A DFT++ Code with Evolutionary Optimization Technique

Will Tipton

May 4, 2009

Abstract

We wrote an electronic structure code in the DFT++ formalism to compute quantum-mechanical energies and wavefunctions of solid crystalline Germanium. DFT++ is a Lagrangian formalism and the Kohn-Sham wavefunctions are variational solutions of an optimization problem. We implement a novel evolutionary algorithm to perform this optimization as well as several more traditional techniques (preconditioned conjugate gradient, steepest-descent) for the sake of comparison. The code works quite well. This document describes the methodology behind the project as well as the practical aspects of the code's implementation and use.

Contents

1 Methodology **2**
1.1 Context and motivation 2
1.2 DFT++ 2
1.3 Evolutionary optimization 5

2 Results and discussion **8**

A Genetic Algorithm **12**

B DFT++ main code **16**

References **17**

Chapter 1

Methodology

1.1 Context and motivation

Density Functional Theory is a very important electronic structure method. DFT++ is an alternate formulation of density functional theory in which the Kohn-Sham equations are expressed as an optimization problem [4]. This optimization step is the most computationally expensive part of the approach. In an attempt to accelerate convergence, we will perform the necessary optimizations using a genetic algorithm.

Through this work, we gain a deeper understanding of the theoretical underpinnings as well as the practical implementation of a density functional theory code. We learn more about a powerful, modern optimization technique and hopefully make an improvement on a current method of performing electronic structure calculations. The mini-course notes by Professor Arias were extremely helpful in understanding DFT++ and implementing the code [1].

1.2 DFT++

The DFT++ method is based on a Lagrangian reformulation of the Kohn-Sham equations. That is, a Lagrangian $L_{\text{LDA}}(\psi_i(r), \phi(r))$ is defined where the $\psi_i(r)$ are the Kohn-Sham effective single-particle electronic states which satisfy an orthonormality condition and $\phi(r)$ is the electrostatic or Hartree potential.

$$L_{\text{LDA}}(\psi_i(r), \phi(r)) = -\frac{1}{2}\sum_i f_i \int \mathrm{d}^3 r \psi_i^*(r) \nabla^2 \psi_i(r) \tag{1.1}$$

$$+ \int \mathrm{d}^3 r V_{\text{ion}}(r) n(r) + \int \mathrm{d}^3 r \epsilon_{\text{xc}}(n(r)) n(r) \tag{1.2}$$

$$- \int \mathrm{d}^3 r \phi(r)(n(r) - n_0) - \frac{1}{8\pi} \int \mathrm{d}^3 r ||\vec{\nabla}\phi(r)||^2 \tag{1.3}$$

The f_i are Fermi-Dirac fillings and the electronic density itself is given by $n(r) = \sum_i f_i ||\psi_i(r)||$.

For the complete formalism, we refer the reader to the original paper which was quite well written [4]. Here, it will suffice to define the operators which will be useful later, show that minimizing the Lagrangian with respect to the ψ_i and ϕ will recover the Kohn-Sham and Poisson equations, and finally, put it together into an algorithm.

First, recall that we have expanded our wavefunction and Hartree potential in terms of basis functions:

$$\psi_i(r) = \sum_\alpha b_\alpha(R) C_{\alpha,i} \tag{1.4}$$

$$\phi(r) = \sum_\alpha b_\alpha(r) \phi_\alpha \tag{1.5}$$

Everything in DFT++ is written explicitly in matrix-form for clarity of computation. Luckily, there are few operators which depend explicitly on the choice of basis. We discuss those first. The overlap and Laplacian operators are given by

$$O_{\alpha,\beta} = \int \mathrm{d}^3 r b^*_\alpha(r) b_\beta(r) \tag{1.6}$$

$$L_{\alpha,\beta} = \int \mathrm{d}^3 r b^*_\alpha(r) \nabla^2 b_\beta(r) \tag{1.7}$$

Note that for orthogonal bases, O is proportional to the identity.

Next, we consider the need to change representation of objects between a vector of expansion coefficients in the bases and the space of function values in real space. We define the forward transfer operator

$$I_{p,\alpha} = b_\alpha(p) \tag{1.8}$$

and the inverse operator which we call J.

If we now let F be the matrix of electronic occupancy, it becomes easy to write down what we want: the Lagrangian and its derivatives in terms of our matrix data and operators.

$$L_{\text{LDA}}(C, \phi) = -\frac{1}{2}\text{Tr}(FC^\dagger LC) + (Jn)^\dagger (V_{\text{ion}} + OJ\epsilon_{\text{xc}}(N) - O\phi) + \frac{1}{8\pi}\phi^\dagger L\phi \tag{1.9}$$

$$\frac{\partial L_{\text{LDA}}}{\partial Y^\dagger} = (I - OCC^\dagger)HCFVU^{-1/2} + OCVQ(V^\dagger[\tilde{H}, F]V) \tag{1.10}$$

$$\frac{\partial L_{\text{LDA}}}{\partial \phi^\dagger} = -\frac{1}{2}OJn + \frac{1}{8\pi}L\phi \tag{1.11}$$

with

$$H = -\frac{1}{2}L + I^\dagger(\text{Diag}V_{\text{sp}})I \tag{1.12}$$

$$\tilde{H} = C^\dagger H C \tag{1.13}$$

where n is of course the electronic density coefficients with respect to our basis.

So, the DFT++ energy algorithm may be summarized. Note that the ionic locations are fixed, and we've chosen some basis functions which will be described by the index α. These basis functions are denoted b_α. We write the vector of expansion coefficients of our *normalized* wavefunctions in this basis as Y.

1. Calculate the ionic potential

$$(V_{\text{ion}})_\alpha = \int b^*_\alpha(R)V_{\text{ion}}(R)\text{d}^3r \tag{1.14}$$

   where $(V_{\text{ion}})_\alpha$ represents the overlap of the ionic potential with each basis function.

2. Choose initial wavefunctions W randomly and make them orthonormal with $Y = WU^{-1/2}$ where U is the matrix of wave function overlaps.

3. Optimize the wavefunction Y until the energy is converged. For example, if we are using the steepest-descent optimization algorithm, this may involve:

   (a) Calculate the energy as described below.

   (b) Calculate the gradient of the energy with respect to the wavefunction.

   (c) Take a step down the gradient G of the energy with

$$Y_n = Y_{n-1} - \lambda G \tag{1.15}$$

   where λ is some step size.

The steps to do the energy calculation itself are important. Given a trial wavefunction W, we may calculate the energy as follows.

1. Calculate the overlap matrix $U = W^\dagger OW$.

2. Calculate the orthonormal wavefunctions $Y = WU^{-1/2}$.

3. Calculate the charge density with $n = f\ \text{diag}((IC)(IC)^\dagger)$ where f is orbital occupancy.

4. Calculate the vector expansion coefficients of the electrostatic potential by solving the Poisson equation: $\hat{\phi} = -4\pi L^{-1}OJn$.

5. Finally, calculate the energy by

$$E = -\frac{1}{2}f\mathrm{Trace}(W^{\dagger}LWU^{-1}) + \tilde{V}^{\dagger}n + \frac{1}{2}n^{\dagger}J^{\dagger}O\hat{\phi} + n^{\dagger}J^{\dagger}OJ\epsilon_{\mathrm{xc}}n \quad (1.16)$$

We will implement various traditional optimization algorithms: steepest-descent and conjugate gradient methods along with preconditioners for them. However, our focus here is on the use of a more novel technique, the genetic algorithm

1.3 Evolutionary optimization

In nature, species tend to become better suited to their environments. The process by which this happens is known as natural evolution and was first described by Charles Darwin.

There is natural variation in the phenotypes of organisms within a species. The environment may favor one phenotype over another in that the favored organism's genetic information (DNA) is preferentially transmitted to the next generation. For instance, consider a population of giraffes who feed on the leaves of trees. Notice that giraffes with longer necks will be able to access more food than short giraffes. Thus, in times when food is scarce, the lower-hanging fruit will be consumed more quickly, after which only the taller giraffes will be able to eat.

If all the short giraffes die off, then only the tall giraffes will survive to mate, and (assuming that "tallness" is something that can be passed from parent to child) the next generation will tend to be taller than the previous one. In this way, the environment has exerted a pressure on the giraffe population and modified its gene frequencies to be more suited to the environment.

This process can be thought of as a strategy for approaching optimization problems. Suppose we have some objective function which we may sample but for which we may have no easily-manipulated analytical form. We can search for a solution using an algorithm inspired by natural evolution as follows.

We first choose some collection of trial solutions. We calculate how good each of these solutions is by evaluating our objective function on each candidate solution. We then create a collection of "child" solutions by "mating" the initial group of solutions, choosing the better solutions to be parents with a higher frequency. This mating operation is problem specific, but it is important that the properties of the parent solution that make it a good solution are transmitted to the child solution. By repeating this process,

our sets of solutions will become better and better solutions to the problem, and we iterate until we have found a sufficiently good solution.

For the sake of concreteness, we describe the basics of an evolutionary algorithm for another difficult optimization problem in solid state physics: ab initio crystal structure prediction. The crystal structure that is physically realized is the one with the lowest quantum-mechanical free energy. Given a crystal structure, we can calculate its energy using electronic structure methods such as DFT.

However, the inverse problem is difficult: the energy is a very complicated function of the ionic coordinates, and we have no explicit form for it. So, several authors have attempted a genetic algorithm for the problem's solution. As described above, the idea is to generate a set of candidate solutions. Each of these is a crystal structure with random lattice parameters and ionic positions. We may then calculate each structure's energy using, say, a DFT code and mate the best of these to create even better solutions.

One possible mating operation is known as the slicing crossover. To perform this operation, two parent crystals are considered. The code then simply takes a slice from the unit cells of each parent. For example, it may take the top half of one parent's unit cell and the bottom half of the other's. It then slaps the two halves together to create the child solution which it relaxes and evaluates with an energy code. After some generations of this algorithm, the best solution found is likely to be the physically-realized crystal structure [5],[2],[3].

In the present work, we are interested in optimizing Kohn-Sham wavefunctions to minimize the energy. We have written expressions for the energy above: it is something we can compute explicitly in order to evaluate potential solutions.

The essential genetic information on which we will be acting is the wavefunction itself and, in particular, its vector representation in the computer. That is, two trial solutions to the energy minimization problem will be two different wavefunctions.

In particular, the genetic algorithm works as follows:

1. Choose random initial population. Evaluate the energy and fitness of each organism.

2. Create the new child generation:

    (a) Promote some number of the best solutions from the previous generation to the new one.

    (b) Create the rest of the child solutions using the crossover operation described below. For each new solution created this way, two distinct parent solutions are selected based on their own fitnesses.

    (c) Evaluate the energy and fitness of each new solution.

3. Set (child population) → (parent population) and go to 1.

We now discuss the salient features of the two most important components to this algorithm: parent selection and the mating operation. Parent selection is very important in a genetic algorithm because the preferential selection of parent solutions based on their quality is the only way a genetic algorithm encourages the improvement of the population.

We define the fitness of a solution in terms of its energy and the energies of all the other solutions in its population by

$$f = \frac{energy - worstEnergy}{bestEnergy - worstEnergy}$$

where $energy$ is the solution's energy, and $bestEnergy$ and $worstEnergy$ are defined in terms of the overall population. In this way, the fitness is a number between 0 and 1 where 1 indicates the best solution. Now, to select a parent, we simply consider a random organism in the parent population. We generate a random number, r, between 0 and 1. If the fitness of the randomly selected solution is greater than r, then we choose it to be a parent. Otherwise, we try again. In this way, better solutions will more often be parents.

In order to describe the mating operation, it is important to understand the representation of the wavefunctions in our code. A solution wavefunction is an N by M matrix whose M columns are wavefunctions of the first M lowest energy states written in terms of N expansion coefficients. We implement two separate crossover operations, both of which are based on combining the matrix elements of two parents to make the child. In the first, we simply combine, randomly, columns from the two parents to make the child. In the second, we consider each column in turn and take a contiguous chunk of the corresponding column from each parent to make the column of the child.

Chapter 2

Results and discussion

The genetic optimization process mostly fails in that it usually converges slowly to a solution which is much higher energy than the true minimum-energy solution. However, with hindsight, we can gain some insight into genetic algorithms, in general, and into this optimization problem, specifically, that may help us make more progress in the future.

We believe that there are two main reasons that the genetic algorithm fails to work as hoped. Firstly, the crossover method does not actually do a good job of transmitting essential characteristics of the parent solutions to children. In particular, the essential characteristic is the local electron density. Because energetically-important interaction between electrons fall off with the distance between electrons, the crossover operation should try to maintain coherently large chunks of the electron density in child solutions.

This would imply that the crossover operation should work in real-space. It should combine solutions by taking chunks of parent wavefunctions sampled on the real-space grid. However, due to our representation of the wavefunctions in terms of delocalized basis functions, the way in which we parts of the parent wavefunctions to make a child does not actually exploit the partial spatial separability of the problem. This shows that the representation of solutions is very important when designing genetic algorithms; the representation should be designed in order to explicitly encapsulate the pieces of information which need to be maintained coherently in the population. Because of this failure, the algorithm converges very slowly.

The second major issue with our we have identified explains why the algorithm converges to an incorrect solution instead of simply converging slowly to the correct one. The reason is that the algorithm does not effectively search over the whole space of solutions. It simply generates a random starting population and then mixes and matches parts of it in order to generate new solutions. Nowhere is genuinely new genetic information introduced into the population, and so, if the necessary sequences are not present anywhere in the original population, they can not be found in a

solution.

The solution to this found in many evolutionary algorithms has biological inspiration. These codes implement a mutation operation which is capable of randomly perturbing the data in one solution to create another. Although this operation rarely results in a better solution, such an operation introduces new genetic information into the population and can prevent the algorithm from getting "stuck."

Luckily, we also implemented conjugate gradient and steepest descent optimizers so that we can demonstrate the efficacy of the rest of our electronic structure code. The convergence time of each of these traditional operators is shown in Fig. 2.1.

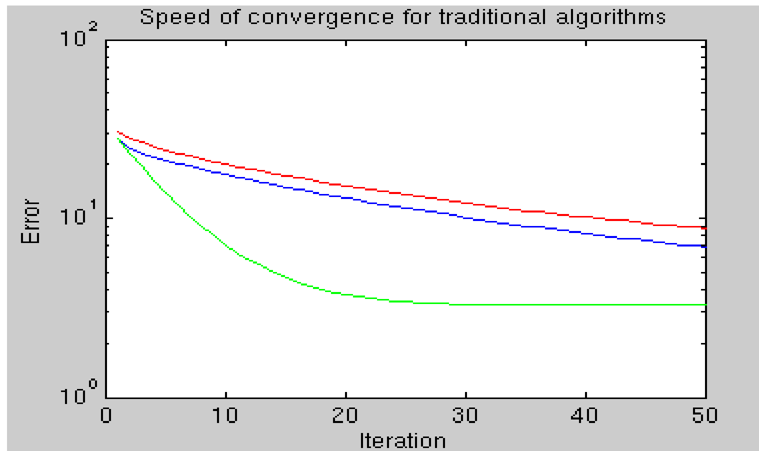

Figure 2.1: Convergence time of different traditional optimization algorithms. Red: steepest-descent. Blue: conjugate-gradient. Green: preconditioned conjugate-gradient.

These convergence tests were done for the solution of the Hydrogen atom. We can visualize the electron densities which we solved for. Fig 2.2 shows an orbital with s-character, and 2.3 shows a solution with p-character.

Finally, we present a calculation on solid Germanium that we performed. We used our DFT++ code with a pseudopotential by Starkloff-Joannopoulos to perform an energy calculation on an 8-atom diamond cell with experimental lattice parameters. A cross-section of a (100) plane of the ground-state electron density is shown in Fig. 2.4.

Notice that the cores of the atoms have no electronic density. This is because we used a pseudopotential.

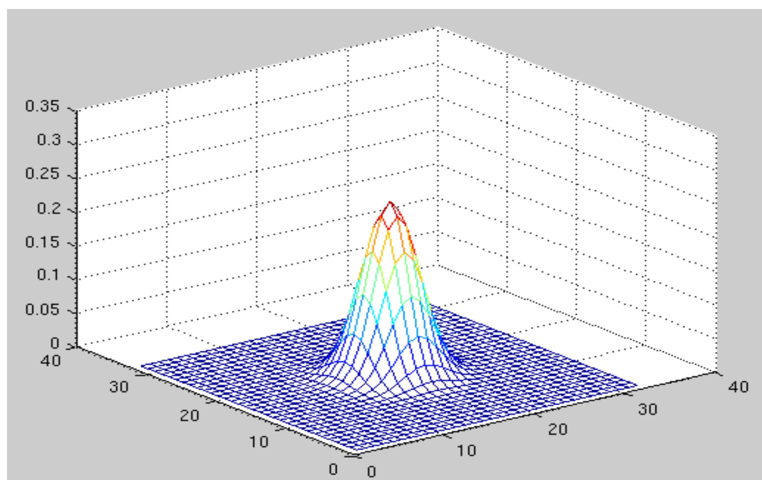

Figure 2.2: H-orbital with s-character.

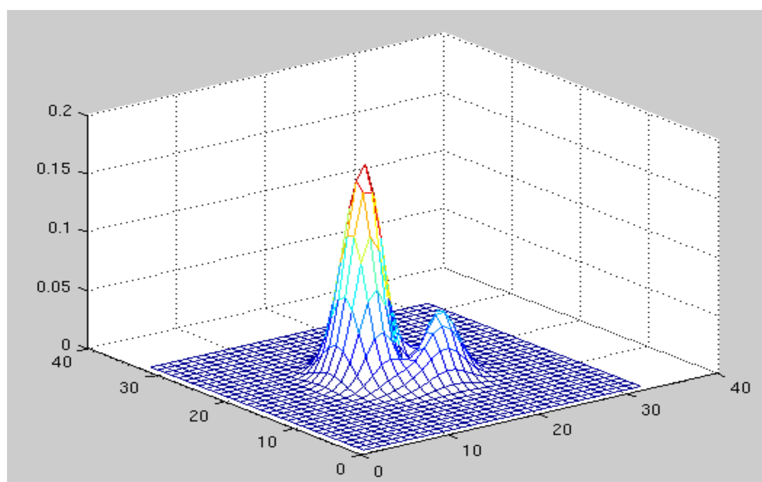

Figure 2.3: H-orbital with p-character.

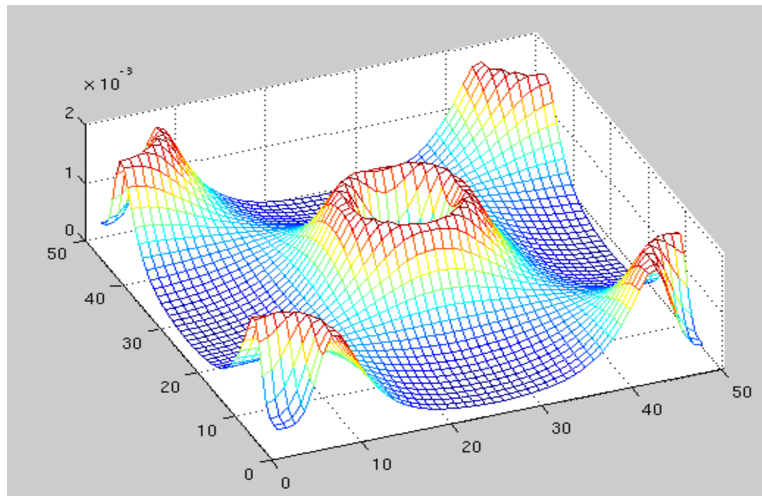

Figure 2.4: (100) slice of Ge ground state electronic density.

Appendix A

Genetic Algorithm

```
function [out_wfn] = Genetic_min(Nit,Ns)

global gbl_active;

Nsolns_per_gen = 12;
Npromote = 2;
Nmutants = 0;

% make initial population
parent_energy = zeros(1,Nsolns_per_gen);
parent_pop = cell(1,Nsolns_per_gen);
for i = 1:Nsolns_per_gen
    parent_popi=(randn(length(gbl_active),Ns)+i*randn(length(gbl_active),Ns));
    parent_energy(i) = GetE(parent_popi);
end

% evaluate fitness of initial population
parent_fitness = get_fitness(parent_energy);

% perform main iteration loop
for i = 1:Nit
    % make children population::
    %    - promote best few solutions to next gen
    child_energy = zeros(1,Nsolns_per_gen);
    child_pop = cell(1,Nsolns_per_gen);
    for j = 1:Npromote
        indx = get_nth_best(parent_fitness,j);
        child_popj = parent_popindx;
        child_energy(j) = parent_energy(indx);
    end
```

```
31      % introduce mutants (genetic variation)
32      for j = Npromote+1:Npromote+Nmutants
33          child_popj=(randn(length(gbl_active),Ns)+i*randn(length(gbl_active),Ns));
34          child_energy(j1) = GetE(child_popj);
35      end
36      %    - finish the children population with a crossover operation
37      for j = Npromote+1+Nmutants:Nsolns_per_gen
38          success = 0;
39          while (success == 0)
40             % select two parents based on fitnesses
41             parentA = parent_popselect_parent(parent_pop, parent_fitness);
42             parentB = parent_popselect_parent(parent_pop, parent_fitness);
43             while (parentA == parentB)
44                parentB = parent_popselect_parent(parent_pop, parent_fitness);
45             end
46             % "mate" them to make a child
47             child_popj = mate_wavefcns(parentA,parentB);
48             % make sure we have a unique solution (maintain genetic
49             % diversity)
50             success = 1;
51             for k = 1:j-1
52                if (sum(sum(child_popk-child_popj)) == 0)
53                    success = 0;
54                    break
55                end
56             end
57          end
58          % calculate energy
59          child_energy(j) = GetE(child_popj);
60      end
61
62      % evaluate children population and make children fitness vector
63      child_fitness = get_fitness(child_energy)
64
65      % life goes on, children become parents, etc
66      parent_pop = child_pop;
67      parent_fitness = child_fitness;
68      parent_energy = child_energy;
69
70      min(child_energy)
71
72  end
73
74  % out wavefunction is the best solution of the last generation
```

```
75
76  out_wfn = child_popget_nth_best(child_fitness,1);
77
78  end
79
80  function child = mate_wavefcns(parentA,parentB)
81      sz = size(parentA);
82
83      if (rand() > 0.5)
84          child = parentA;
85          for iwfn = 1:sz(2)
86              chunksz = ceil(0.75*sz(1).*rand());
87              startloc = ceil(0.25*sz(1).*rand());
88              child(startloc:startloc+chunksz,iwfn) = parentB(startloc:startloc+chunks
89          end
90      else
91          child = parentA;
92          frac = rand();
93          for iwfn = 1:sz(2)
94             if (rand() > frac)
95                 child(:,iwfn) = parentB(:,iwfn);
96             end
97          end
98      end
99
100 end
101
102 function indx = get_nth_best(fitnesses,n)
103     for i = 1:n
104         y = find(fitnesses == max(fitnesses));
105         fitnesses(y(1)) = 0;
106     end
107     indx = y(1);
108 end
109
110 function indx = select_parent(pop, fitnesses)
111     success = 0;
112     Nparents = length(pop);
113
114     while (success == 0)
115         indx = ceil(Nparents.*rand());
116         if (fitnesses(indx) > rand())
117             success = 1;
118         end
```

```
119     end
120 end
121
122 function fit = get_fitness(pop_energies)
123     Norgs = length(pop_energies);
124
125     maxE = max(pop_energies);
126     minE = min(pop_energies);
127     fit = zeros(1,Norgs);
128     for j = 1:Norgs
129         fit(j) = (pop_energies(j) - maxE)/(minE - maxE);
130     end
131 end
```

Appendix B

DFT++ main code

```
1
2  %Set orbital occupancies
3  global gbl_f;
4  gbl_f = 2; % 2 electrons per orbital
5
6  global gbl_Vdual;
7
8  % set up the SHO potential V with frequency w = 2
9  w =  2;
10 dr = sqrt(sum((r-(ones(prod(S),1)*sum(R,2)'/2)).^2,2));
11
12 %# Ge pseudopotential
13 Z=4;
14 lambda=18.5;
15 rc=1.052;
16 Gm=sqrt(G2);
17 Vps=-2*pi*exp(-pi*Gm/lambda).*cos(Gm*rc).*(Gm/lambda)./(1-exp(-2*pi*Gm/lambda));
18 for n=0:4
19   Vps=Vps+(-1)^n*exp(-lambda*rc*n)./(1+(n*lambda./Gm).^2);
20 end
21 Vps=Vps.*4*pi*Z./Gm.^2*(1+exp(-lambda*rc))-4*pi*Z./Gm.^2;
22 n=[1:4];
23 Vps(1)=4*pi*Z*(1+exp(-lambda*rc))*(rc^2/2+1/lambda^2* ...
24                        (pi^2/6+sum((-1).^n.*exp(-lambda*rc*n)./n.^2)));
25 % define the global operator Vdual
26 gbl_Vdual = CJ(Vps.*Sf);
27
28 %# Number of states
29 Ns=16;
30
```

```
31  % Make initial random wavefunctions
32  randn('seed',0.2004);
33  %W=(randn(prod(S),Ns)+i*randn(prod(S),Ns));
34  W=(randn(length(gbl_active),Ns)+i*randn(length(gbl_active),Ns));
35
36  %# Run finite difference test
37  more off; %# View output as it is computed
38  Fdtest(W);
39
40  % Make wavefunctions orthonormal
41  U = W'*O(W);
42  W = W * U^(-1/2);
43
44
45  %# Allow for more digits in printouts
46  format long
47  %# Optimize!
48  W = Sd(W,20); % start with 20 iterations of simple sd() to get nearer to the minimu
49  W = W*inv(sqrtm(W'*O(W))); %# Restart as orthonormal functions
50  [W, Epccg] = Pccg(W,50); % 50 iterations of pccg
51
52  [Psi, epsilon] = GetPsi(W);
53  n=abs(CI(Psi(:,1))).^2;
54  %# Basic slice from ??100?? edge of cell
55  sl=reshape(n(1:S(1)*S(2)),S(1),S(2));
56  %# Make and view image
57  mesh(sl); pause;
58  %# 110 slice cutting bonds (assumes cube)
59  sl=reshape(n(find(M(:,2)==M(:,3))),S(1),S(2));
60  %# Expand by 2, drop data to restore (approximate) aspect ratio
61  li=find(rem([1:size(sl,1)],3)~=0); sl=sl(li,:);
62  %# Make and view image
63  mesh(sl);
64
```

References

[1] T. Arias, *Dft++ minicourse*. http://dft.physics.cornell.edu/minicourse/.

[2] D. M. Deaven, *Molecular geometry optimization with a genetic algorithm*, Physical Review Letters, 75 (1995).

[3] C. W. Glass, *Uspex–evolutionary crystal structure prediction*, Computer Physics Communications, 175 (2006).

[4] S. Ismail-Beigi and T. A. Arias, *New algebraic formulation of density functional calculation*, Computer Physics Communications, 128 (2000), p. 1.

[5] G. Trimarchi, *Global space-group optimization problem: Finding the stablest crystal structure without constraints*, Physical Review B, 75 (2007).