_XM( dec_M )
real( kind = 8 ) :: J_calc_GM_XM( dec_M, dec_M )
real( kind = 8 ) :: G_Jac_GM( dec_M )
integer( kind = 4 ) :: continue_eval
!<>
!<>  <fitting-doc-tag-end> <unsupported-arg-block>
!<>
!<>================  static local variables
!<>
!<>  <fitting-doc-tag-begin> <unsupported-local-block>
!<>
integer( kind = 4 ), parameter :: alloc_M = 20
integer( kind = 4 ) :: spec_need ; save spec_need
integer( kind = 4 ) :: spec_done ; save spec_done
integer( kind = 4 ) :: v_XM ; save v_XM
real( kind = 8 ) :: X_step ; save X_step
real( kind = 8 ) :: G_fwd_GM( alloc_M ) ; save G_fwd_GM
real( kind = 8 ) :: G_rev_GM( alloc_M ) ; save G_rev_GM
!<>
!<>  <fitting-doc-tag-end> <unsupported-local-block>
!<>
!<>================  local variables
!<>
!<>  <fitting-doc-tag-begin> <unsupported-local-block>
!<>
integer( kind = 4 ), parameter :: SPEC_done_Jac = 2
integer( kind = 4 ), parameter :: SPEC_done_Rone = 3
integer( kind = 4 ), parameter :: SPEC_done_dir = 4
integer( kind = 4 ), parameter :: SPEC_done_fwd = 3
integer( kind = 4 ), parameter :: SPEC_done_init = 2
integer( kind = 4 ), parameter :: SPEC_done_none = 1
integer( kind = 4 ), parameter :: SPEC_done_opt = 5
integer( kind = 4 ), parameter :: SPEC_done_rev = 4
integer( kind = 4 ), parameter :: SPEC_need_Jac = 1
integer( kind = 4 ), parameter :: SPEC_need_Rone = 2
integer( kind = 4 ), parameter :: SPEC_need_dir = 3
integer( kind = 4 ), parameter :: SPEC_need_fwd = 2
integer( kind = 4 ), parameter :: SPEC_need_init = 1
integer( kind = 4 ), parameter :: SPEC_need_opt = 4
integer( kind = 4 ), parameter :: SPEC_need_rev = 3
integer( kind = 4 ) :: i_GM
real( kind = 8 ) :: J_build
real( kind = 8 ) :: f_frac
real( kind = 8 ) :: f_pow_frac
real( kind = 8 ) :: f_float_accur
!<>
!<>  <fitting-doc-tag-end> <unsupported-local-block>
!<>
!<>================================  body (Jac_double_REVX)
!<>
!<>  <fitting-doc-tag-begin> <unsupported-body-block>
!<>
if ( n_eval == 0 ) then
spec_done = SPEC_done_none
spec_need = SPEC_need_init
v_XM = 1
end if
if ( n_eval == 0 ) then
```

```
X_step = f_pow_frac( f_float_accur( ), 1, 3 ) * X_scale
end if
if ( spec_done == SPEC_done_init ) then
call float_array_copy( ord_M, G_Jac_GM, G_eval_GM )
spec_need = SPEC_need_fwd
end if
if ( spec_done == SPEC_done_fwd ) then
call float_array_copy( ord_M, G_fwd_GM, G_eval_GM )
spec_need = SPEC_need_rev
end if
if ( spec_done == SPEC_done_rev ) then
call float_array_copy( ord_M, G_rev_GM, G_eval_GM )
spec_need = SPEC_need_fwd
end if
if ( spec_done == SPEC_done_rev ) then
do i_GM = 1, ord_M
J_build = f_frac(1,2) *(G_fwd_GM(i_GM) -G_rev_GM(i_GM))/X_step
J_calc_GM_XM( i_GM, v_XM ) = J_build
end do
end if
if ( spec_done == SPEC_done_rev ) then
v_XM = v_XM + 1
end if
continue_eval = 1
if ( .NOT. ( v_XM <= ord_M ) ) continue_eval = 0
if ( continue_eval /= 0 ) then
if ( spec_need == SPEC_need_init ) then
call float_array_copy( ord_M, X_eval_XM, X_Jac_XM )
spec_done = SPEC_done_init
end if
if ( spec_need == SPEC_need_fwd ) then
call float_array_copy( ord_M, X_eval_XM, X_Jac_XM )
X_eval_XM( v_XM ) = X_eval_XM( v_XM ) + X_step
spec_done = SPEC_done_fwd
end if
if ( spec_need == SPEC_need_rev ) then
call float_array_copy( ord_M, X_eval_XM, X_Jac_XM )
X_eval_XM( v_XM ) = X_eval_XM( v_XM ) - X_step
spec_done = SPEC_done_rev
end if
end if
!<>
!<>  <fitting-doc-tag-end> <unsupported-body-block>
!<>
!<>================================  Jac_double_REVX
end subroutine
!<>
!<>
!<>
!<>
!<>================================  opt_Nash_REVX
!<>
!<>  <fitting-doc-tag-short> <need-sub-description>
!<>
!<>  <fitting-doc-tag-short> <unsupported-sub-declare>
subroutine opt_Nash_REVX( X_eval, G_eval, X_best, G_best, n_eval,&
            continue_eval,err_opt, X_init, X_change, X_scale, G_scale,&
            X_toler, G_toler )
!<>
!<>  <fitting-doc-tag-begin> <supported-use-block>
!<>
use global_module
!<>
!<>  <fitting-doc-tag-end> <supported-use-block>
!<>
!<>================================  declarations
!<>
!<>================  arguments
```

```
!<>
!<>--------  input
!<>
!<>  <fitting-doc-tag-begin> <unsupported-arg-block>
!<>
real( kind = 8 ) :: G_eval
integer( kind = 4 ) :: n_eval
real( kind = 8 ) :: X_init
real( kind = 8 ) :: X_change
real( kind = 8 ) :: X_scale
real( kind = 8 ) :: G_scale
real( kind = 8 ) :: X_toler
real( kind = 8 ) :: G_toler
!<>
!<>  <fitting-doc-tag-end> <unsupported-arg-block>
!<>
!<>--------  output
!<>
!<>  <fitting-doc-tag-begin> <unsupported-arg-block>
!<>
real( kind = 8 ) :: X_eval
real( kind = 8 ) :: X_best
real( kind = 8 ) :: G_best
integer( kind = 4 ) :: continue_eval
character( len = * ) :: err_opt
!<>
!<>  <fitting-doc-tag-end> <unsupported-arg-block>
!<>
!<>================  static local variables
!<>
!<>  <fitting-doc-tag-begin> <unsupported-local-block>
!<>
integer( kind = 4 ) :: n_expand ; save n_expand
real( kind = 8 ) :: X_step ; save X_step
!<>
!<>  <fitting-doc-tag-end> <unsupported-local-block>
!<>
!<>================  local variables
!<>
!<>  <fitting-doc-tag-begin> <unsupported-local-block>
!<>
real( kind = 8 ), parameter :: tune_Nash_contract_factor = 0.25_8
real( kind = 8 ), parameter :: tune_Nash_expand_factor = 0.60_8
integer( kind = 4 ), parameter :: tune_Nash_max_eval = 9
integer( kind = 4 ), parameter :: tune_Nash_max_expand = 5
integer( kind = 4 ) :: flag_bootstrap
integer( kind = 4 ) :: flag_success
integer( kind = 4 ) :: flag_expand
real( kind = 8 ) :: X_residual
real( kind = 8 ) :: expand_factor, contract_factor
integer( kind = 4 ) :: max_expand
integer( kind = 4 ) :: max_eval
integer( kind = 4 ) :: n_expand_safe
real( kind = 8 ) :: X_step_safe
real( kind = 8 ) :: f_abs
!<>
!<>  <fitting-doc-tag-end> <unsupported-local-block>
!<>
!<>================================  body (opt_Nash_REVX)
!<>
!<>  <fitting-doc-tag-begin> <unsupported-body-block>
!<>
expand_factor = tune_Nash_expand_factor
contract_factor = tune_Nash_contract_factor
max_expand = tune_Nash_max_expand
max_eval = tune_Nash_max_eval
```

```
flag_bootstrap = 0
if ( n_eval == 0 ) then
flag_bootstrap = 1
X_best = X_init
X_step = global__float_zero
flag_expand = 0
end if
if ( n_eval == 1 ) then
flag_bootstrap = 1
X_best = X_eval
G_best = G_eval
X_step = X_change
flag_expand = 1
end if
if ( .NOT. flag_bootstrap /= 0 ) then
flag_success = 0
if ( G_eval < G_best ) then
flag_success = 1
end if
X_step_safe = X_step
X_step = - contract_factor * X_step_safe
flag_expand = 0
if ( flag_success /= 0 ) then
X_best = X_eval
G_best = G_eval
X_step = + expand_factor * X_step_safe
flag_expand = 1
end if
end if
n_expand_safe = n_expand
n_expand = n_expand_safe + 1
if ( .NOT. flag_expand /= 0 ) then
n_expand = 0
end if
continue_eval = 1
if ( n_eval > 0 ) then
X_residual = f_abs( X_step ) / X_scale
if ( X_residual < X_toler ) continue_eval = 0
end if
if ( continue_eval /= 0 ) then
if ( n_expand > max_expand ) then
continue_eval = 0
return
end if
if ( n_eval + 1 > max_eval ) then
continue_eval = 0
return
end if
end if
X_eval = X_best + X_step
!<>
!<>  <fitting-doc-tag-end> <unsupported-body-block>
!<>
!<>================================  opt_Nash_REVX
end subroutine
!<>
!<>
!<>
!<>
```

APPENDIX D

FINAL RESIDUALS

The final Gallium bulk properties used and the residual for each are given in table D.1.

Property	fitted value	reference value	weight	residual
Ga-α Bulk	0.00108891	0.00150928	0.002	-211.86053983
Ga-α Bulk	0.00048555	0.00067317	0.002	-94.55887350
Ga-α Bulk	0.00013033	0.00018363	0.002	-26.86049887
Ga-α Bulk	0.00000000	0.00000000	0.002	0.00000000
Ga-α Bulk	0.00007054	0.00008642	0.002	-8.00164586
Ga-α Bulk	0.00032298	0.00041180	0.002	-44.76338915
Ga-α Bulk	0.00074298	0.00094515	0.002	-101.88899631
Ga-α Bulk	0.00131307	0.00165605	0.002	-172.85675153
Ga-β Bulk	0.00096709	0.00109701	0.002	-65.47820889
Ga-β Bulk	0.00041535	0.00042990	0.002	-7.33176424
Ga-β Bulk	0.00009297	0.00007599	0.002	8.55923426
Ga-β Bulk	0.00000000	0.00000000	0.002	0.00000000
Ga-β Bulk	0.00013604	0.00016614	0.002	-15.17001649
Ga-β Bulk	0.00049068	0.00054712	0.002	-28.44370394
Ga-β Bulk	0.00102741	0.00111581	0.002	-44.55335841
Ga-β Bulk	0.00174131	0.00185373	0.002	-56.65945091

Table D.1: Final bulk property residuals.

The final Gallium cluster properties used and the residual for each are given in table D.2 and table D.3.

Property	fitted value	reference value	weight	residual
Ga_3 D_{3h} r	1.43487808	1.482283	1.3	-36.75625283
Ga_3 D_{3h} Energy	-0.03657752	-0.036063	0.045	-11.52502081
Ga_5 D_{5h} r	2.25589641	2.280500	2.6	-9.53841468
Ga_5 D_{5h} Energy	-0.04536755	-0.04010300	0.045	-117.92332174
Ga_5 D_{4h} r	2.52463120	2.517010	2.6	2.95461579
Ga_5 D_{4h} Energy	-0.04565275	-0.043727	0.045	-43.13577432

Table D.2: Final Gallium cluster property residuals.

Property	fitted value	reference value	weight	residual
Ga_5 C_{2v} r_1	1.21486480	1.263048	1.3	-37.35970732
Ga_5 C_{2v} r_2	1.22085794	1.283961	1.3	-48.92808619
Ga_5 C_{2v} z_1	2.37134197	2.466808	2.6	-37.01064586
Ga_5 C_{2v} z_2	2.35985813	2.533019	2.6	-67.13168551
Ga_5 C_{2v} Energy	-0.05173958	-0.049683	0.045	-46.06648618
Ga_6 C_{2v} a z	2.40543072	2.615500	2.6	-81.44048449
Ga_6 C_{2v} a r_1	1.99663562	1.994878	2.0	0.88582179
Ga_6 C_{2v} a r_2	1.45284734	1.636769	1.3	-142.60694404
Ga_6 C_{2v} a Energy	-0.05695845	-0.055101	0.045	-20.75213072
Ga_6 D_{3h} z	2.19784791	2.600000	2.6	-155.90790319
Ga_6 D_{3h} r	1.53888498	1.570000	1.3	-24.12558836
Ga_6 D_{3h} Energy	-0.05795073	-0.055902	0.045	-45.89061844
Ga_6 C_{2v} b r_1	1.76406275	1.911780	2.0	-74.44788619
Ga_6 C_{2v} b r_2	1.70684369	1.721574	1.3	-11.42141083
Ga_6 C_{2v} b r_3	1.30134698	1.371566	1.3	-54.44557542
Ga_6 C_{2v} b z_1	0.13990359	0.239604	2.0	-50.24792034
Ga_6 C_{2v} b z_2	2.49130495	2.577068	2.6	-33.24895483
Ga_6 C_{2v} b Energy	-0.05558643	-0.055101	0.045	-10.87337068
Ga_6 D_{3d} r	1.62185035	1.765801	1.3	-111.61470727
Ga_6 D_{3d} z	1.86542318	1.894614	2.0	-14.71185600
Ga_6 D_{3d} Energy	-0.05520248	-0.054886	0.045	-7.08907091
Ga_6 D_{2d} r_1	1.20431976	1.230612	1.3	-20.38615695
Ga_6 D_{2d} r_2	2.28962801	2.265730	2.0	12.04433889
Ga_6 D_{2d} z	1.05579194	1.243194	1.3	-145.30553645
Ga_6 D_{2d} Energy	-0.04955038	-0.045464	0.045	-91.53300052
Ga_7 C_{3v} r_1	1.69920581	1.767103	1.3	-52.64530106
Ga_7 C_{3v} r_2	1.59802688	1.617807	1.3	-15.33687261
Ga_7 C_{3v} z_1	2.03090797	2.078788	2.0	-24.13101157
Ga_7 C_{3v} z_2	1.93661231	1.993268	2.0	-28.55385310
Ga_7 C_{3v} Energy	-0.05749826	-0.059215	0.045	38.45413787
Ga_7 C_s r_1	0.23181712	0.265887	2.0	-17.17084934
Ga_7 C_s r_2	1.43264180	1.681425	1.3	-192.89849695
Ga_7 C_s r_3	2.32856692	2.513624	2.6	-71.74365818
Ga_7 C_s r_4	1.33383421	1.313252	1.3	15.95878367
Ga_7 C_s r_5	2.17027601	2.341020	2.6	-66.19470343
Ga_7 C_s r_6	0.68161157	0.694996	2.6	-5.18892837
Ga_7 C_s z_1	2.10267701	1.942529	2.6	62.08681071
Ga_7 C_s z_2	2.22966546	2.371415	2.6	-54.95401806
Ga_7 C_s z_3	2.20791812	1.998043	2.6	81.36521323
Ga_7 C_s Energy	-0.05886136	-0.060602	0.045	38.98938985
Ga_8 D_{2h} r_1	1.42813867	1.405199	2.0	11.56134495
Ga_8 D_{2h} r_2	2.16986894	2.371196	2.0	-101.46664559
Ga_8 D_{2h} z_1	1.21837079	1.267377	1.3	-37.99784191
Ga_8 D_{2h} z_2	1.19556857	1.279033	1.3	-64.71563565
Ga_8 D_{2h} Energy	-0.06253196	-0.062638	0.045	2.37528981

Table D.3: Final Gallium cluster property residuals.

The final Gallium Nitride cluster properties used and the residual for each are given in table D.4 and table D.5.

Property	fitted value	reference value	weight	residual
Ga_1N_3 C_{2v} r_1	1.96107874	1.896811	1.4	45.93400039
Ga_1N_3 C_{2v} r_2	1.29373921	1.397985	1.4	-74.50746174
Ga_1N_3 C_{2v} z	0.62911387	0.594313	1.4	24.87317960
Ga_1N_3 C_{2v} Energy	-0.09068114	-0.10365850	0.10	129.85405272
Ga_1N_3 $C_{\infty h}$ r_1	2.01779008	1.912230	1.4	75.44682276
Ga_1N_3 $C_{\infty h}$ r_2	1.22065495	1.201475	1.4	13.70845980
Ga_1N_3 $C_{\infty h}$ r_3	1.16599979	1.136267	1.4	21.25087925
Ga_1N_3 $C_{\infty h}$ Energy	-0.12997784	-0.13104750	0.10	10.70326728
Ga_1N_3 Pyr. r_1	1.03573571	1.190017	1.4	-110.26927042
Ga_1N_3 Pyr. r_2	1.30329321	1.448884	1.4	-104.05792083
Ga_1N_3 Pyr. r_3	0.62478750	0.589452	1.4	25.25529321
Ga_1N_3 Pyr. z	1.81052627	1.558995	1.4	179.77662549
Ga_1N_3 Pyr. Energy	-0.09621957	-0.10415450	0.10	79.39855630
Ga_3N_1 $C_{\infty h}$ r_1	2.46234324	2.704187	1.4	-172.85268088
Ga_3N_1 $C_{\infty h}$ r_2	1.70791602	1.725278	1.4	-12.40910433
Ga_3N_1 $C_{\infty h}$ r_3	1.90968009	1.845941	1.4	45.55615626
Ga_3N_1 $C_{\infty h}$ Energy	-0.06486886	-0.07495350	0.10	100.90891815
Ga_3N_1 D_{3h} r	1.91206761	1.915125	1.4	-2.18520640
Ga_3N_1 D_{3h} Energy	-0.06997367	-0.09120750	0.10	212.47004297
Ga_1N_4 $C_{\infty h}$ r_1	1.81798028	1.767594	1.4	36.01252052
Ga_1N_4 $C_{\infty h}$ r_2	1.86623543	1.782539	1.4	59.82023922
Ga_1N_4 $C_{\infty h}$ r_3	1.24076908	1.197856	1.4	30.67120839
Ga_1N_4 $C_{\infty h}$ r_4	1.16775707	1.130722	1.4	26.47003018
Ga_1N_4 $C_{\infty h}$ Energy	-0.13245292	-0.10910380	0.10	-233.63597668
Ga_2N_3 $C_{\infty h}$ r_1	2.48401855	2.744226	1.4	-185.97774100
Ga_2N_3 $C_{\infty h}$ r_2	1.90510833	1.872233	1.4	23.49694254
Ga_2N_3 $C_{\infty h}$ r_3	1.24739455	1.198579	1.4	34.88987320
Ga_2N_3 $C_{\infty h}$ r_4	1.18046492	1.136618	1.4	31.33865172
Ga_2N_3 $C_{\infty h}$ Energy	-0.11399023	-0.10788460	0.10	-61.09417268
Ga_2N_3 $D_{\infty h}$ r_1	1.26334680	1.173079	1.4	64.51699280
Ga_2N_3 $D_{\infty h}$ r_2	1.97744385	2.020915	1.4	-31.07007726
Ga_2N_3 $D_{\infty h}$ Energy	-0.10199629	-0.10853960	0.10	65.47364033

Table D.4: Final Gallium Nitride cluster property residuals.

Property	fitted value	reference value	weight	residual
Ga_3N_2 $D_{\infty h}$ r_1	1.74979945	1.740596	1.4	6.57797034
Ga_3N_2 $D_{\infty h}$ r_2	1.78835803	1.880500	1.4	-65.85651044
Ga_3N_2 $D_{\infty h}$ Energy	-0.07944046	-0.08630740	0.10	68.71199387
Ga_4N_1 C_{2v} r_1	3.39992022	3.700572	1.4	-214.88445997
Ga_4N_1 C_{2v} r_2	1.27831495	1.209966	1.4	48.85095619
Ga_4N_1 C_{2v} r_3	1.91893693	1.902282	1.4	11.90375879
Ga_4N_1 C_{2v} z	1.43383416	1.481297	1.4	-33.92305214
Ga_4N_1 C_{2v} Energy	-0.07004712	-0.07983020	0.10	97.89149375
Ga_4N_1 $C_{\infty h}$ r_1	2.51709146	2.686556	1.4	-121.12117373
Ga_4N_1 $C_{\infty h}$ r_2	2.32957791	2.550218	1.4	-157.69780638
Ga_4N_1 $C_{\infty h}$ r_3	1.71133512	1.719684	1.4	-5.96718739
Ga_4N_1 $C_{\infty h}$ r_4	1.90520958	1.843453	1.4	44.13919863
Ga_4N_1 $C_{\infty h}$ Energy	-0.05932974	-0.06696020	0.10	76.35188914
Ga_6N_6 D_{3d} r_G	1.83431304	1.748908	1.4	61.04143633
Ga_6N_6 D_{3d} r_N	2.07768866	2.025861	1.4	37.04272037
Ga_6N_6 D_{3d} z	2.12729860	2.026850	1.4	71.79349594
Ga_6N_6 D_{3d} Energy	-0.10981760	-0.10289000	0.10	-69.31900864

Table D.5: Final Gallium Nitride cluster property residuals.

CURRICULUM VITÆ

Lyle C. Smith, III

Education

2015	**University of Louisville** Ph.D. in Applied and Industrial Mathematics
2005	**University of Louisville** M.S. in Physics
2005	**The Southern Baptist Theological Seminary, Louisville, KY** M.Div. in Christian Ministry
1997	**Virginia Tech, Blacksburg, VA** M.S. in Mathematics
1995	**Virginia Tech, Blacksburg, VA** B.S. in Physics

Publications

"Cogging Torque Modeling and Analysis," L. Smith, C.P. Cho, Incremental Motion Control Systems & Devices Symp.

"FEA Analysis of Various Motor Configurations," L. Smith, R. McConnell, C.P. Cho, Naval Symp. on Electric Machines.

"Modeling and Simulation of a Novel Integrated Electric Motor/Propulsor," Dr. C. Peter Cho, William Krol Jr., and Lyle Smith III, Incremental Motion Control Systems and Devices Symposium.

"A Novel Integrated Electric Motor/Propulsor for Underwater Propulsion," John Raposa, Lyle Smith III, William Fennell, William Krol Jr., James Uhlman, Daniel Thivierge, and Dr. C. Peter Cho, Naval Symposium on Electric Machines.

"Application of a High-Energy Density Permanent Magnet Material in Underwater Propulsion Systems," Dr. C. Peter Cho, Andrew Kim, and Lyle Smith III, Fifteenth International

Workshop on Rare-Earth Magnets.

"Techniques for Modeling Motors in ANSYS," Dr. Mike Yaksh, EMAG Specialist and Lyle Smith III, Naval Undersea Warfare Center, ANSYS Users' Conference and Exhibition.

"Motor Macros for Operation of ANSYS Electromagnetic Models," Lyle Smith III, NUWC Internal Technical Memo.