Optimization with Surrogates for Electronic-Structure Calculations

by

Yonas Beyene Abraham

A Thesis Submitted to the Graduate Faculty of
WAKE FOREST UNIVERSITY
in Partial Fulfillment of the Requirements
for the Degree of
Masters of Science
in the Department of Computer Science
May 2004
Winston-Salem, North Carolina

Approved by:
Dr. Robert Plemmons, Advisor _____
Examining Committee:
Dr. Paul Pauca, Chairman _____
Dr. Marielba Rojas _____
Dr. N. A. W. Holzwarth, _____

Contents

List of Figures

List of Tables

Acknowledgments

I would like to gratefully acknowledge the enthusiastic supervision of Dr. Marielba Rojas and Dr. Robert Plemmons during this work. I thank Dr. Natalie Holzwarth for coming up with the original problem and Dr Marielba Rojas for her inspiring idea of surrogate modeling. My gratitude goes to all my committee members for their constructive comments and corrections to my thesis.

I am grateful to the department of physics for allowing me to do a masters degree in Computer Science.

Finally, I am forever indebted to my family for their endless help.

Abstract

Crystal geometry relaxation is needed to get a stable crystal structure of atoms. The relaxation is achieved by rearranging the position of atoms so as to produce the lowest total energy of the system. The high cost of function evaluations in relaxation calculations usually prevents the use of standard techniques to solve the associated optimization problems. Therefore, more sophisticated approaches are necessary. This work explores the use of surrogate optimization methods for these problems and presents results for simple total-energy functionals in one-,two-, and ten-dimensions.

Chapter 1

Introduction

Performing a geometry optimization is often the first step in the computational study of a material. Geometry optimization typically attempts to locate a minimum of the potential energy surface in order to predict equilibrium structures of materials and it may also be used to locate transition structures.

Nearly all physical properties of a material are related to total energies or to differences between total energies [22]. For example, the equilibrium lattice constant of a crystal is the lattice constant that minimizes the total energy; and surfaces and defects of solids adopt the structures that minimize their corresponding total energies. Total-energy techniques have also been successfully used to predict with accuracy physical quantities such as equilibrium lattice constants, bulk moduli, phonons, piezoelectric constants, and phase-transition pressures and temperature [6]. Mechanical failure of a material starts at the atomic scale when one bond is stressed beyond its yield stress and starts to break. Thus, understanding the basic property of a material starts by finding a stable configuration of atoms that gives the minimum total-energy function.

One of the applications in which we want to perform total-energy calculations is the Plane Wave Projector Augmented Wave (PWPAW) [13, 23] method. The PWPAW code is a plane wave implementation of the Projector Augmented Wave (PAW) method developed by Blöchl [2] for electronic structure calculations within the framework of density functional theory. In addition to the self-consistent calculation of the electronic structure of a periodic solid, the program has a number of other capabilities, including structural geometry optimization and molecular dynamics simulations within the Born-Oppenheimer approximation [13].

Density functional theory developed by Hohenberg, Kohn, and Sham [12, 15] gives a relatively simple prescription for the total energy of a system of electrons and nuclei. It offers a powerful and elegant method for calculating the ground-state total energy and electron density of a system of interacting electrons. The system may range in complexity from a single atom to a complex system, such as gas molecules together with the atoms of the solid surface on which they are about to be adsorbed and where they will react with one another, guided by the total energy. The theory is based on n *onal* of the electron density. The key functional is the one that describes the total energy of the electrons as a functional of their density.

The Total-Energy Functional looks like the following [15].

$$\begin{aligned} E[\{\psi(\vec{x})\}] &= \sum_i f_i \int \psi_i^*(\vec{x}) \left(-\frac{\hbar^2}{2m} \nabla^2 \psi_i(\vec{x}) \right) dV \\ &+ \int V_{nuc}(\vec{x}) n(\vec{x}) dV \\ &+ \frac{1}{2} \int \phi(\vec{x}) n(\vec{x}) dV \\ &+ \int f_{xc}(n(\vec{x})) dV \\ &+ U_{nn} \end{aligned} \tag{1.1}$$

for $i = 1, 2, ...,$ number of orbital, and where:

- f_i (usually equal to two) is the number of electrons in orbital .
- $\psi_i(\vec{x})$ are called the Kohn-Sham orbitals. An orbital often is depicted as a three-dimensional region within which there is a high probability of finding the electron [21]. Figure 1.1 illustrates and p orbitals. These orbitals also describe the behavior of the electrons.
- $\psi_i^*(\vec{x})$ are the complex conjugates of $\psi_i(\vec{x})$.
- $\hbar$ is the Plank's constant.
- m is the mass of an electron.
- dV is a volume element.

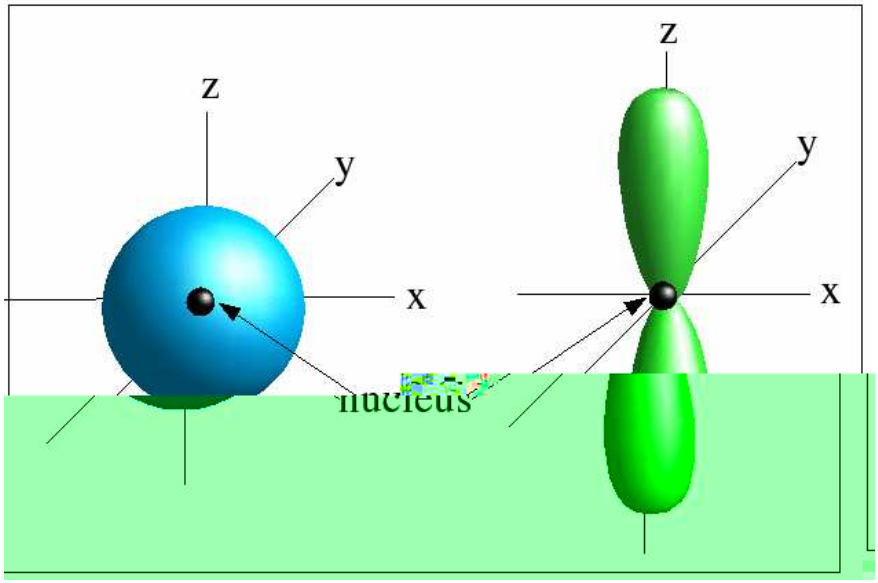

Figure 1.1: Electron orbitals in atoms (left) orbital; (right) p orbital

- $V_{nuc}(\vec{x}) = -Ke^2 \sum_i \frac{Z_i}{R_i}$ is the single electron potential at point $\vec{x}$, where K is the Coulomb's constant, R_i is the distance from point $\vec{x}$ to nucleus i. e is the charge of an electron, and Z_i is the atomic number of nucleus ,
- $U_{nn} = \frac{1}{2}Ke^2 \sum_{i \neq j} \frac{Z_i Z_j}{R_{ij}}$ is the total potential energy arising from the interactions among all the nuclei. Where R_{ij} is the separation between nuclei and .
- $n(\vec{x}) = \sum_i f_i |\psi_i(\vec{x})|^2$is the volume density (number per unit volume) of electrons.
- $\phi_i(\vec{x}) = Ke^2 \int \frac{n(\vec{x}\prime)dV}{|\vec{x}-\vec{x}\prime|}$ is the total potential for a single electron at position $\vec{x}$.
- f_{xc} is the exchange and correlation function.

It is important to note that not all parts of the right hand side of (1.1) are known. In particular, the eXchange-Correlation (XC) part (which is the $f_{xc}(n(\vec{x}))$ term) must be approximated. This function depends on what approximation method we are using.

Note that expression (1.1) maps each possible choice of the set of electronic orbitals $\{\psi(\vec{x})\}$ to a unique value of the energy of the system, and therefore gives the total energy as a function of the orbital functions $\psi_i(\vec{x})$.

Within quantum mechanics, the

One way of computing the minimum value of the total energy for a given atomic configuration is solving (1.5) by means of the Self-Consistent-Field method (**SCF**) [15]. The sequence of steps required to carry out the SCF calculation is shown in Figure 1.2. This method requires an initial guess for the electron-density function $n(\vec{x})$, from which the Hartree potential (V_H) and the eXchange Correlation potential $V_{XC}(\vec{x})$ can be calculated. Note that (1.5) specifies a system of equations, whose coefficient matrix can be diagonalized to obtain the Kohn-Sham eigenstats in (1.5). This eigenstates are then used to generate a new charge density. If this new charge density is not the same (consistent) as the starting charge density within a tolerance value, then a new set of Hartree and eXchange correlation functions must be constructed using the new charge density. The process is then repeated until the solutions are consistent.

Up to this point, we have a procedure, i.e. the SCF method, for computing the value of the total energy at a given geometry of atoms. The question is now, what is the correct geometry? This is the question we try to answer in this project.

Two approaches are known to compute the atomic configuration or geometry. They are called direct and indirect minimization of the Kohn-Sham energy functional (1.1). The direct minimization technique [11] is based on the Conjugate Gradient Method (CG) [20]. In this method, the position of the nuclei changes at each iteration and the value of the energy functional is required for each new geometry. The evaluation of the energy functional requires a complete cycle of the SCF method, which can be very expensive. For example, certain complex systems might require weeks of SCF computations before the desired energy value is found. If CG requires many function evaluations, as it often does for these problems, this approach becomes very slow and time consuming.

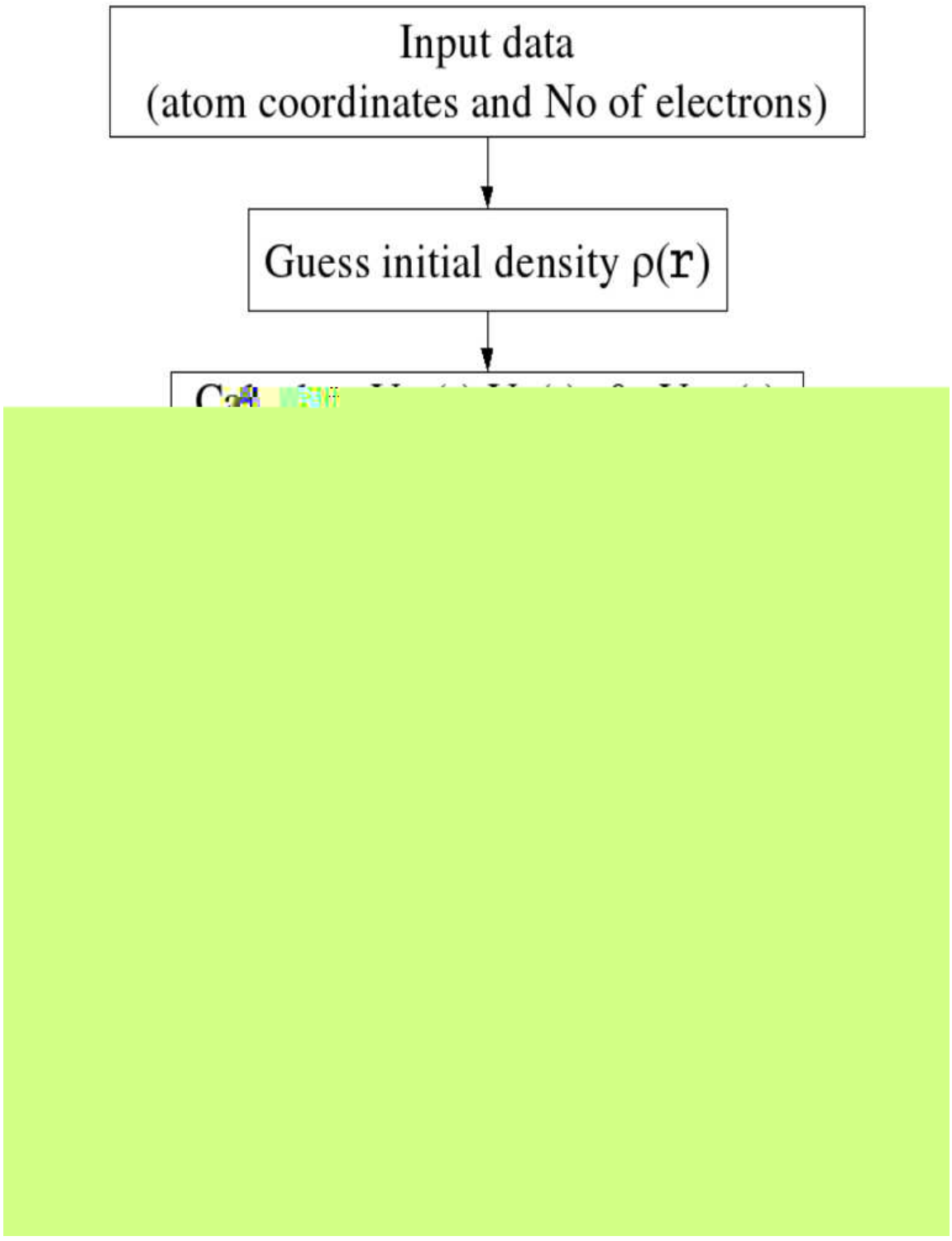

Figure 1.2: Flow chart describing the computational procedure for the calculation of total-energy in the SCF cycle.

The second approach, called indirect minimization, combines both the SCF calculations and geometry minimization in one equation. This is the basis for the Molecular-Dynamics (MD) approach also known as the Car-Parrinello method [4]. The total energy of the system, which is the electronic energy E_{el} plus the electrostatic repulsion energy of the nuclei, U_{rep}, can be written as a functional depending on the orbitalsψ_k and of the nuclear coordinates R_k:

$$E_{total} = E_{el}(\{\psi_k\}, \{R_k\}) + U_{rep}(\{R_k\}) = E_{tot}(\{\psi_k\}, \{R_k\})$$

where the orbitals ψ_k form an orthonormal set:

$$< \psi_i | \psi_j >= \delta_{ij}. \tag{1.6}$$

Usually, a finite basis set $\{\chi_i\}$ is used, in terms of which the orbitals are given as

$$\psi_k = \sum_i C_i^k \chi_i(r)$$

so that the energy can be written in terms of the C_i^k and $\{R_k\}$,

$$E_{total} = E_{total}(\{C_i^k\}, \{R_k\}). \tag{1.7}$$

Car and Parrinello used the form (1.7) with the constraint (1.6) as a starting point for finding the minimal energy configuration by locating the minimum of the total energy as a function of C_i^k and the nuclear coordinates $\{R_k\ \}$. This means that the electronic structure does not have to be calculated exactly for each atomic configuration, as both the electronic orbitals and the nuclear positions are varied simultaneously in order to locate the minimum. The energy minimization problem can now be considered as an abstract numerical problem, and any minimization algorithm can in principle be applied.

Car and Parrinello chose a molecular dynamics method as the minimization algorithm. Aside from the time dependence of the nuclear coordinates, they assigned a fictitious time-dependence to the electronic wave functions . To put in other words, each coefficient C_i^k can be regarded as the coordinate of a classical particle. Then they constructed a dynamical Lagrangian which includes both the electronic wave functions and nuclear coordinates with their time derivatives as the variables. This leads to a classical mechanics problem with the energy (1.7) acting as a potential.

The Lagrangian of the classical system reads:

$$L(\{\psi_k\},\{R_k\}) = \frac{\mu}{2}\sum \dot{\psi}_k^2 + \sum_n \frac{M_n}{2}\dot{R}_n^2 - E_{Total}(\{\psi_k\},\{R_k\}) + \sum_{kl}\Lambda_{kl}[<\psi_k|\psi_l> -\delta_{kl}], \tag{1.8}$$

where μ is a fictitious mass (quite small) associated with the electronic orbital and M_n is the actual mass of the $n-th$ nucleus with position R_n. The last term on the right hand side is necessary to ensure orthonormality of ψ_k. The Lagrange multipliers Λ_{kl} must always be calculated from this requirement.

The details of the kinetic energy of the electrons do not matter: what matters is the fact that the mass μ should be sufficiently small to enable the electronic wave function to adapt reasonably to changes in nuclear configurations. Thus, μ should be much smaller than the nuclear masses.

The equations of motion for the electronic states are derived from the Lagrange equations of motion,

$$\frac{d}{dt}\left[\frac{\partial L}{\partial \dot{\psi}_k}\right] - \frac{\partial L}{\partial \psi_k} = 0,$$

which give

$$\mu\ddot{\psi}_k = -\frac{\partial E_{total}}{\partial \psi_k} + \sum_l \Lambda_{kl}\psi_l(r) = -H\psi_k + \sum_l \Lambda_{kl}\psi_l(r). \tag{1.9}$$

where H is the single particle Hamiltonian of the system (e.g. Kohn-Sham Hamiltonian).

The equations of motion for the nuclei is:

$$M_n\ddot{R}_n = -\frac{\partial E_{total}}{\partial R_n}.$$

The values of the Lagrange parameters Λ_{kl} depend on time - they must be calculated at each molecular dynamics step such that they guarantee the orthonormality constraint (1.6).

The constrained molecular-dynamics equations of motion for the electronic states (1.9) ensure that the electronic wave functions remain orthonormal at every instant in time. The molecular-dynamics evolution of the electronic wave functions under these equations of motion would also conserve the total energy in the electronic degrees of freedom for the

system of fixed ions. However, to ensure these properties, the values of the Lagrange multipliers must vary continuously with time, and so implementation of this form of the molecular dynamics equations requires that the Lagrange multipliers be evaluated at infinitely small time separations. To make the calculations tractable, variation of the Lagrange multipliers during a time step is neglected and the Lagrange multipliers are approximated by a constant value during the time step. In this case the wave functions will not be exactly orthonormal at the end of the time step, and a separate orthonormalization step is needed in the calculation.

Once the acceleration ,$\ddot{\psi}_k$, of the coefficients have been calculated, the equation of motion has to be integrated. Car and Parrinello used the Verlet algorithm [24] to integrate the equations as follows.

$$\psi_i(t+\Delta t) \quad = \quad 2\psi_i(t) - \psi_i(t-\Delta t) - \Delta t^2 \ddot{\psi}_i(t).$$

Where Δt is the length of time step, $\psi_i(t)$ is the value of the state at the present time step, and $\psi_i(t-\Delta t)$ is the value of the state at the last time step.

The main advantage of the Car-Parrinello method is that it does not need to calculate the total energy functional explicitly for each change in geometry. However, the method becomes inefficient as the size of the system increases. For systems with large supercell length scales, the maximum stable time step is completely dominated by the need to suppress fluctuations in the charge density. These fluctuations in charge lead to instabilities in the solutions of (1.1). Most m al are very sensitive to these density fluctuations, and as a result, the MD method is rarely used on metals.

Note that it is always possible to ensure stable evolution of the electronic configuration using the MD method, but this requires smaller time steps and hence more computational time as the size of the system increases.

In this work, we propose a new approach to try to overcome the shortcomings of existing methods. We explore the use of state-of-the-art optimization techniques known as $\mathit{rro}\ a$ $\mathit{m o}$ $l\ n$ and pa $\mathit{m app}\ n$ for computing the minimum of the total-energy functional (1.1). In optimization with surrogates (see for example [3]), we construct a model (the $\mathit{rro}\ a$) of the objective function that is easy to compute and inexpensive to evaluate, and that preserves some of the features of the original function. The idea is then to minimize the surrogate instead of the true objective function. Since the surrogate

can be evaluated at low cost, classical optimization techniques can be used to solve the surrogate problem. The space mapping component in our case, consists of evaluating the true energy functional at the minimizer of the surrogate model, and of using this point along with previous data to construct a new surrogate. More details on the space mapping technique and its use with surrogate modeling can be found, for example, in [1].

Our method, consists of an iteration where at each step we construct a model of the total-energy functional, minimize that model, and evaluate the total-energy functional at the minimizer of the model. If the minimizer of the surrogate yields an "acceptable" value for the total-energy functional, then we stop the iteration. If not, the new point is added to the existing data and the iteration continues.

The proposed approach has the following advantages:

1. It requires few SCF calculations.

2. It allows the use of classical and efficient optimization methods to minimize the surrogate function.

3. It makes use of all available information, since all the SCF values previously computed are used to compute the surrogate function.

This report is organized as follows. In Chapter 2, we describe our problem in more detail. In Chapter 3, we describe our optimization approach, including the choice of surrogate models and optimization methods. Experimental results on test problems are shown in Chapter 4. Chapter 5 contains some concluding remarks and future work.

Chapter 2

Problem

The problem at hand is

$$\begin{aligned} \min \quad & E(r_1, r_2, ..., r_n) \\ \text{subject to} \quad & r_1, r_2, ..., r_n \quad > 0 \end{aligned} \tag{2.1}$$

where E is the total energy functional (1.1) and $r_1, r_2, ..., r_n$ are the n independent atomic positions of atoms in a given crystal structure that we want to optimize. These are generally the x, y, z coordinates of the atoms in a crystal relative to some origin. As it is mentioned in the introduction, equation (2.1) generally does not have an analytical expression that explicitly depends on $r_1, r_2, ..., r_n$. That is, by simply plugging in values of $r_1, r_2, ..., r_n$ we can not get the correct value of E. The total energy curve is assumed to have a strictly convex shape, that is it has only one minimum value throughout the range of our position coordinates.

Each atom has three degrees of freedom. So if we have k atoms to relax, then we will have $n = 3k$. For example, if we have four atoms in the crystal that we want to relax, then we will have 12 parameters. In reality, the number of parameters is less than $3k$. This is because most crystals have some symmetry that can reduce this number. Moreover, one could also use a generalized coordinate system to reduce the number of parameters.

To solve (2.1), we use an interpolation method (please refer to Chapter 3 for detailed explanation). The interpolation method needs to know the value of at m different values of $r_1, r_2, ..., r_n$. These different values of $r_1, r_2, ..., r_n$ are called the design sites and the value of (2.1) evaluated at these design sites are called responses throughout this thesis.

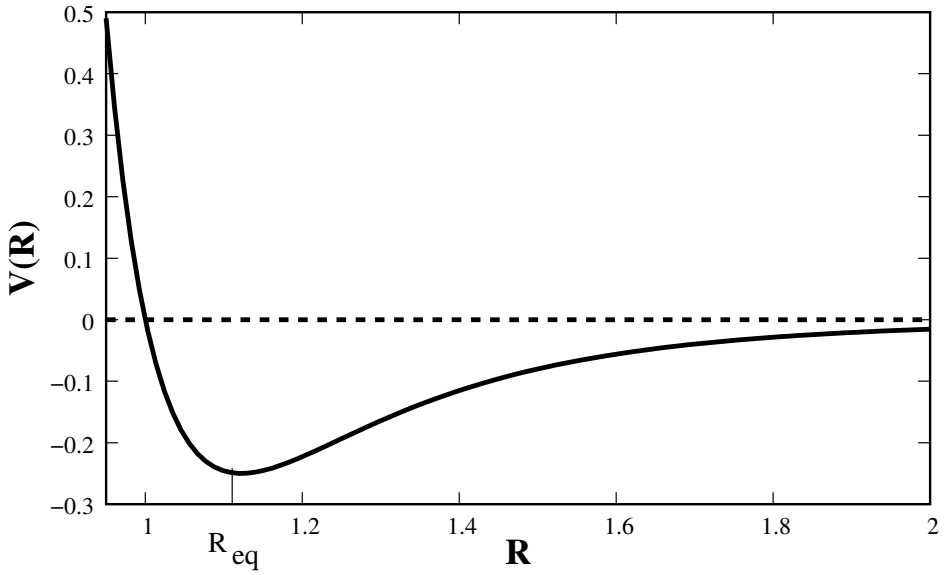

Figure 2.1: Potential energy curve for a bound electronic state of a diatomic molecule

2.1 Diatomic Potential Energy

This section will present some of the well-known diatomic potentials [19]. Since these potentials depend on a single variable, the interatomic distance, the function $E(r)$, is therefore much simpler than the multi-variable function of many body systems. It is possible to fit experimental data to adjust the parameters on these functions so that they can be customized to a given molecule. Therefore, there exist many empirical functions that can predict certain properties of diatomic molecules. For these reasons, we choose these potentials as ideal test cases for our surrogate method.

In a diatomic molecule AB, there is only one nuclear coordinate,namely, the inter-atomic distance R_{AB}. In this case, the potential energy describes the potential energy of the system, $V(R)$, as the two atoms are brought closer together, or are separated from one another. Figure 2.1 shows an example of these kind of potentials.

The point at which the curve flattens out at large inter-atomic distances is termed the dissociation limit, and represents a state where the molecule is no longer bound, being instead, two separate atoms. The dissociation energy, D_e, is the vertical distance between the dissociation limit and the minimum of the curve, found at the equilibrium bond length, R_{eq}.

2.1.1 Examples of Empirical Intermolecular Potentials

Hard sphere

The hard-sphere potential represents only the repulsive part of the Lennard-Jones potential, and uses an "infinite-or-none" representation of the repulsion. It contains one parameter, σ.

$$E(r) = \begin{cases} \infty & \text{for } r < \sigma \\ 0 & \text{for } r \geq \sigma \end{cases}$$

This model treats atoms as rigid, impenetrable spheres. It accounts for short-range repulsive forces but for long-range attraction. This potential can represent excluded volume, but it cannot account for cohesion, because it contains no intermolecular attraction.

Point centers of attraction or repulsion

Sutherland model

This model includes both attractive and repulsive forces. The minimum will be at $r = \sigma$. It is good for ion-ion interactions when n = 2.

$$E(r) = \begin{cases} \infty & \text{r} < \sigma \\ -cr^{-n} & r > \sigma \end{cases}$$

Lennard-Jones :

One of the commonly used two-body interactions is the *nnar - on pa r po n-al*. The Lennard-Jones potential used for atoms which do not form a bond but for which a shallow van der Waals minimum exists at rather large internuclear separations. This is an ideal potential for inert gases. For example, for potential between two neon atoms. This potential is given by the following expression for the interaction potential between a pair of atoms [18].

$$E_{LJ}(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]. \tag{2.2}$$

The parameters ϵ and σ are chosen to fit the physical properties of the material. For simplicity, we can choose them to be $\sigma = 1$ and $\epsilon = \frac{1}{4}$. Here we are assuming that r is one dimensional.

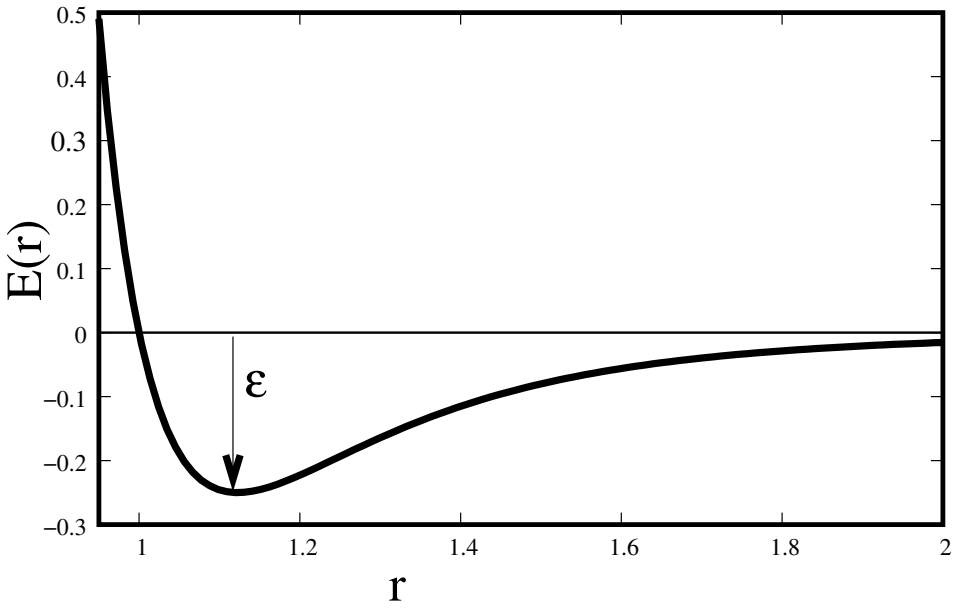

Figure 2.2: The Lennard-Jones potential for $\sigma = 1$ and $\epsilon = \frac{1}{4}$

For $r > 0$, Figure 2.2 shows that:

- The potential has an attractive tail at large r.
- It reaches a minimum at 1.122σ.
- Is strongly repulsive at shorter distance passing through zero at $r = \sigma$.
- Increasing steeply as r is decreased further.

2.1.2 Examples of Molecular-Bonding Potentials

When atoms in a molecule form a chemical bond we have to take into account the chemical nature of the species. The attractive interactions will become much stronger when the atoms become a few angstroms apart. In the case of a diatomic the only bonding interaction is a bond stretch. A very good model for this is the Morse potential.

Morse potential (handling 2-body interaction)

The Morse potential is used for two atoms which form a chemical bond, for example the ground state of two hydrogen atoms. It has the form:

$$E(r) = D_e \left\{ e^{(-2\alpha(r-r_{eq}))} - 2e^{(-\alpha(r-r_{eq}))} \right\}.$$

where

- D_e is the pair potential well depth.
- α is adjustable parameter which determines the range of the interparticle forces.
- r_{eq} is the equilibrium separation between atoms.

Rydberg potential

Rydberg potential is a slight modification of the Morse potential. The Morse potential has problems predicting properties that depend on higher derivatives of the function. It is good only up to second order [19]. The Rydberg potential answers those problems. It has the following form

$$E(r) = -D_e(1 + \alpha(r - r_{eq})e^{(-\alpha(r-r_{eq}))}).$$

Where D_e, α, r_{eq} have the same meaning as in Morse potential.

Chapter 3

Method

Most standard optimization algorithms/tools depend upon evaluating the objective function at different points. These algorithms are efficient if you have an analytical form of your objective function or if you can evaluate the function very easily. In our problem, we do not have an analytical function for our objective function. Instead, we have to solve a Kohn-Sham equation self consistently. But this is a very time consuming process, as it takes from days to several weeks to perform one SCF calculation in single 2.4 Ghz Pentium processor. So this makes it hard or difficult to use the conventional algorithms directly.

Our method uses space-mapping techniques instead. In this method, surrogate functions are employed in a sequence of optimization steps, where the original expensive objective functions are used to update the surrogates during the space-mapping process.

We follow these steps to solve the problem:

- Generate SCF values for selected geometry configurations. That is for different r values.
- Build the surrogate function.
- Find the minimum of the surrogate function.
- Update the surrogate function.

Figure 3.1 shows the above process as a flow diagram.

Our project was implemented in Matlab. To achieve the above steps, some Matlab toolboxes were used as follows:

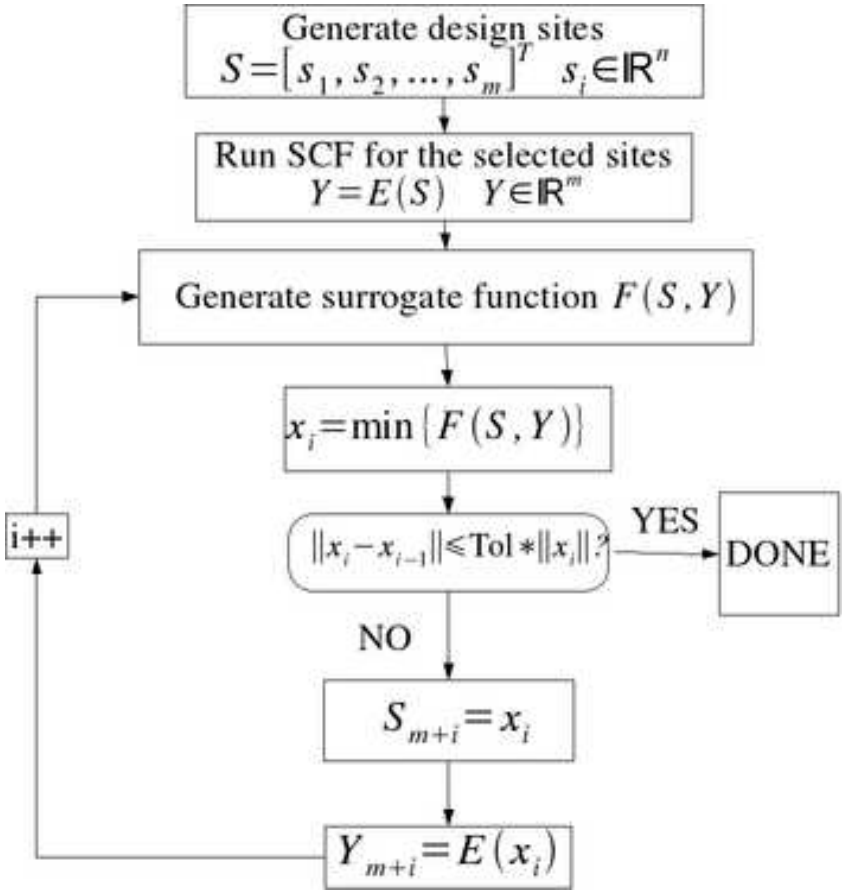

Figure 3.1: Flow diagram of our surrogate optimization.

- Use `ndgrid` or `gridsamp` to generate the sample points and evaluate the test energy function at those points to simulate the SCF calculation.
- n-dimensional spline (`interpn`) and `dacefit[17]` for building our interpolation function.
- `fminunc` or `fminsearch` Matlab functions are used to minimize our model.

Bellow is a brief introduction to the function mentioned above.

3.1 Selecting Design Sites

In addition to the convexity requirement for (1.1), we need to select our design sites so that our surrogate function is also convex. This can be achieved if one selects the sites so that the true minimum would be in the range of the selected design sites. That is, $r_1 < r_2 < ... < r_{min} < ... < r_n$. Where r_{min} is the true minimum value of our function.

3.2 DACE: A Matlab Package for Building an Interpolation Function

DACE (Design and Analysis of Computer Experiments) [17] is a Matlab toolbox for constructing kriging approximations to computer models. It can be used to construct an approximation model based on known values. This approximation model can then be used as a surrogate to predict values of new input values.

The word "kriging" is synonymous with "optimal prediction" [14]. It is a method of interpolation which predicts unknown values from data observed at known locations. It was named after D. G. Krige from South Africa.

3.2.1 Elements of DACE

The following are the basic elements needed by the kriging model.

- A set of n *po n* $(s_1, y_1), \ldots, (s_m, y_m)$, with y_i denoting the response at site $s_i \in \mathbb{R}^n$.
- A *r r on m o l* $\mathcal{F}$. This is a linear combination of basis functions $f_1, \ldots, f_p$, chosen by the user, and $\mathcal{F}(\beta, x) = f(x)^{\mathrm{T}}\beta$, where $f(x) = [f_1(x) \ldots f_p(x)]^{\mathrm{T}}$.

- A correlation model $\mathcal{R}$, so that $\mathcal{R}(\theta, x, s) \in [0, 1]$ is the correlation between the responses at x and s. The vector $\theta \in \mathbb{R}$ holds parameters of the model.

The toolbox has two major programs

- `dacefit`. This computes the elements of the Kringing model.
  CALL:
  [dmodel, perf] = `dacefit`(S, Y, regr, corr, theta0, lob, upb)

  Input parameters:
  S Design sites: an $m \times n$ array.
  Y $m \times q$ array with response at S.
  regr Handle to regression function.
  corr Handle to correlation function.
  theta hold initial guess in θ.
  lob (optional) hold lower bound on θ.
  upb (optional) hold upper bound on θ.
  Output:
  dmodel DACE model.
  perf Information about the optimization.

- `predictor`. Predicts the response at an untried site and estimate its error.
  CALL:
  y=`predictor`(x,dmodel)

  Input parameters:
  x m trial site(s) with n dimensions
  dmodel Struct with the DACE model. That is the output of `dacefit`

  Output:
  y Predicted response.

3.2.2 Regression Models

The toolbox provides regression models with polynomials of orders 0, 1, and 2 as follows.

Constant, $p = 1$:

$$f_1(x) = 1,$$

Linear, $p = n + 1$:

$$f_1(x) = 1, \quad f_2(x) = x_1, \ldots, f_{n+1}(x) = x_n,$$

Quadratic, $p = \frac{1}{2}(n+1)(n+2)$:

$$\begin{aligned} f_1(x) &= 1 \\ f_2(x) &= x_1, \ldots, f_{n+1}(x) = x_n \\ f_{n+2}(x) &= x_1^2, \ldots, f_{2n+1}(x) = x_1 x_n \\ f_{2n+2}(x) &= x_2^2, \ldots, f_{3n}(x) = x_2 x_n \\ \ldots &\quad \ldots \quad f_p(x) = x_n^2. \end{aligned}$$

3.2.3 Correlation Models

The toolbox contains the following 7 correlation functions.

Name	$\mathcal{R}_j(\theta, d_j)$
EXP	$\exp(-\theta_j \lvert d_j \rvert)$
EXPG	$\exp(-\theta_j \lvert d_j \rvert^{\theta_{n+1}}), \quad 0 < \theta_{n+1} \leq 2$
GAUSES	$\exp(-\theta_j d_j^2)$
LIN	$\max\{0, 1 - \theta_j \lvert d_j \rvert\}$
SPHERICAL	$1 - 1.5\xi_j + 0.5\xi_j^3 \quad \xi_j = \min\{1, \theta_j \lvert d_j \rvert\}$
CUBIC	$1 - 3\xi_j^2 + 2\xi_j^3, \quad \xi_j = \min\{1, \theta_j \lvert d_j \rvert\}$
SPLINE	$\begin{cases} 1 - 15\xi_j^2 + 30\xi_j^3 & \text{for } 0 \leq \xi_j \leq 0.2 \\ 1.25(1 - \xi_j)^3 & \text{for } 0.2 \leq \xi_j \leq 1 \\ 0 & \text{for } \xi_j \geq 1 \end{cases} \quad \xi_j = \theta_j \lvert d_j \rvert$

Table 3.1: Correlation functions. $d_j = w_j - x_j$

The choice of correlation function depends on the problem we are trying to solve. If the function we want to optimize is continuously differentiable, the correlation function will likely show a parabolic behavior near the origin, which means that the Gaussian, the cubic or the spline function should be chosen. Similarly problems that exhibit a linear behavior near the origin would use EXP, EXPG, LIN or SPHERICAL would usually perform better.

3.3 SPLINES

When approximating functions for interpolation or for fitting measured data, it is necessary to have classes of functions which have enough flexibility to adapt to the given data, and which, at the same time, can be easily evaluated on a computer. Traditionally polynomials have been used for this purpose. These have some flexibility and can be computed easily. However, for rapidly changing values of the function to be approximated the degree of the polynomial has to be increased, and the result is often a highly oscillating function. The situation changes dramatically when the basic interval is divided into subintervals, and the approximating or fitting function is taken to be a piecewise polynomial. That is, the function is represented by a different polynomial over each subinterval. The polynomials are joined together at the interval endpoints (knots) in such a way that a certain degree of smoothness (differentiability) of the resulting function is guaranteed. If the degree of the polynomials is k, and the number of subintervals is $n+1$ the resulting function is called a (polynomial) spline function of degree k (order $k+1$) with n knots.

Splines are highly recommended for function approximation or data fitting whenever there is no particular reason for using a single polynomial or other elementary functions such as sine, cosine or exponential functions. A spline function is a function that consists of polynomial pieces joined together with certain smoothness conditions. The choice of degree most frequently made for spline is three [5]. The resulting splines are called cubic splines. In this case, we join cubic polynomials together in such a way that the resulting spline function has two continuous derivatives everywhere.

For practical problems, spline functions have the following useful properties:

- they are smooth and differentiable up to certain degree,
- they are easy to store and manipulate on a computer,
- they are easy to evaluate, along with their derivatives and integrals,

- they are can be modified to work for higher dimensions.

The n-dimensional cubic spline interpolation was done using the `interpn` function in Matlab. interpn is a multidimensional data interpolation function. This is a general n-dimensional interpolation function. It can compute linear and cubic polynomials, Cubic spline interpolation and nearest neighbor interpolation.

Syntax F=interpn(r1,r2,...rn,E,x1,x2,...,xn,'cubic')

Where

- r1, r2, ...rn are design sites vectors.
- E is the value of the function evaluated at the design sites.
- x1, x2, ...xn are the untried sites.
- F is the interpolation function evaluated at x1, x2, ...xn.

3.4 Minimization Techniques

3.4.1 Matlab Function Fminunc

fminunc is a Matlab built in function that finds a minimum of a scalar function of several variables, starting at an initial estimate. This is generally referred to as unconstrained nonlinear optimization.

```
Syntax
[x,fval,flag,output,grad,hess] = fminunc(fun,x0,options,P1,P2,...)
```

Algorithm

It has both large scale and medium scale optimization algorithms. By default fminunc chooses the large-scale algorithm if the user supplies the gradient in function (and the GradObj parameter is set to 'on' using optimset). This algorithm is a subspace trust region method and is based on the interior-reflective Newton method [7, 8]. Each iteration involves the approximate solution of a large linear system using the method of preconditioned conjugate gradients (PCG).

Medium-Scale Optimization.

fminunc, with the LargeScale parameter set to 'off' with optimset, uses the BFGS Quasi-Newton method [9] with a mixed quadratic and cubic line search procedure. This quasi-Newton method uses the BFGS formula for updating the approximation of the Hessian matrix. The DFP formula [10], which approximates the inverse Hessian matrix, is selected by setting the HessUpdate parameter to 'dfp' (and the LargeScale parameter to 'off'). A steepest descent method is selected by setting HessUpdate to 'steepdesc' (and LargeScale to 'off'), although this is not recommended in general.

The default line search algorithm, i.e., when the LineSearchType parameter is set to 'quadcubic', is a safeguarded mixed quadratic and cubic polynomial interpolation and extrapolation method. A safeguarded cubic polynomial method can be selected by setting the LineSearchType parameter to 'cubicpoly'. This second method generally requires fewer function evaluations but more gradient evaluations. Thus, if gradients are being supplied and can be calculated inexpensively, the cubic polynomial line search method is preferable.

Limitations

The function to be minimized must be continuous. fminunc may only give local solutions. fminunc only minimizes over the real numbers, that is, x must only consist of real numbers and $f(x)$ must only return real numbers. When x has complex variables, they must be split into real and imaginary parts.

Large-Scale Optimization.

To use the large-scale algorithm, the user must supply the gradient in fun (and GradObj must be set 'on' in options. Currently, if the analytical gradient is provided in fun, the options parameter DerivativeCheck cannot be used with the large-scale method to compare the analytic gradient to the finite-difference gradient. Instead, use the medium-scale method to check the derivative with options parameter MaxIter set to 0 iterations. Then run the problem again with the large-scale method.

3.4.2 Matlab Function Fminsearch

fminsearch finds a minimum of a scalar function of several variables, starting at an initial estimate. This is generally referred to as unconstrained nonlinear optimization

```
Syntax
[x,fval,exitflag,output] = fminsearch (fun,x0,options,P1,P2,...)
```

Algorithms

`fminsearch` uses the simplex search method [16]. This is a direct search method that does not use numerical or analytic gradients as in `fminunc`. `fminsearch` is generally less efficient than fminunc for problems of order greater than two. However, when the problem is highly discontinuous, fminsearch may be more robust.

Limitations

fminsearch can handle discontinuities particularly if it does not occur near the solution. fminsearch may only give local solutions. `fminsearch` only minimizes over the real numbers, that is, x must only consist of real numbers and $f(x)$ must only return real numbers. When x has complex variables, they must be split into their real and imaginary parts.

Chapter 4

Tests and Experiments

4.1 One-Dimensional Results

One dimensional tests are performed using the Lennard-Jones (LJ) potential, equation (2.2). For $\sigma = 1$ and $\epsilon = 0.25$ the LJ potential takes the form

$$E_{LJ}(r) = \frac{1}{r^{12}} - \frac{1}{r^6}\,. \tag{4.1}$$

To find the equilibrium separation, we need to solve $\nabla_r E_{LJ}(r_{eq}) = 0$.

$$\begin{aligned} \frac{\partial E(r_{eq})}{\partial r} &= 0 \\ \frac{6}{r_{eq}^7} - \frac{12}{r_{eq}^{13}} &= 0 \\ r_{eq} &= 2^{(\frac{1}{6})} \\ &= 1.12246 \end{aligned}$$

To demonstrate the effect of the choice of the design sites, we use (4.1) as a test case and build our model for different choices of the design sites. The following four cases were considered.

Case 1:

All the design points were chosen to the left of the true minimum. That is $r_1 < r_2 < ... < r_m < r_{eq}$. Where m is the number of design sites. Figure 4.1 shows these process.

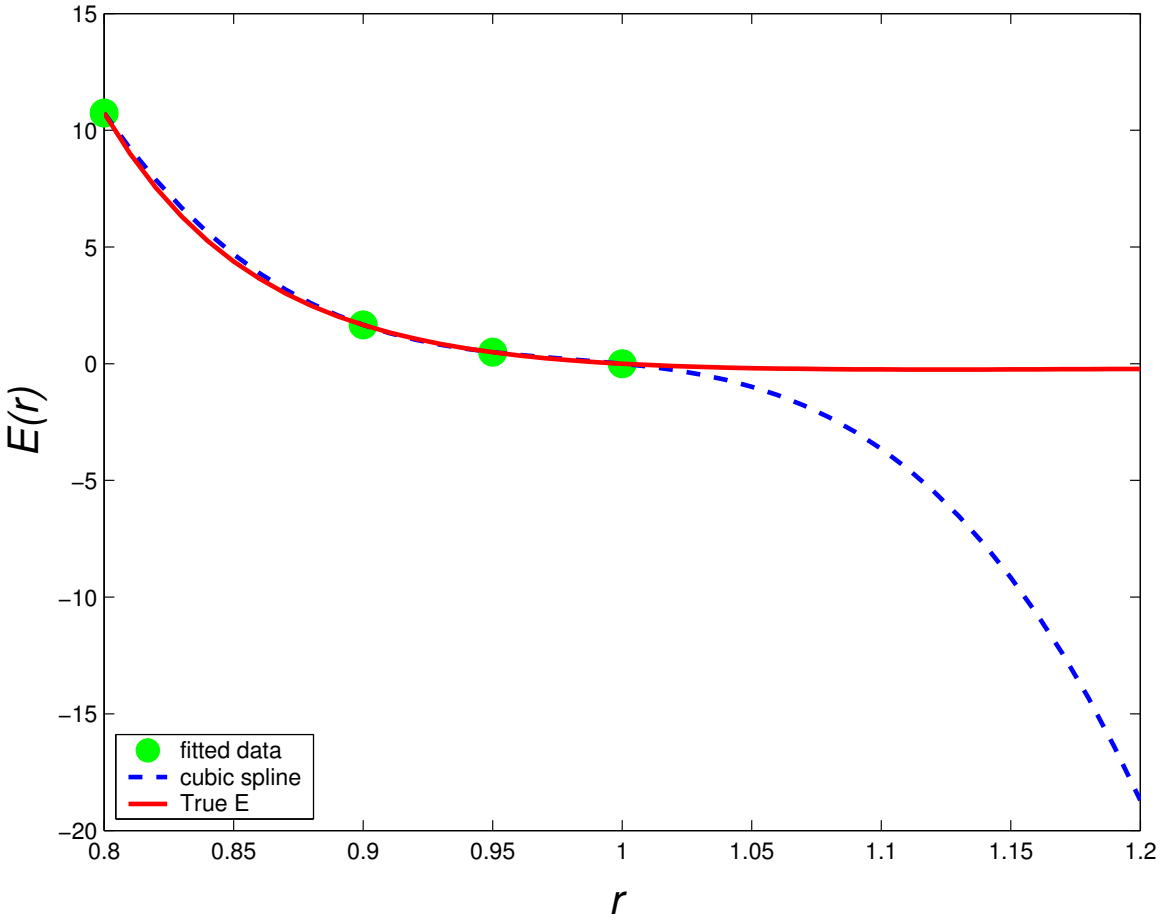

Figure 4.1: The true function (solid line), the interpolant function (dashed line) when only points to the right side of the exact minimizer are used as a design sites.

The true minimum is around 1.12246.

The figure clearly shows that the interpolant function is not a convex function. Thus the optimization method will have a hard time finding the minimum.

Case 2:

This case is the same as case 1, except that we add a few design points to the right of the true minimum. Figure 4.2 shows that the cubic spline interpolant function (dashed line) is nicely convex around the true minimum. This will be an ideal function to start with.

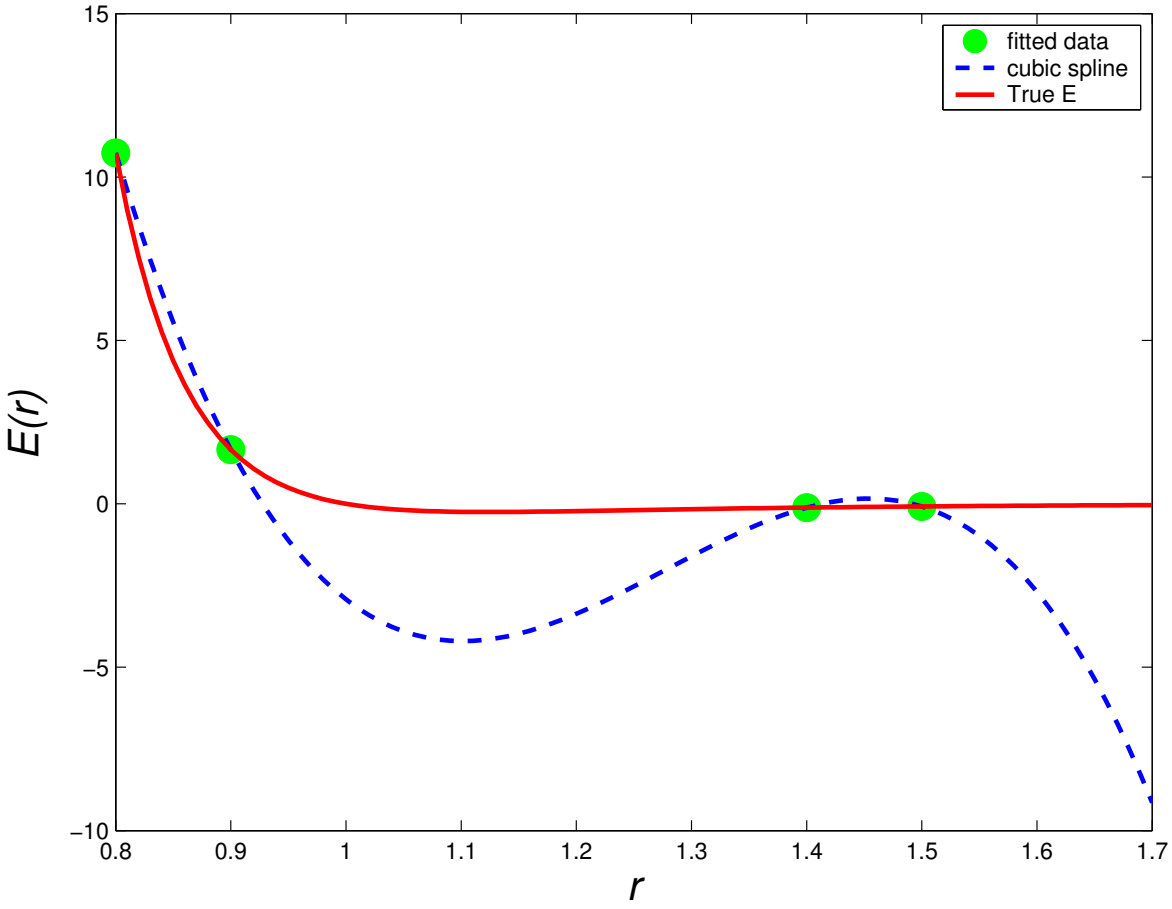

Figure 4.2: The true function (solid line), the 1-D interpolant function (dashed line) same as Figure 4.1 except that now some points are to the right of the exact minimizer.

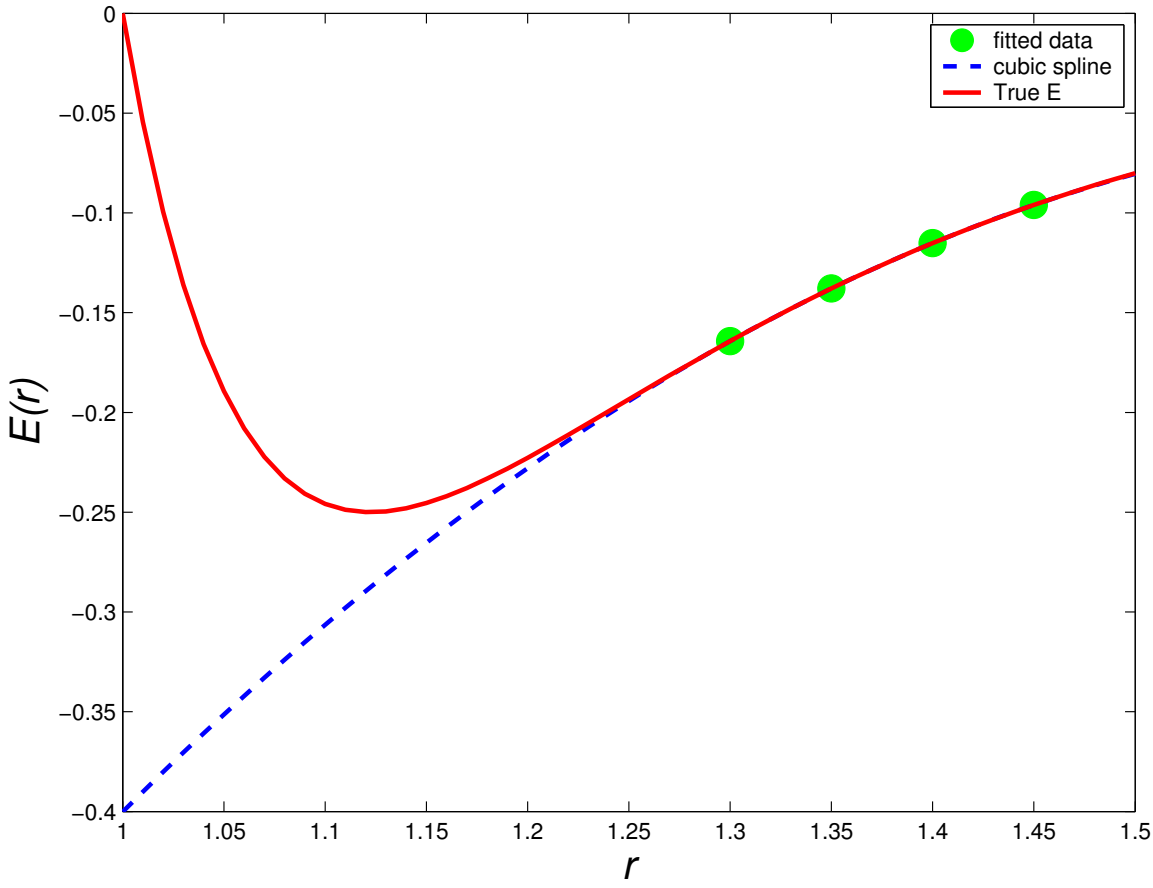

Figure 4.3: The true function (solid line), the 1-D interpolant function (dashed line) which contains only design points on the right side of the exact minimizer.

Case 3:

All the design sites are chosen to the right of the true minimum. That is $r_{eq} < r_1 < r_2 < ... < r_{m-1} < r_m$. Here also the interpolant (dashed line in Figure 4.3) is not convex. If one starts using this points as an initial design sites, there is a possibility of not finding the desired exact minimum.

Case 4:

The same as case 3: except that we have now at least one design point to the left of the true minimizer $r = 1.12246$. As can be seen in Figure 4.4, the interpolant (dashed line) is convex. It is clear that this function is guaranteed to have a minimum value which

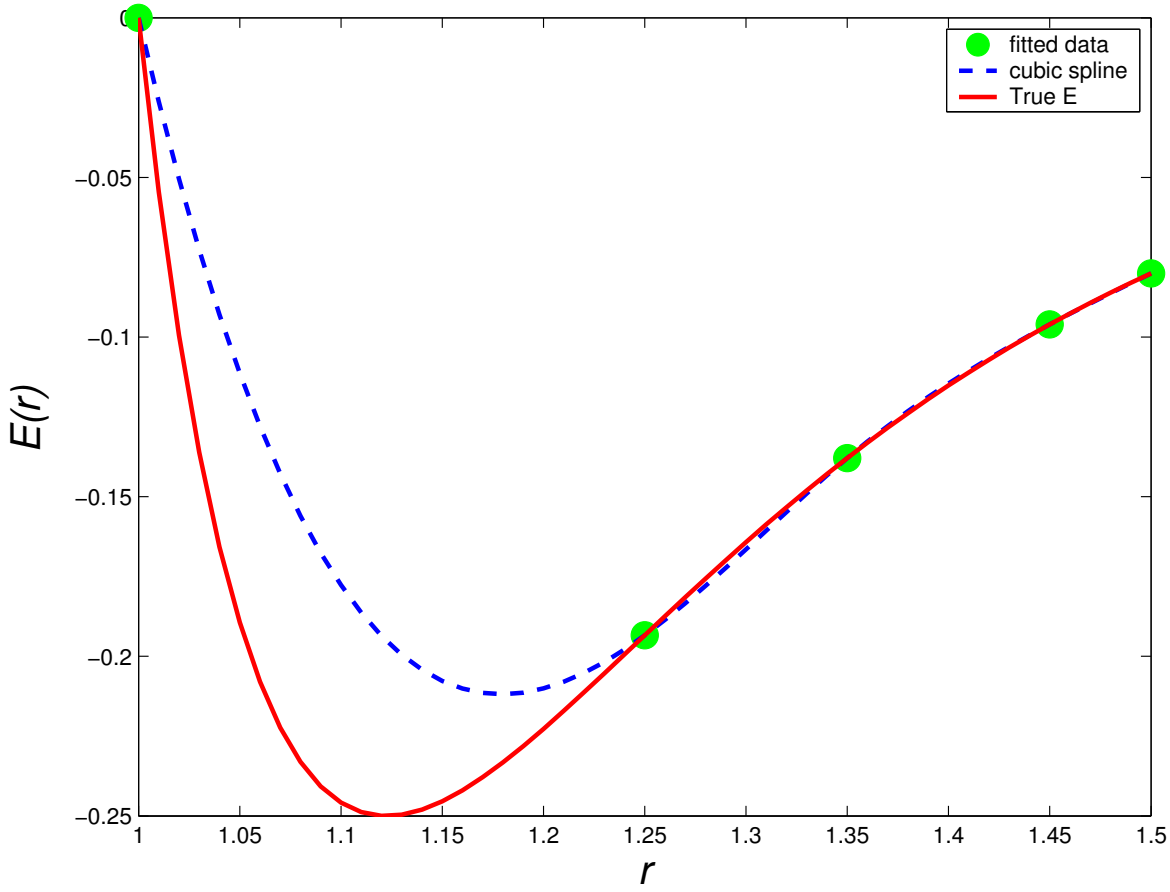

Figure 4.4: The true function (solid line), the 1-D interpolant function (dashed line) with more design points to the left of the exact minimizer.

is not that far from the exact minimum value.

Figures 4.2, 4.2, 4.3 and 4.4 illustrate that when using a surrogate function, the initial design points should be selected in such a ways that the exact minimizer is between the initial design sites. If the design sites are only on one side of the minimizer, then the surrogate might not mimic the original objective function and we could end up solving a different problem.

Minimizing our model

To test the validity of our model for the 1-dimensional test case, we used the design points used to generate Figures 4.2 and 4.4 as initial interpolant functions. Figure

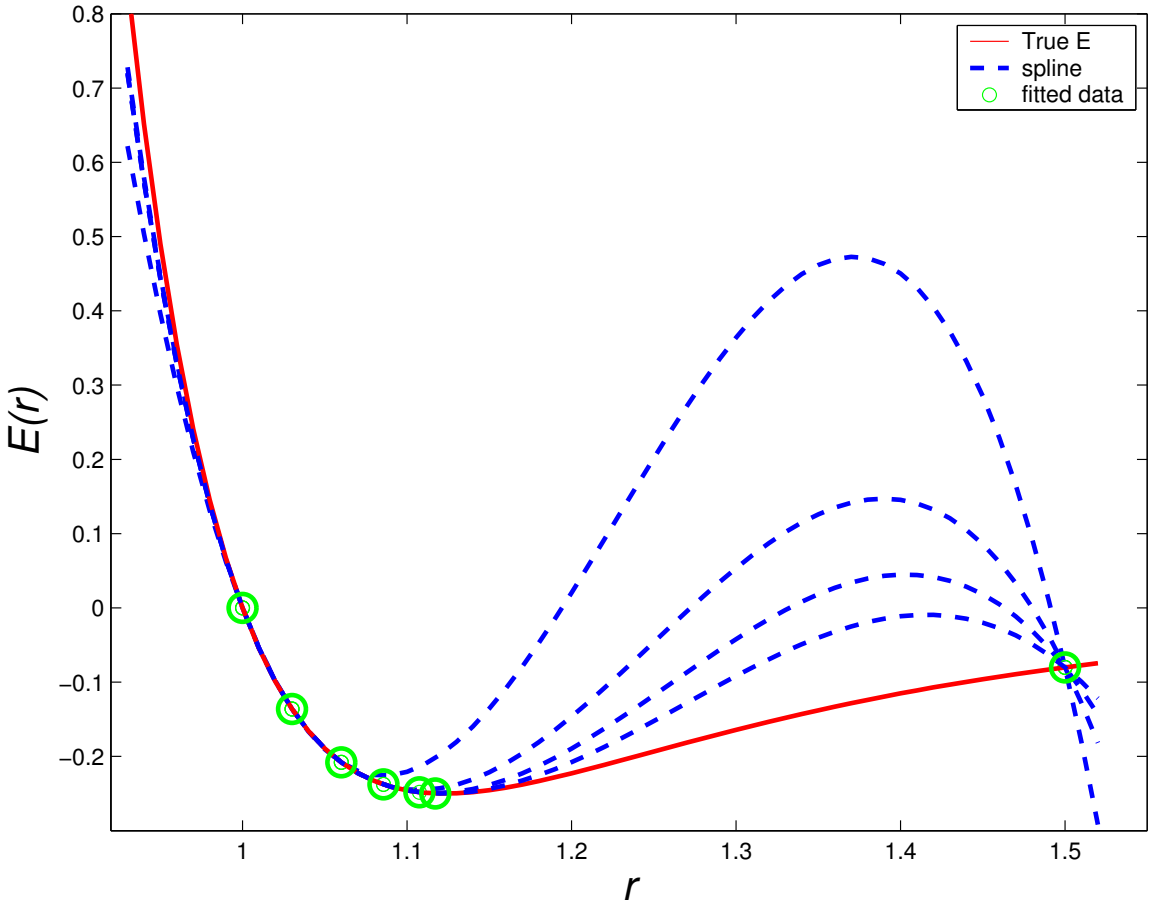

Figure 4.5: The interpolant function for successive interpolation when starting with Figure 4.2. The true minimum was found after four interpolation.

4.5 uses the same design points as the once used to generated Figure 4.2 as its first surrogate function. This surrogate was generated using `interpn` cubic spline Matlab function. This function was then used as an objective function to the fminsearch optimization function. The minimizer of this surrogate function was then checked for convergence as in Figure 3.1 and if it did not converge, we evaluate in (2.1) at this new minimizer. Now we have a new set of design sites and their responses. This new set was then used to build new surrogate function. This process continued until the desired solution was found.

As can be seen in Figure 4.5, we built four surrogate functions until we reached to the exact solution within a tolerance of 10^{-5}. This required seven SCF evaluations including the design sites used to generate the first surrogate function. Table 4.1 shows the value of

the minimizer of each of the surrogate functions built above.

Iter	r_{eq}
1	1.0856
2	1.1075
3	1.1172
4	1.1226

Table 4.1: Number of iteration versus r_{eq} for Figure 4.5.

Figure 4.6 is similar to Figure4.5 except at this time we started by using more initial design points to the right of the exact minimizer. The purpose was to demonstrate that as long as one starts by choosing the initial sites on both sides of the exact minimizer, one can reach the true minimizer within a given tolerance. This means that it really does not matter whether more initial design sites are on the right or left side of the minimizer as long as we have some on each side. Figure 4.6 also shows that eight SCF calculations needed to reach the desired solution. Table 4.2 contains the values of the minimizers for the consecutive surrogate functions.

Iter	r_{eq}
1	1.1812
2	1.1571
3	1.1310
4	1.1257
5	1.1227

Table 4.2: Number of iteration versus r_{eq} for Figure 4.6

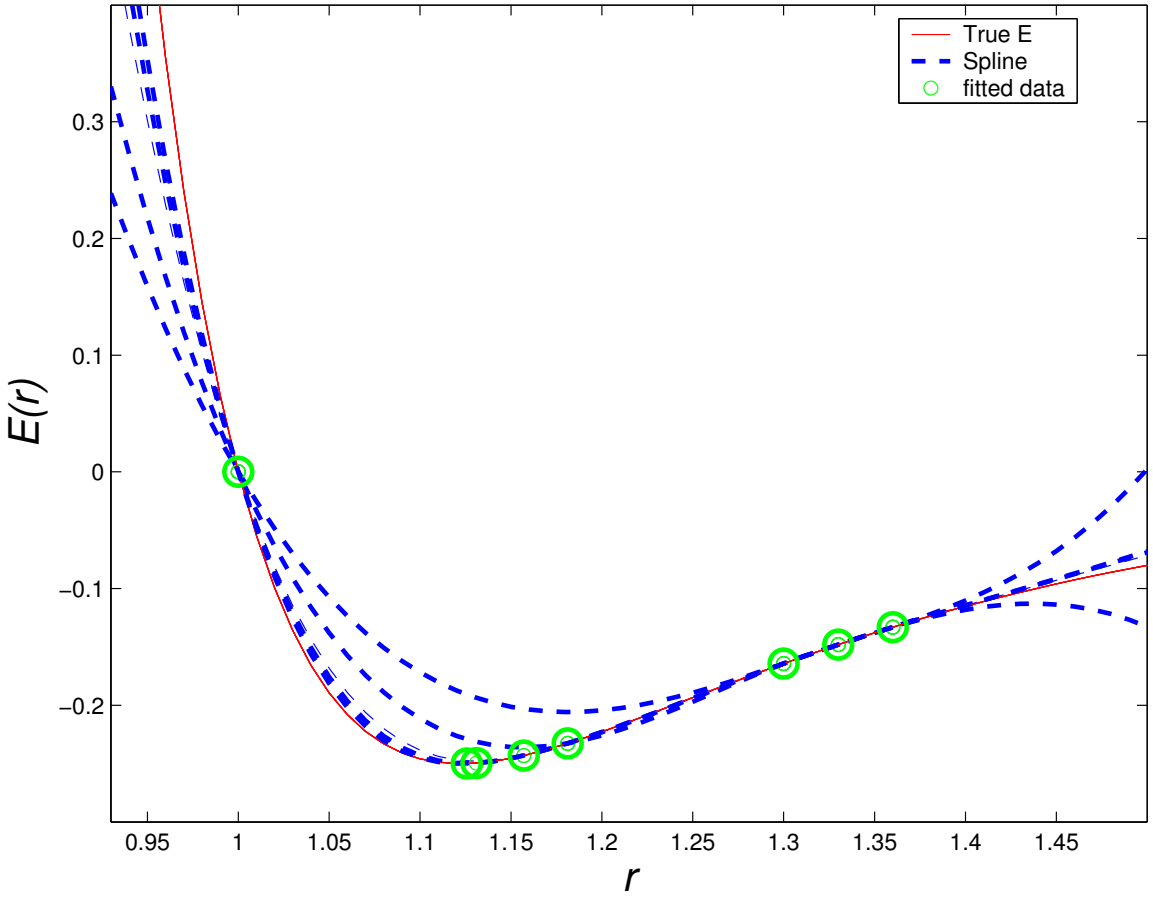

Figure 4.6: successive interpolation starting with Figure 4.4. The true minimum was found after five space-mapping iterations.

4.2 Two-Dimensional Results

$$E(r_1, r_2) = \left(1 - e^{-(r_1^2 + r_2^2)}\right)^2 \tag{4.2}$$

Figure 4.7 shows the initial design sites and its interpolated function for our two-dimensional test model (4.2). The initial response points were generated by evaluating equation (4.2) for the mesh grid generated by using $r_1 = [-2, -1.5, -1, 1, 2]$ and $r_2 = [-2, -1.5, -1, 1, 2]$. Figure 4.8 shows the minimum values of $r_1$and r_2 at each optimization cycle. Fminunc minimization function with initial points $r_0 = [1\,1]$ is used. The theoretical minimum value of (4.2) occurs when $r_1 = 0$ and $r_2 = 0$. As it is clearly shown on the figure, it took only 3 iterations to reach within 10^{-2} of the exact solution but it took a fair amount of iteration to reach the exact value with a tolerance of 10^{-5}. This is because of the nature of the Gaussian potential. It has a flat region that makes it hard for the fminserach to find the exact minimum if it starts from one of the flat regions. To avoid such problems one needs to add a kind of 'kick' to avoid the stagnation. This kick would be to add some small value say 10^{-3} to the minimum found for the current interpolant function so as to jump out of the stagnation.This aspect requires further study.

To examine the dependence of the initial point when we are using Gaussian potential and fminserach, we take the same design sites and their response for equation (4.2) but using $r_0 = [1\,2]$. Figure 4.9 shows the result of the second attempt: fminseach converged to the true solution in ten space-mapping iterations within the tolerance of 10^{-8} of the exact solution.

Even though it is not shown here, it is still true that the initial points should be on the right and left of the exact minimum for the interpolated function to have a local minimum.

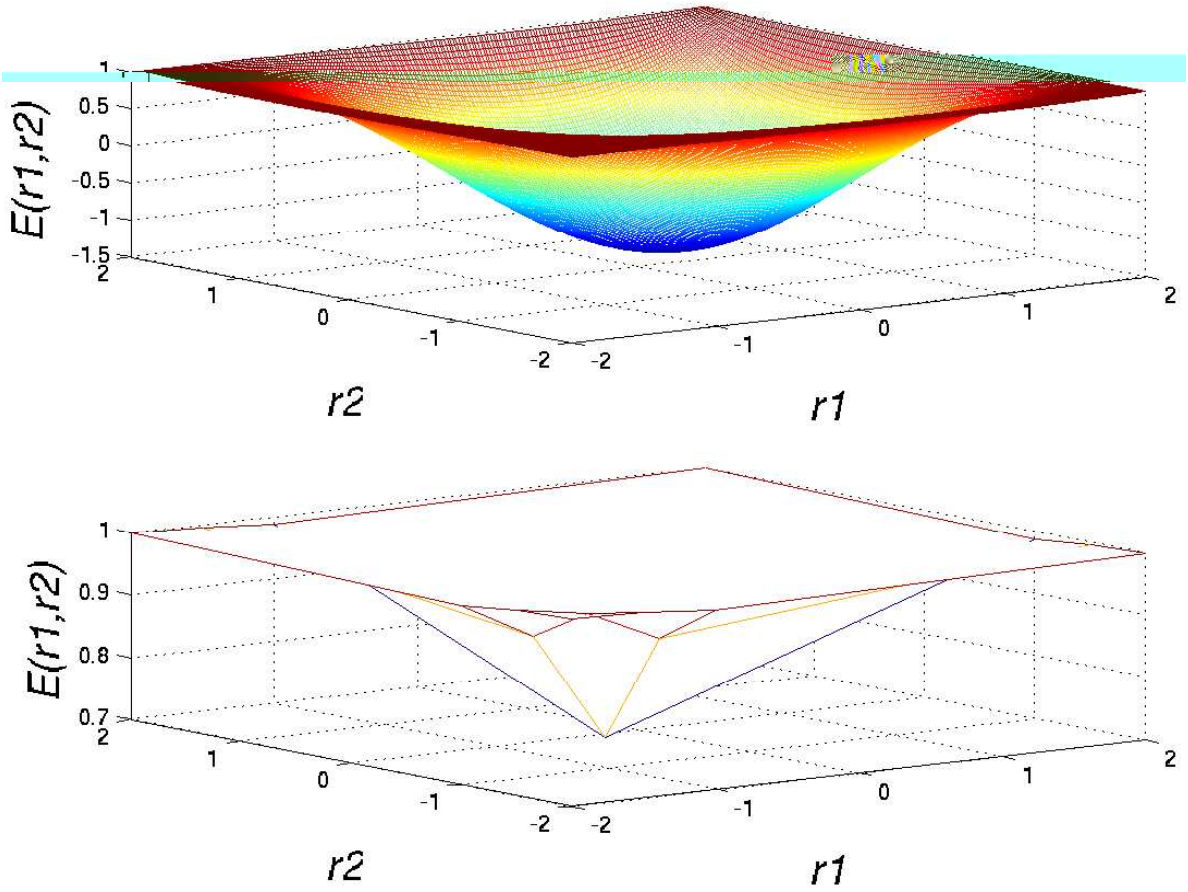

Figure 4.7: Lower : The initial design sites for (4.2). Upper: the interpolant function using cubic spline `interpn` function.

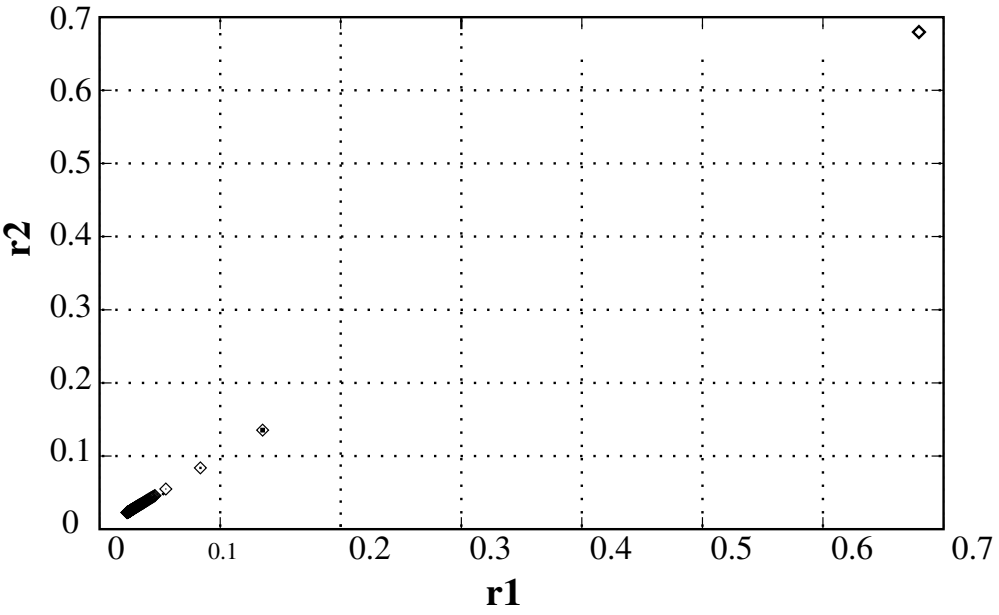

Figure 4.8: r_{min} values at each iteration starting with initial guess $r_0 = [1\ 1]$.

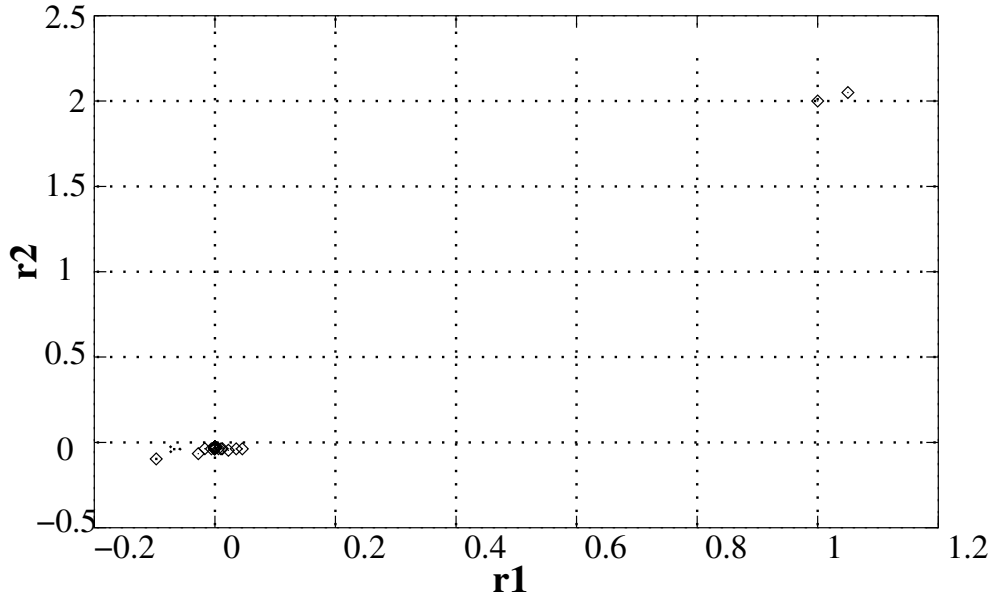

Figure 4.9: Result of 2D for $r_0 = [1\ 2]$

The second test for a 2-D system was done on an empirical total-energy function

having a parabola shape as follows:

$$E(r_1, r_2) = (r_1 - 2)^2 + (r_2 - 2)^2. \tag{4.3}$$

The exact minimizer of (4.3) occur at $r_1 = 2$ and $r_2 = 2$. Figure 4.10 shows the design sites and its corresponding surrogate function generated using dacefit function found in the DACE tool box.

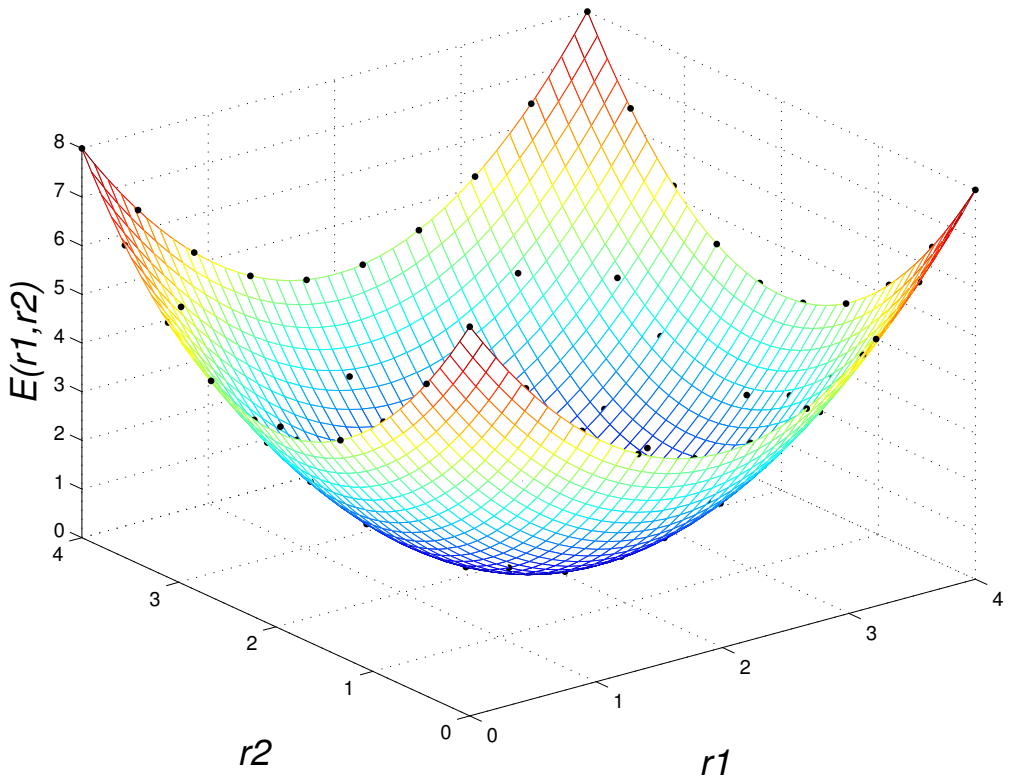

Figure 4.10: Predicted values for a 2-D system. The dot points are design sites. Using DACE

Iter	r_1	r_2
1	3.00000000	3.00000000
2	1.99998381	2.00004589
3	1.99997236	2.00002856
4	2.00002856	1.99997236

Table 4.3: Starting from $r_0 = [3\,3]$

Iter	r_1	r_2
1	3.00000000	4.00000000
2	1.99999997	2.00003948
3	1.99998767	1.99995695
4	1.99999997	2.00003948

Table 4.4: Starting from $r_0 = [3\,4]$

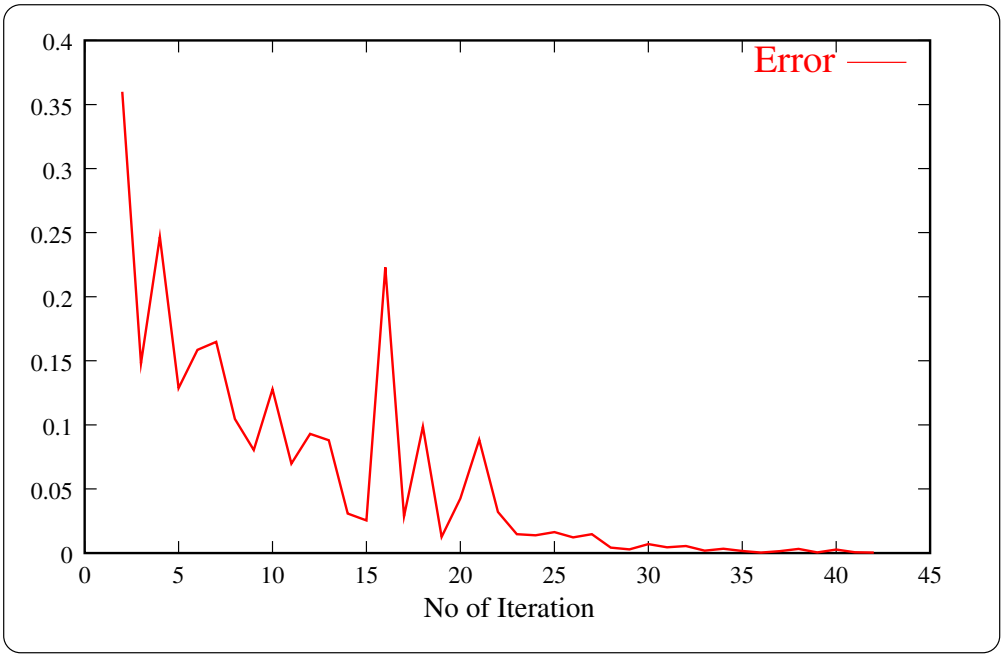

Figure 4.11: Relative error of 10-D test case.

4.3 10-Dimensional Problems

$r_1, r_2, ..., r_{10}$ are selected using `gridsamp` (a function on DACE) and

$$E(r_1, r_2, ..., r_{10}) = \sum_{i=1}^{10} (r_i - 2)^2$$

is used to generate the total-energy value at a meshgrid of $r_1, r_2, ..., r_{10}$. `dacefit` is then used to generate our surrogate. `fminsearch` using the `predictor` function is then used to find the minimum of our surrogate function. Figure 4.11 shows the error of the minimizer of each surrogate function relative to the exact solution as a function of the number of iterations. The relative error was calculated using

$$Error = \frac{||x_{min} - x_{true}||}{||x_{min}||}.$$

where x_{true} is the exact solution and x_{min} is the minimizer of the surrogate at a given iteration. The exact solution is a vector of dimension 10 whose elements are all equal 2.

Chapter 5

Concluding Remarks and Future Work

This project explored ways to speed up the crystal geometry optimization calculations. Geometry optimization involves a great number of function evaluations. Every function evaluation needs a complete cycle of expensive SCF calculations. This high cost of function evaluations usually prevents the use of standard techniques to solve the optimization problems. One of the solutions we found is to use optimization with surrogates. This process reduces the number of expensive SCF cycle calculations by doing minimization on an interpolant function.

We have tested our method for the minimization of total-energy functions. The SCF calculations were simulated by solving some known empirical potential energy functionals. Specifically, one-dimensional (Lennard-Jones potential), two-dimensional (Gaussian function), and three to twenty-dimensional (parabola function) are tested. Both kriging and interpolation were used to build our surrogate model. For finding the minimum of our model we used the Matlab fminsearch and fminunc routines.

Our preliminary result seem to indicate that surrogate optimization is a promising method for electronic-structure calculations. We found that the hardest part was selecting the initial design sites. This is because one of the requirement of initial design sites, which says that the true minimum of our problem has to be within the range of the selected design sites. In practice, we can not guarantee this, but we can use physical properties such as the dissociation energy to help as guess a suitable range for our initial design sites.

For the future work, we need to test our model in real total-energy potentials. This means that we have to generate the response of our design sites using an SCF calculation. This will help us compare head to head with the existing methods for optimization of a crystal structure. The main challenge to test in real systems is the implementation of n-dimensional spline functions or DACE in FORTRAN or C languages.

Bibliography

[1] J.W. Bandler and K. Madsen. Surrogate modelling and space mapping for engineering optimization. *p ṁ za on an n n r n* , 2:367–368, 2001.

[2] P. E. Blöchl. projector augmented wave (paw) method. *al v w* , 50:17953–17979, 1994.

[3] A.J. Booker, Jr. J.E. Dennis, P.D. Frank, D.B. Serafini, V. Torczon, and M.W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *S r ral p ṁ za on*, 17(1):1–13, 1999.

[4] R. Car and M. Parrinello. Unified approach for molecular dynamics and density-functional theory. *. v.* ., 55:2471–2474, November 1985.

[5] Ward Cheney and David Kincaid. *ṁ r al a ṁa an Co ṁp n* . Thomson Learning, February 1999.

[6] M. L. Cohn. Electronic structure of solids. *. p*, 110:293, 1984.

[7] T. F. Coleman and Y. Li. On the convergence of reflective newton methods for large-scale nonlinear minimization subject to bounds. *a ṁa al ro ra ṁṁ n* , 67(2):189–224, 1994.

[8] T. F. Coleman and Y. Li. An interior, trust region approach for nonlinear minimization subject to bounds. *SIA o rnal on p ṁ za on*, 6:418–445, 1996.

[9] R. Fletcher. A new approach to variable metric algorithms. *Co ṁp r o rnal*, 13:317–322, 1970.

[10] R. Fletcher. *ra al o o p ṁ za on.* John Wiley and Sons, 1980.

[11] M. J. Gillan. Calculation of the vacancy formation energy in aluminum. *.:Con n . a r*, 1(4):689–711, Jan 1989.

[12] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *. v.*, 136:B864–B871, November 1964.

[13] N. A. W. Holzwarth, A. R. Tackett, and G. E. Mattews. A projector augmented wave (paw) code for electronic structure calculations, part i: *a o m paw* for periodic solids in plane wave basis. *Co mp r Co m m n a on* , (135):329–347, 2001.

[14] A. G. Journel and CH. J. Huijbregts. *n n G o a* . Academic Press, 1981.

[15] W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *. v.*, 140:A1133–A1138, November 1965.

[16] J.C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the nelder-mead simplex method in low dimensions. *SIA o rnal o p m za on*, 9(1):112–147, 1998.

[17] Soren N. Lophaven, Hans Bruum Nielsen, and Jacob Sondergaard. DACE A MATLAB kriging toolbox version 2.0. Technical report, Technical University of Denmark, 2002.

[18] D.J. Tildesley M. P. Allen. *Co mp r S m la on o* . Oxford science publications, 1992.

[19] J. H. Murrell, S.Carter, S.C. Frantos, P.Huxley, and A.J.C.Varandas. *ol lar o- n al n r n on* . John Wiley and Sons, 1984.

[20] Jorge Nocedal and Stephen J. Wright. *m r al p m za on*. Springer Verlag, 1999.

[21] Encyclopdia Britannica Online. Orbital, 2004.

[22] M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J. D. Joannopoulos. Iterative minimization techniques for *ab n o* total-energy calculations: molecular dynamics and conjugate gradients. *v. o .* ., 64(4):1045–1097, October 1992.

[23] A. R. Tackett, N. A. W. Holzwarth, and G. E. Matthews. A projector augmented wave (paw) code for electronic structure calculations, part ii: *pwpaw* for periodic solids in plane wave basis. *Co mp r Co m m n a on* , (135):348–376, 2001.

[24] Loup Verlet. Computer “experiments” on classical fluids. i. Thermodynamical properties of Lennard-Jones molecules. . *v.*, 159:98–103, July 1967.

Appendix A

MATLAB code listing

A.1 1-D case using `interpn` and `fminbnd`

Listing A.1: splineloop.m

```
%  :-       ona    .  Abra  a
%  :-       a   7   2003
%            a   r v  r  pro ra    or
%       .  I          -        n  r
%    n   on.
%x = [  ,  .3,  .33,  .3 ];
%x = [0.5,  ,  .03,  .5];
x = [1,1.03,1.06,1.5];
Tol=10^(-3);
xt=2^(1/6);
it=0;
xmin=0;
while abs(xt-xmin) > Tol*xt
    it =it+1;
    y = energy(x);
    xte = 0.93:0.01:1.52;
    yy = spline(x,y,xte);
    cs = csapi(x,y);
```

```
    [xmin] = fminbnd('mysplinef',0.8,1.5,[],cs);
    disp([it xmin]);
    plot(xte,energy(xte),'r-');
    hold on
    plot(xte,yy,'b--'),
    plot(x,y,'g+'),
    legend('True E',num2str(it),'fitted data')
    [m,n]=size(x);
    x(n+1)=xmin;
    sort(x);
end
hold off
```

Listing A.2: energy.m

```
%           ona     Abra am
% n:-       ar 7 2003
% a   n   on   a     n ra        po  n  al a  a   v n r
%              on       r v r pro ram  pl n loop.m
%
%       a  a  o   or       nonl n ar m n m za on

function e=energy(r)
e = (1./r).^12 - (1./r).^6;
```

Listing A.3: mysplinef.m

```
function F=mysplinef(X,fn)
% n     a  pl n     n   on  nam
%x     a   alar val
F=fnval(fn,X);
```

A.2 1-D case using `dacefit` and `fminbnd`

Listing A.4: daceloop.m

```
%     ona     Abra a m
% n           2004
%           a m  a    pl n loop. m  x  p     a                    AC
%  oolbox  o  r a      mo  l.
x = [1,1.3,1.33,1.36];
%x = [0.5,  , .03, .5];
%x = [ , .03, .0 , .5];
Tol=10^(-3);
xt=2^(1/6);
it=0;
theta=10;;
lob=1;
upb=1.5;
x=x';
xmin=0;
while abs(xt-xmin) > Tol*xt
    it =it+1;
    y = energy(x);
    dmodel=dacefit(x, y,@regpoly0, @corrgauss,theta,lob,upb);
    [xmin] = fminbnd('predictor',0.8,1.5,[],dmodel);
    disp([it xmin]);
    [n,m]=size(x);
    x(n+1)=xmin;
    sort(x);
end
```

A.3 2-D case using `interpn` and `fminsearch`

Listing A.5: energy2d.m

```
%a   n   on   a    n ra       po  n  al a  a   v n  r
%
%      a  a  o   or      nonl n ar  m n m za  on
```

```
function e=energy2(r1,r2)
%  = ( ./(r . 2 r2. 2)).    - ( ./(r . 2 r2. 2)). 3;
e = r1.^2 + r2.^2;
```

Listing A.6: myinterp2.m

```
function F=myinterp2(x,r1,r2,E)
F=interp2(r1,r2,E,x(1),x(2),'cubic');
```

Listing A.7: driver.m

```
%  :-          ona     Abra am
%  :-         Apr 2    2003

%   r v r    n   on.
% I          n  rpn  o   r a   a mo  l an
% m n  ar     o m nm z       mo  l
% or 2        m

function Xmin=Driver(X0);
% r v r  or  n r  2

%      n
r1=[-2,-1.5,-1,-0.5,0.5,1,1.5,2];
r2=[-2,-1.5,-1,-0.5,0.5,1,1.5,2];
r1f=-2:0.02:2;
r2f=-2:0.02:2;
OPTIONS = [];

delta=0.05; %      a  a  k  k
Tolx=10^(-7);

[R1f,R2f] = meshgrid(r1f,r2f);
```

```
it=0;
xmins=0;      % or         m n  val
ymins=0;

while (it<100)
     it=it+1;

      % n ra              n
     [R1,R2] = meshgrid(r1,r2);
     % n ra    n r  val   a             n       .
     E=energy2d2(R1,R2);
     [msx,nsx]=size(xmins);
     [msy,nsy]=size(ymins);
     [m1,n1]=size(r1);
     [m2,n2]=size(r2);
     Xmin=fminsearch('myinterp2',X0,OPTIONS,r1,r2,E);
      disp([it   Xmin]);
      [m1,n1]=size(r1);
      flagx=1;
      flagy=1;
      for i=1:n1,
           if(abs(r1(i)-Xmin(1))<=Tolx*(max(r1(i),Xmin(1))))
                flagx=0;
           end
      end
      for i=1:n2,
           if(abs(r2(i)-Xmin(2))<=Tolx*(max(r2(i),Xmin(2))))
                flagy=0;
           end
      end
      if(flagx & flagy)
           r1(n1+1)=Xmin(1);
```

```
            r2(n2+1)=Xmin(2);
            r1t=sort(r1);
            r1=r1t;
            r2t=sort(r2);
            r2=r2t;
            xmins(nsx+1)=Xmin(1);
            ymins(nsy+1)=Xmin(2);
        else
            X0=X0+delta;
        end
    end

    subplot(2,1,1)
    mesh(R1f,R2f,VI);
    subplot(2,1,2)
    mesh(R1,R2,E);
```

A.4 10-D case using `dacefit` and `fminsearch`

Listing A.8: energynd.m

```
%*****************************************
%    : ona    . Abra a m
%   :
%a   n   on    a     n ra     a   m pl
% S m pl   ar m on     p   parabola   n r
%    n   on
%
%       a  a  o    or       nonl n ar  m n m za  on
%*****************************************

function e=energy_nd(x)
[m n]=size(x);
```

```
e=0;
for i=1:n
    e=e + (x(:,i)-2).^2;
end
```

Listing A.9: 10dloopdriver.m

```
%*****************************************
%  :  ona   . Abra am
%  :
%*****************************************
X0=[2,1,3,1,2,2,1,3,1,2];
trval(1:10)=2;
x1(1:10)=0;
x2(1:10)=4;
q(1:5)=2;
q(6:10)=4;
%x= r   amp([0 0 0 0 0;4 4 4 4 4],3);
x=gridsamp([x1;x2],q);

lob(1:10)=0.001;
upb(1:10)=4;
theta(1:10)=10;
tol=1e-3;
it=1;

fid = fopen('10d_error.txt','w');

options = ...
optimset('LargeScale','off','Display','Iter','MaxFunEvals',1000);

ee=energy_nd(x);
dmodel=dacefit(x, ee,@regpoly0, @corrgauss,theta,lob,upb);
```

```
%xmin= mnn('predictor',X0,option,model);
xmin = fminsearch('predictor',X0,options,dmodel);
disp([it xmin]);
[n,m]=size(x);
x(n+1,:)=xmin;
x0=xmin;
tmp=xmin;

while (1 && it <100)
    it=it+1;
    ee=energy_nd(x);
    dmodel=dacefit(x, ee,@regpoly0, @corrgauss,theta,lob,upb);
    %[xmin] = mnbn('predictor',0.8,2,[], model);
    %xmin= mnn('predictor',X0,option,model);
    xmin = fminsearch('predictor',X0,options,dmodel);
    disp([it xmin]);
    x0=xmin;
    [n,m]=size(x);
    x(n+1,:)=xmin;
    norm_xmin=norm(xmin)
    norm_tmp=norm(tmp)
    norm_val=norm(xmin - tmp)
    norm_true=norm(trval)
    fprintf(fid,'%3d %12.8f %12.8f %12.8f %12.8f\n',...
        it,norm_xmin,norm_tmp,norm_val,norm_true);
    if(norm(xmin - tmp) <= tol*norm(xmin))
        break;
    else
        tmp=xmin;
        continue;
    end
    % or (x);
```

```
end
ee=energy_nd(x);
dmodel=dacefit(x, ee,@regpoly0, @corrgauss,theta,lob,upb);
%x_m n= m n n ('pr      or',X0,op  on , mo  l);
xmin = fminsearch('predictor',X0,options,dmodel);
disp([it xmin]);
fclose(fid)
```