KSSOLV User's Guide

Chao Yang
Juan C. Meza
Computational Research Division
Lawrence Berkeley National Laboratory
Berkeley, CA 94720

February 11, 2009

Contents

1 Introduction **5**

2 Getting Started **7**
- 2.1 Download KSSOLV . . . 7
- 2.2 Setting the environmental variable `KSSOLVPATH` . . . 7
- 2.3 Test Problems . . . 7
- 2.4 Setting up an Molecular System . . . 8
  - 2.4.1 Initialization . . . 8
  - 2.4.2 Specifying the Atom List . . . 8
  - 2.4.3 Specifying the Positions of the Atoms . . . 9
  - 2.4.4 Kinetic Energy Cutoff . . . 10
  - 2.4.5 Checking the Attributes of an `Molecule` Object . . . 11
- 2.5 Solving the Kohn-Sham Equations . . . 12
- 2.6 Understanding the Output from KSSOLV . . . 14
- 2.7 Visualizing the Results . . . 14

3 KSSOLV Classes **17**
- 3.1 Atom . . . 18
- 3.2 Molecule . . . 19
- 3.3 Ham . . . 20
- 3.4 Wavefun . . . 21
- 3.5 FreqMask . . . 23
- 3.6 Operator Overloading . . . 23

4 KSSOLV Methods **25**
- 4.1 Planewave Discretization . . . 25
- 4.2 Pseudopotentials . . . 27
- 4.3 The finite dimensional Kohn-Sham problem . . . 28
- 4.4 SCF . . . 30
  - 4.4.1 Charge mixing . . . 31
  - 4.4.2 Trust Region . . . 31
  - 4.4.3 LOBPCG for Solving Linear Eigenvalue Problems . . . 32
- 4.5 DCM . . . 32
- 4.6 Options for SCF and DCM functions . . . 33

Bibliography **39**

Chapter 1

Introduction

KSSOLV is a MATLAB toolbox for solving a class of nonlinear eigenvalue problems also known as the Kohn-Sham equations. This type of problem arises from electronic structure calculation which is nowadays an essential tool for studying the microscopic quantum mechanical properties of molecules, solids and other nanoscale materials. Through the density functional theory (DFT) formalism, one can reduce the many-body Schrödinger equation used to describe the electron-electron and electron-nuclei interactions within an atomistic system to a set of single-electron equations that have far fewer degrees of freedom. These equations are first derived by Kohn and Sham in [14], and they have the form

$$H(\rho)\psi_i = \lambda_i \psi_i, \quad i = 1, 2, ..., n_e, \tag{1.1}$$

where n_e is the number of electrons in the system, $\{\psi_i(r)\}$ $(i = 1, 2, ..., n_e)$ is a set of single electron wavefunctions that are mutually orthonormal, $\rho(r)$ is the total electron charge defined by

$$\rho(r) = \sum_{i=1}^{n_e} |\psi_i(r)|^2, \tag{1.2}$$

and

$$\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_{n_e}$$

are the n_e smallest eigenvalues of $H(\rho)$, a linear operator that depends on ρ. The operator H is known as the Kohn-Sham Hamiltonian and defined by

$$H(\rho) = -\frac{1}{2}\Delta + V_{ion}(r) + \rho \star \frac{1}{|r|} + V_{xc}(\rho), \tag{1.3}$$

where Δ is the Laplacian operator, V_{ion} and V_{xc} are known as the ionic and exchange-correlation potentials respectively [14], and $\star$ denotes the convolution operator.

The Kohn-Sham equations are the the Euler-Lagrange equations for the total energy minimization problem

$$\begin{array}{ll} \min & E_{total}^{KS}(\psi_i) \\ \text{s.t} & \psi_i^* \psi_j = \delta_{i,j}, \end{array} \tag{1.4}$$

where

$$E_{total}^{KS} = \frac{1}{2}\sum_{i=1}^{n_e} \int_\Omega |\nabla \psi_i|^2 dr + \int_\Omega \rho V_{ion} dr + \frac{1}{2}\int_\Omega \int_\Omega \frac{\rho(r_1)\rho(r_2)}{|r_1 - r_2|} dr_1 dr_2 + E_{xc}(\rho), \tag{1.5}$$

$E_{xc}(\rho)$ is the exchange-correlation energy defined in [14].

After (1.1) is discretized, it becomes a set of nonlinear equations that bear similarity to algebraic eigenvalue problems found in standard linear algebra textbook. The main feature that distinguishes the Kohn-Sham equations from the standard linear eigenvalue problem is that the matrix operators in these equations are functions of the eigenvectors to be computed. Due to this nonlinear coupling between the matrix operator and its eigenvector, the Kohn-Sham equations are much more difficult to solve than a standard linear eigenvalue problem. Currently, the most widely used numerical method for solving the this type of problem is the Self Consistent Field (SCF) iteration, which we will examine in detail in Chapter 4. An alternative approach which is designed to minimize the total energy (1.5) directly will also be reviewed there. Both methods have been implemented in KSSOLV to allow users to compute the solution to the Kohn-Sham equations associated with various molecules and solids.

KSSOLV is designed using the objected oriented programming features available in MATLAB. Several classes are created in KSSOLV to define both physical objects such as atoms and molecules and mathematical entities such as wavefunctions and Hamiltonians. These classes are defined in a hierarchical fashion so that a molecule object can be easily constructed from atom objects. The use of objects not only allows users with a minimal physics or chemistry background to quickly assemble a realistic atomistic system, but also allows them to develop and compare new numerical methods for solving the Kohn-Sham equations in a user friendly environment.

This user's guide will help users get familiar with various features of the toolbox. It is organized as follows. In next chapter, we will demonstrate how the toolbox can be used to set up simple molecular systems and how the Kohn-Sham equations associated with these systems can be solved by calling the numerical methods currently implemented in KSSOLV. In Chapter 3, we will take a closer look at various classes defined in KSSOLV, and show how KSSOLV objects can be created and modified. Chapter 4 focuses on numerical methods for solving the Kohn-Sham equations that have been implemented in KSSOLV. In particular, we will examine the SCF iteration and the DCM algorithm. We will discuss the importance of using the trust region technique to maintain monotonic reduction of the total energy function and how charge mixing can be used to accelerate and stabilize the convergence of the SCF iteration.

Chapter 2

Getting Started

2.1 Download KSSOLV

KSSOLV can be downloaded from

```
http://www.crd.lbl.gov/~chao/KSSOLV
```

2.2 Setting the environmental variable KSSOLVPATH

Once you retrieve the package, take a look at the README file in the directory `kssolv`. The file contains a minimal set of instructions for using the package.

Before running KSSOLV, one should define the environmental variable `KSSOLVPATH`. If you are running MATLAB under Unix/Linux operating system, this can be done by adding, for example,

```
setenv KSSOLVPATH  /home/cyang/kssolv
```

to your .cshrc if you use `csh`. If you use `bash`, use

```
export KSSOLVPATH=/home/cyang/kssolv
```

If you are running MATLAB under Windows, click on `Start->My Computer`, and then click `View system information` on the left (`System Task`) panel. Continue with a click on `Advanced`, followed by a click on `Environmental variables`. If `KSSOLVPATH` is not among the defined environmental variables, click new, type in the variable name KSSOLVPATH, and the directory in which KSSOLV is installed.

2.3 Test Problems

The package contains a number of test problems that one can experiment with. Each test problem represents a particular molecule or periodic system. The system is created in a setup file. Table 2.1 contains the names of all setup files and a brief description for each of them. It also shows the number of electron pairs (occupied states (`nocc`) for most systems (with the exception of the electron quantum dot example in which `nocc` is the actual number of electrons in the dot).

setup file name	nocc	Description
c2h6_setup.m	7	an ethane molecule
co2_setup.m	8	a carbon dioxide molecule
h2o_setup.m	4	a water molecule
hnco_setup.m	8	an isocyanic acid molecule
qdot_setup.m	8	a 8-electron quantum dot confined by an external potential
si2h4_setup.m	6	a planar singlet silylene molecule
sibulk_setup.m	6	a silicon bulk system
sih4_setup.m	4	a silane molecule
ptnio_setup.m	43	a Pt_2Ni_6O molecule

Table 2.1: Setup files for examples included in KSSOLV

2.4 Setting up an Molecular System

To create a new system, a user can simply take one of the existing setup files and modify the construction of the `Molecule` object. A user can also change of the size of the primary cell or the kinetic energy cutoff of the existing setup file to examine the change in the convergence behavior of the numerical method or the quality of the computed solution.

In the following, we will use `sih4_setup.m` as an example to demonstrate how a molecular system can be easily set up in KSSOLV. The source code contained in `sih4_setup.m` is shown in Figure 2.2.

2.4.1 Initialization

In KSSOLV, an molecular system is defined by first creating an `Molecule` object, an particular instance of the `Molecule` class to be defined in Chapter 3, using

```
mol = Molecule();
```

Here `mol` is a user defined variable name and `Molecule` is a KSSOLV keyword. This single line of code creates an empty `Molecule` object `mol` with no particular attribute.

2.4.2 Specifying the Atom List

Properties of an `Molecular` object are specified by using the `set` method associated with the object. For example,

```
mol = set(mol,'atomlist', alist);
```

is used in Figure 2.2 to specify the list of atoms contained in `mol`. Here the string `'atomlist'` is the name of an attribute associated with an `Molecule` object predefined by KSSOLV, and `alist` is a user defined variable which supplies the actual atom list. In Figure 2.2, the variable `alist` is defined earlier as an array of `Atom` objects. Each `Atom` object can be defined by its chemical symbol or atomic number. For example, the silicon atom can be defined by

```
a = Atom('Si')
```

or

```
a = Atom(14);
```

Here, `Atom` is a KSSOLV keyword associated with the atom class (see Chapter 3 for details.)

The atom list array `alist` can also be defined by

```
alist(1) = Atom('Si');
for j = 2:5
   alist(j) = Atom('H');
end
```

2.4.3 Specifying the Positions of the Atoms

The position of each atom in the atom list is specified by an $n_a \times 3$ array (`xyzmat` in Figure 2.2). The ith row of this array gives the x, y, z coordinates of the ith atom in the atom list. This array can be passed to the `mol` object through the `set` method as one can see in Figure 2.2. The unit of the coordinates should be in Bohr (1 Bohr = 0.5291772083 Angstrom).

Although it is perfectly OK to define this array directly, it is sometimes more convenient to first define the relative position of each atom in a unit (super)cell (of a periodic lattice) specified by three lattice vectors $\mathbf{c}_1$, $\mathbf{c}_2$ and $\mathbf{c}_3$. If we denote the relative displacement of the atom along $\mathbf{c}_i$ by ξ_i, then the absolute x, y, z coordinates of the atom correspond to the three components of the row vector

$$r = \xi_1 \mathbf{c}_1^T + \xi_2 \mathbf{c}_2^T + \xi_3 \mathbf{c}_3^T = (\xi_1\, \xi_2\, \xi_3) C,$$

where

$$C = \begin{pmatrix} \mathbf{c}_1^T \\ \mathbf{c}_2^T \\ \mathbf{c}_3^T \end{pmatrix}. \tag{2.1}$$

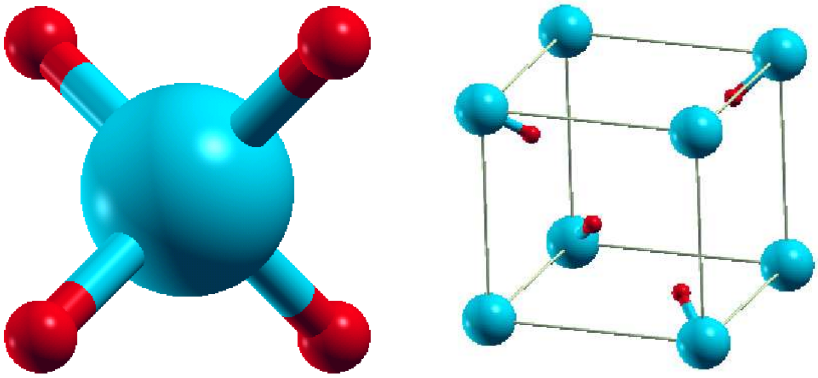
The 3×3 matrix C defines a unit supercell in KSSOLV. It can be used to set the `'supercell'` attribute of a `Molecule` object. For example,

```
mol = set(mol,'supercell',10*eye(3));
```

sets the unit supercell of `mol` to a $10 \times 10 \times 10$ cube.

If the relative positions of n_a atoms within a unit supercell defined by C are placed in an $n_a \times 3$ array `coefs`, then the absolute coordinates of these atoms can be obtained from `coefs*C`.

We should point out that the (x, y, z) coordinates associated with the atoms contained in a `Molecule` object can be negative even though the lattice vectors used to specify the unit supercell are usually in $\mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+$. Because the unit supercell is assumed to periodically extended throughout the entire domain, (x, y, z) is equivalent to $(x + m\mathbf{c}_1, y + n\mathbf{c}_2, z + \ell\mathbf{c}_3)$ for any integer m, n and ℓ, i.e., we can always bring an atom into the specified unit supercell if its coordinates are outside of the unit supercell. For example, if the coordinate of the silicon atom is chosen to be $(0, 0, 0)$, then the atomic positions shown on the left side of Figure 2.1 are equivalent to those shown on the right side of the figure.



(a) Atomic positions outside of the unit supercell

(b) Atomic positions inside of the unit supercell

Figure 2.1: Equivalent rendering of the atomic positions of Si and H in a silane molecule

2.4.4 Kinetic Energy Cutoff

In order to discretize the Kohn-Sham equations with an appropriate number of planewave basis functions, one must specify a kinetic energy cutoff E_{cut} through

```
mol = set(mol,'ecut',cutoff_value);
```

where `cutoff_value` is a user supplied number in the unit of Rydberg (Ryd).

This energy cutoff defines a maximum magnitude limit for the wave numbers g_j allowed in the planewave expansion of a single-particle wavefunction $\psi(r)$. That is, when the cutoff is set to E_{cut}, $\psi(r)$ is expressed as

$$\psi(r) = \sum_{j=1}^{n_g} c_j e^{ig_j^T r}, \tag{2.2}$$

where n_g is the number of all wave numbers g_j that satisfy

$$||g_j||^2 < 2E_{cut}, \tag{2.3}$$

The frequency cutoff E_{cut} also allows us to determine the minimum number of sampling grid points (n_1, n_2, n_3) we need to represent $\psi(r)$ in real space without causing aliasing effects [31].

Clearly, the larger the E_{cut} value we use, the larger the number of planewave basis functions and real space sampling grid points are required in the representations of $\psi(r)$. The use of a large number of planewave basis leads to a more accurate finite dimensional approximation to the continuous Kohn-Sham equations. Unfortunately, this also leads to a high computational cost (in term of both time and storage) required to solve the finite dimensional Kohn-Sham equations.

The optimal choice of E_{cut} is system dependent. The choice also depends on the pseudopotential function used. However, for the purpose of understanding the numerical method for solving a finite dimensional KS equation, it is sufficient to set E_{cut} to a relatively small

number, e.g., $E_{cut} = 25Ryd$. One can experimenting with different choices of E_{cut} to see how the minimum total energy changes as one increase the number of plane wave basis in the discretization.

```
%
% 1. construct atoms
%
a1 = Atom('Si');
a2 = Atom('H');
alist = [a1; a2; a2; a2; a2];
%
% 2. set up primitive cell (Bravais lattice)
%
C = 10*eye(3);
%
% 3. define the coordinates the atoms
%
coefs = [
 0.0     0.0      0.0
 0.161   0.161    0.161
-0.161  -0.161    0.161
 0.161  -0.161   -0.161
-0.161   0.161   -0.161
];
xyzmat = coefs*C';
%
% 4. Configure the molecule
%
mol = Molecule();
mol = set(mol,'Blattice',C);
mol = set(mol,'atomlist',alist);
mol = set(mol,'xyzlist' ,xyzmat);
mol = set(mol,'ecut', 25);  % kinetic energy cut off
mol = set(mol,'name','SiH4');
```

Figure 2.2: Setting up the *Silane* molecule

2.4.5 Checking the Attributes of an `Molecule` Object

After all attributes of an `Molecule` object `mol` have been set, one can view all attributes by simply typing `mol` (without semicolon) at the MATLAB prompt without a semicolon at the end. For example, typing `mol` after running `sih4_setup` gives the output shown in Figure 2.3.

Note that once the list of atoms is specified, the `Molecule` object automatically calculates the number of valence electrons contained in the molecular system. In the case of the SiH_4 molecule, there are 8 valence electrons. Four of them are contributed by the Si atom in the

```
>> mol
Molecule: SiH4
   primitive cell:
      1.000e+01     0.000e+00     0.000e+00
      0.000e+00     1.000e+01     0.000e+00
      0.000e+00     0.000e+00     1.000e+01
   sampling size: n1 = 32, n2 = 32, n3 = 32
   atoms and coordinates:
       1   Si      0.000e+00      0.000e+00      0.000e+00
       2    H      1.610e+00      1.610e+00      1.610e+00
       3    H     -1.610e+00     -1.610e+00      1.610e+00
       4    H      1.610e+00     -1.610e+00     -1.610e+00
       5    H     -1.610e+00      1.610e+00     -1.610e+00
   number of electrons  : 8
   spin type            : 1
   kinetic energy cutoff: 2.500e+01
```

Figure 2.3: Displaying the attributes of the Silane molecule

2s and 2p atomic orbitals. The four hydrogen atoms each contributes an electron. Currently, KSSOLV does not distinguish the spin orientations of valence electrons in molecules or solids. This is indicated by the `spin type` attribute in the `Molecule` object, which is set to 1 in Figure 2.3. What that means is that for this particular system, electrons are considered in pairs, and the number of occupied states (Kohn-Sham wavefunctions) is half of the number of valence electrons.

For systems that do not contain atoms such as electron quantum dots that contain electrons confined by an external potential. The number of electrons `nel` must be specified explicitly through a statement like

```
mol = set(mol,'nel',4);
```

In this case, one may also set the `spin type` to 2 by

```
mol = set(mol,'nspin',2)
```

so that electrons will not be considered in pairs.

2.5 Solving the Kohn-Sham Equations

Once an `Molecule` object is set up properly, computing the electron wavefunctions associated with the minimum total energy of the molecular system can be performed in KSSOLV by calling one of the functions listed in Table 2.2. We will describe these methods in detail in chapter 4.

The simplest way to use the `scf` function which contains an implementation of the self-consistent field (SCF) iteration with charge mixing [22, 23, 11] acceleration is:

```
>> [Etotvec, X, vtot, rho] = scf(mol);
```

function	Description
`scf.m`	Self consistent field iteration
`chebyscf.m`	Chebyshev filtered self consistent field iteration
`dcm.m`	direct constrained minimization (DCM)
`trdcm.m`	direct constrained minimization with flexible trust region parameters
`trdcm1.m`	direct constrained minimization with a fixed trust region parameter

Table 2.2: Methods developed for Solving the KS equation in KSSOLV

The function call takes `mol` as its only input and produces the following outputs:

1. `Etotvec` is an array of total energies computed at each SCF iteration;
2. `X` is a `Wavefun` object that contains the approximate single-particle wavefunctions returned from the SCF calculation. The details on the structure of the `Wavefun` class are explained in Chapter 3;
3. `vtot` is a 3D array that contains the total potential associated with the returned single-particle wavefunctions of the `mol` object.
4. `rho` is a 3D array that contains the electron charge associate with the returned single-particle wavefunctions of the `mol` object.

All functions listed in 2.2 allow additional options to be passed in through an option structure that can be defined by calling the function `setksopt`.

Calling `setksopt` without any argument simply creates an option structure with default parameter settings. For example,

```
opts = setksopt;
```

creates an option structure `opt` that contains the following fields

```
opts =

       verbose: 'on'
    maxscfiter: 10
    maxdcmiter: 10
    maxinerscf: 3
     maxcgiter: 10
        scftol: 1.0000e-08
        dcmtol: 1.0000e-08
         cgtol: 1.0000e-06
       mixtype: 'anderson'
        mixdim: 9
       betamix: 0.8000
         brank: 1
            X0: []
          rho0: []
```

Each field can be modified by simply resetting the value of the field through an assignment statement. For example, to change the maximum number of SCF iterations allowed, we can simply use

```
opts.maxscfiter = 20.
```

This parameter can also be modified by calling `setksopt` again with appropriate argument pairs. For example,

```
opts = setksopt(opts,'maxscfiter',20);
```

achieves the same effect. The advantage of calling `setksopt` is that the function checks the validity of the structure field and its value.

More information about the options and default values for all methods listed in Figure 2.2 can be found in Chapter 4.

2.6 Understanding the Output from KSSOLV

By default, both the SCF and the DCM methods implemented in KSSOLV print out a bunch of information on the screen. A typical SCF run produces diagnostic output like the one shown in Figure 2.4. After some initialization, the code uses the LOBPCG algorithm [13] to be described later in section 4 to solve a sequence of linear eigenvalue problem. In the case of the silane molecule (SiH_4), there are 4 electron pairs. Thus 4 smallest eigenvalues of a fixed Hamiltonian are approximated at each SCF iteration. The diagnostic output shows both the eigenvalue approximation and the estimated error associated with these eigenvalues and their corresponding eigenvectors. At the end of each SCF iteration, the code also prints out the total energy evaluated at the current wavefunction approximation, the residual error associated with eigenvalue and wavefunction approximations (to the solution of the Kohn-Sham equations (1.1)), and the lack of self-consistency in the total potential.

2.7 Visualizing the Results

The electron charge density, total potential and single particle wavefunctions can be easily visualized using either the MATLAB's isosurface rendering function `isosurface` or the volume rendering function written by Joe Conti, which we also include in KSSOLV. To see the isosurface of the charge density `rho`, one can simply type

```
>> isosurface(rho).
```

The `scf`, `dcm` and `trdcm` functions in KSSOLV return a charge density defined within the specified unit supercell even though the atomic coordinates defined during the initial setup may not lie within that cell. To see the a charge density in the original atomic coordinates, one may use the MATLAB `fftshift` function first to shift density before rendering. That is, one may use

```
>> isosurface(fftshift(rho))
```

to view the isosurface of the charge density.

Figure 2.5 shows the isosurface rendering of the charge density of the SiH_4 molecule associated with the atomic positions defined in Figure 2.2 and displayed in Figure 2.1(a). Similarly, one can use

```
[Etotvec, X, vtot, rho] = scf(mol);
initialization...
Beging SCF calculation...
LOBPCG iter =   1
eigval( 1) =   7.195e+00, resnrm =   3.277e+00
eigval( 2) =   7.368e+00, resnrm =   3.347e+00
eigval( 3) =   7.551e+00, resnrm =   3.233e+00
eigval( 4) =   7.703e+00, resnrm =   3.239e+00

LOBPCG iter =   2
eigval( 1) =   5.097e-01, resnrm =   8.297e-01
eigval( 2) =   6.200e-01, resnrm =   8.719e-01
eigval( 3) =   6.912e-01, resnrm =   9.045e-01
eigval( 4) =   9.038e-01, resnrm =   9.130e-01
...
SCF iter  1:
norm(vout-vin) =  5.807e+00
Total energy   = -5.8719608972592e+00
 resnrm =   1.857e-02
 resnrm =   2.161e-02
 resnrm =   2.161e-02
 resnrm =   2.162e-02
...
```

Figure 2.4: SCF output

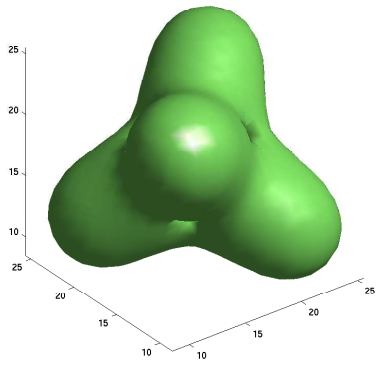


Figure 2.5: The computed charge density of the SiH_4 molecule.

```
>> vol3d(fftshift(rho));
```

to view a volume rendering of `rho`.

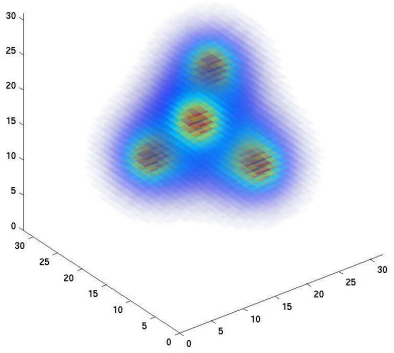


Figure 2.6: A volume rendering of the computed charge density of the SiH_4 molecule.

Chapter 3

KSSOLV Classes

In this chapter, we will describe all classes defined in KSSOLV in detail. In particular, we will examine the structure of these classes and methods that can be invoked. We will also discuss overloaded operators that can be used directly on objects that belong to certain classes.

In MATLAB, a user-defined class is identified by a directory whose name begins with the '@' symbol. Therefore, once KSSOLV is installed, one should find the following directories in KSSOLV's root directory `kssolv/`:

```
@Atom/
@Molecule/
@Ham/
@Wavefun/
@FreqMask/
```

Each @-prefixed directory contains an M-file with the same name (excluding the @ symbol). This M-file is a *constructor* that can be used to instantiate a specific KSSOLV object that belongs to a certain class. The function defined in this M-file can be called without any input argument to create a null object that needs to be refined later by calling the `set` method. For example, the following call creates a null `Atom`object.

```
a = Atom();
```

One may also call the constructor with one or a few arguments to initialize the object when it is created. For example, the following call creates an Hamiltonian associated with a predefined `Molecule` object `mol` using a predefined charge density `rho`.

```
H = Ham(mol,rho);
```

Each class contains a number of attributes specific to that class. These attributes must be defined properly before an object belonging to that class is used. When these attributes are not initialized upon the initial construction, they can be set by using the `set` method defined for that class. For example, if an `Atom` object `a` is created by calling the constructor with no input argument (as illustrated above), we can use

```
a = set(a,'symbol','H');
```

to specify that `a` is an hydrogen atom. Here the string 'symbol' is a keyword that represents the name of a specific attribute, which in this case refers to the chemical symbol of an atom. The user provided string 'H' is a value assigned to that attribute.

Note that not all attributes of an object need to be set before the object can be used in subsequent calculations. Setting some key attributes will ensure that other attributes of the same object are properly set automatically. In fact, for an `Atom` object, the only attribute that needs to be set is either the atomic symbol 'symbol', or the atomic number 'anum'. All other attributes of the atom, which we will examine below, will be defined automatically once 'anum' or 'symbol' is set.

All attributes of an object can be retrieved by using the `get` method defined for the class to which the object belongs. For example,

```
nvel = get(a,'vnum');
```

retrieves the number of valence electrons in atom `a`, and assigns that number to the variable `nvel`.

The essential attributes of an object can also be displayed by typing the name of object at the MATLAB prompt without a semicolon. For example, typing

```
>> a = Atom('Si')
```

shows

```
atomic symbol: Si
atomic number: 14
number of valence electrons: 4
```

3.1 Atom

Atoms are the basic constituents of molecules and solids. Therefore, it is natural to define an `Atom` class in KSSOLV so that we can easily create atom objects and use them to construct molecules and bulk systems. An atom can be declared in KSSOLV as an `Atom` object using an `Atom` constructor. For example

```
a = Atom('Si');
```

or

```
a = Atom(14);
```

defines a silicon atom object named `a`. Note that an atom can be defined by either its chemical symbol or by its atomic number. An atom object contains a number of attributes listed and described in Table 3.1. Most of these attributes are properly set once the chemical symbol or the atomic number of the atom is initialized. They are mainly used by KSSOLV internally to construct ionic pseudopotentials.

All these attributes are accessible through the `get` method defined for the `Atom` class. However, a user would rarely need to retrieve attributes other than the atomic number and the number of valence electrons.

attribute name	description	data type
'symbol'	chemical symbol	string
'anum'	atomic number	integer
'venum'	number of valence electrons	integer

Table 3.1: Attributes of an `Atom` object

3.2 Molecule

The keyword `Molecule` is used in KSSOLV broadly to refer to both a single molecule and a periodic (bulk) system. The `Molecule` class is created to encapsulate both the physical attributes of a molecule and the planewave discretization parameters that must be defined before the Kohn-Sham equations associated with a particular `Molecule` object can be solved.

To create an `Molecule` object, one typically uses

```
mol = Molecule();
```

to first construct an empty object called `mol` (a user defined variable name). This call simply sets up the required data structure that would be used to describe various attributes of `mol`. The actual attributes of `mol` such as the number and type of atoms in this molecule, the size and shape of the supercell that contains the molecule etc. can be defined through the `set` method.

```
mol = set(mol,attrname,attrvalue);
```

where the input argument `attrname` is a predefined string associated with the `Molecule` class that specifies the name of a particular attribute, and `attrvalue` is a user supplied quantity that will be stored in `mol`. Table 3.2 lists the all attributes of an `Molecule` object. Some of these attributes such as the name of the molecule and the external potential are optional. Some attributes such as the number of sampling grid points `n1`, `n2`, `n3` will be properly initialized after other attributes such as `ecut` are defined.

The datatypes associated with most of the attributes are straightforward. A few require further explanation. The `'supercell'` attribute defines the shape and size of the unit supercell that contains the basic atomic constituents of a single molecule or solid. In KSSOLV, the super cell is described by a 3×3 matrix. Each row of this matrix defines a lattice vector emanating from the origin. For example,

```
mol = set(mol,'supercell',10*eye(3));
```

sets the unit supercell of `mol` to a $10 \times 10 \times 10$ cube whose three basic lattice vectors are parallel to the x, y and z axes respectively. Atomic units (Bohr) are used in the definition of the unit cell.

The 'atomlist' attribute is defined to be an array of `Atom` objects. An example of how to construct such an array has been shown in Chapter 2. The coordinates associated with n_a atoms listed in 'atomlist' must be defined in an $n_a \times 3$ array that contains the x, y, z coordinates (in atomic units) for each atom.

Currently, KSSOLV makes no distinction between the wavefunctions associated with electrons occupying the same orbitals. In this case, the spin type is normally set to `nspin=1`.

However, KSSOLV can also be used to solve the Kohn-Sham equations for electron-only systems. In this case, electron wavefunctions are not associated with any atoms, and they must be treated differently, which indicated in KSSOLV by setting `nspin` to 2.

attribute name	purpose	value type
'name'	The name of the Molecule	string
'supercell'	Bravais lattice box	3×3 matrix
'atomlist'	list of atoms	array of `Atom` objects
'xyzlist'	list of atomic coordinates	an $n_a \times 3$ matrix
'vol'	The volume of the primary (super)cell that contains the molecule	float
'natoms'	number of atoms in the primary cell	integer
'ecut'	planewave energy cutoff	floating point scalar
'n1'	the number of sampling points along the x-axis	integer
'n2'	the number of sampling points along the y-axis	integer
'n3'	the number of sampling points along the y-axis	integer
'nspin'	the spin type	1 or 2
'nel'	the number of valence electrons	integer
'vext'	external potential	3D array of size $n1 \times n2 \times n3$

Table 3.2: Attributes to be set in an `Molecule` object

3.3 Ham

A properly defined `Molecule` object `mol` can be used to initialize a KS Hamiltonian. The initialization can be done by simply calling the constructor

```
H = Ham(mol);
```

The constructed `Ham` object should contain all information one would need to solve the Kohn-Sham equations. Although the KS Hamiltonian $H(X)$ is treated as a matrix in equation (4.12), it is not stored as a matrix in KSSOLV. Instead, the `Ham` class keeps L, $\hat{V}_{ion}$ and total potential $V_{tot} = \hat{V}_{ion} + \mathrm{Diag}(L^\dagger \rho) + \mathrm{Diag}(\mu_{xc}(\rho))$ as separate attributes. This separation makes it easy to update the Hamiltonian in both the SCF and DCM calculations. Table 3.3 shows all other important attributes of an `Ham` object.

attribute name	purpose	value type
'gkk'	The magnitude square of wave numbers associated with the kinetic energy that satisfy the cutoff constraint (2.3)	1D array
'idxnz'	positions of the reciprocal lattice points associated with all entries of `gkk` within a 3D array	1D array
'vtot'	total local potential	3D array
'vion'	local ionic pseudopotential in real space	3D array
'wqmat'	Fourier representation of the non-local pseudopotential	2D array (matrix)
'vext'	external potential	3D array
'vnp'	charge dependent portion of the potential	3D array

Table 3.3: Attributes to be set in an `Molecule` object

The kinetic component of a `Ham` object is represented and stored as a 1D array (`gkk`) of integers that represent the magnitude squares of the wave numbers that satisfy the energy cutoff constraint (2.3). These values correspond to the eigenvalues of a Laplacian operator

(with periodic boundary conditions) sampled in the reciprocal (frequency) space. Only wave numbers (or frequencies) that satisfy the kinetic energy cutoff set in `mol` are stored. There numbers correspond to reciprocal space lattice points that lie within a sphere with a radius of $\sqrt{2E_{cut}}$. The locations of these lattice points in a 3D array of all lattice points (including those outside of the cutoff sphere) are kept in a separate array `idxnz` so that wave numbers associated with reciprocal space lattice points outside of the kinetic energy cutoff sphere are not used or stored.

The local ionic potential $\hat{V}_{ion}$ is also constructed and stored as a 3D array at initialization. The construction process involves loading the atomic pseudopotential stored in the directory `Pseudopot` and summing them up using the scheme suggested by Kleinman and Bylander [12]. KSSOLV provides Troullier-Martins [27] atomic pseudopotentials associated with all atoms listed in the first four rows of the periodic table. Some elements in higher row of the table are also provided.

The determination of both the Hartree and exchange correlation potential requires the availability of the charge density ρ which is in turn a function of the wavefunctions to be computed. Since a good approximation to the desired wavefunctions is not available at initiation, the initial ρ is computed in KSSOLV by combining atomic charge densities associated with each atom in the `mol` object [12].

Once a `Ham` object has been defined, one can retrieve various attributes of the object through the `get` function, e.g.,

```
vt = get(H,'vtot');
```

returns the total potential from a `Ham` object named `H` and assigns it to a user defined variable `vt`. This potential can be used and updated in a subsequent SCF or DCM calculation. An updated `vt` can be passed into `H` by using the `set` function

```
H = set(H,'vtot',vt);
```

3.4 Wavefun

We created a special class `Wavefun` in KSSOLV to specify one or a set of wavefunctions rather than representing them by a MATLAB vector or matrix. The rationale for creating such a class is mainly to reduce the amount of codes a user would have to write to reshape a 3D array (which is the data structure required for a 3D FFT call) into a 1D vector (which is the most convenient data structure for linear algebra operations) and vice versa. Representing a wavefunction as a `Wavefun` object also makes it easy to perform the multiplication of a KS Hamiltonian with a wavefunction and basic linear algebra operations between wavefunctions through operator overloading.

In KSSOLV, a `Wavefun` object stores the Fourier coefficients of one or several wavefunctions that are expanded in a planewave basis defined by (2.2). The object can be constructed using either a noncompact scheme or a compact scheme. In a noncompact representation, a `Wavefun` object `X` can be constructed through the command

```
X = Wavefun(psi),
```

where `psi` is a MATLAB 3D array if `X` represents a single wavefunction, or a cell array that contains a list of 3D arrays if `X` represents a set of wavefunctions.

Under the compact scheme, a `Wavefun` object stores only the Fourier coefficients ($c_{i,j}$) of a truncated expansion of a single wavefunction $\psi(r)$ or a set of wavefunctions $\{\psi_i(r)\}_{i=1}^{n_e}$. These coefficients are associated with wave numbers (g_j) that satisfy the cutoff constraint (2.3), and stored contiguously as a MATLAB cell array of size n_g by `nocc`, where n_g is the number of these Fourier expansion coefficients, and `nocc` is the number of occupied states (electron pairs). All other Fourier coefficients that may appear in a full planewave expansion are considered zeros, thus never used or stored. Storing the nonzero Fourier expansion coefficients contiguously in a 1D array makes linear algebra operations on `X` more efficient. However, to perform 3D FFT operations on `X`, we must place these coefficients in a 3D array. Hence, in the compact scheme, we must also record the locations of these Fourier coefficients in a 3D array, which are the locations of the reciprocal lattice points associated with the corresponding wave numbers in the planewave basis. These locations are stored in a separate array which is labeled as the '`idxnz`' attribute of the `Wavefun` object. Recall that the same strategy is used to store the magnitude squares of the wave numbers and their locations in a 3D array for the `Ham` object. In signal processing, functions that are expanded by planewaves up to certain frequency or wave number are called *bandlimited* functions.

The size of the 3D array that is used to hold the Fourier coefficients when an FFT is performed is specified by the '`n1`', '`n2`' and '`n3`' attributes of the `Wavefun` object. These numbers also give the dimension of the corresponding wavefunction in real space. They should be the same as those specified in a `mol` object.

The constructor for a `Wavefun` object can be called in a number of ways. In addition to creating a null object by using,

```
X = Wavefun();
```

we may also create a null object of certain size by using

```
X = Wavefun(n1,n2,n3);
```

where `n1`, `n2` and `n3` are predefined integers. The following use of the constructor creates a `Wavefun` object using a noncompact scheme.

```
X = Wavefun(Q);
```

where `Q` is either a 3D array or a cell array that contains a number of 3D arrays. These 3D arrays should contain the Fourier expansion coefficients of one or several wavefunctions.

Finally, a fixed size Wavefun object in compact scheme can be constructed by using

```
X = Wavefun(Q,n1,n2,n3,idxnz);
```

where Q is a 1D array that stores the Fourier expansion coefficients continuously and `idxnz` is a 1D array that describes the locations of these coefficients in a 3D array of size $n1 \times n2 \times n3$.

In the following section, we will see that all major matrix operations have been overloaded for `Wavefun` objects for both the compact and noncompact schemes. KSSOLV also provides a utility function `genX0` that allows one to easily construct initial `Wavefun` objects for the SCF or DCM calculations. To generate a set of random bandlimited wavefunctions using the kinetic energy cutoff specified in a `Molecule` object `mol`, one can simply use the command

```
X = genX0(mol).
```

Converting a `Wavefun` object `X` to a 3D array (or a list of 3D arrays) is straightforward when `X` is constructed using a noncompact scheme. The following command

```
X3D = get(X,'psi')
```

returns the wavefunctions as a cell array `X3D` of 3D arrays. Although rarely needed when using KSSOLV, the following lines of codes show how the same conversion can be accomplished for an `X` constructed using a compact representation scheme

```
n1  = get(X,'n1');
n2  = get(X,'n2');
n3  = get(X,'n3');
psi = get(X,'psi');
idx = get(X,'idxnz');
X3D = zeros(n1,n2,n3);
X3D(idx) = psi{1}.
```

3.5 FreqMask

The `FreqMask` class is created to define an object that stores the wave numbers (frequencies) that appear in (2.2) and their magnitudes. The three components of each wave number g_j are stored in three separate arrays `gkx`, `gky` and `gkz`. These arrays are attributes of a `FreqMask`. They are used in the pseudopotential calculation (`getvion.m`, `getwq.m`) and a number of other KSSOLV functions. Only the wave numbers that satisfy the energy cutoff criterion (2.3) are defined and stored when a `FreqMask` object is constructed. The number of wave numbers that satisfy (2.3) is stored as the attribute `ng`.

A `FreqMask` object can be constructed by calling

```
gm = FreqMask(mol).
```

In this case, the energy cutoff defined in the `Molecule` object `mol` is used to determine which wave numbers should be set in `gkx`, `gky` and `gkz`. The constructor of the object also calculates and stores the magnitude square of each Fourier expansion coefficient that satisfies (2.3) in an separate array `gkk`. Because such a quantity is used frequently in the numerical procedure for solving the Kohn-Sham equations we store it in a `FreqMask` object instead of computing it repeatedly from `gkx`, `gky` and `gkz` each time it is used.

Alternatively, one can pass in a different energy cutoff and create a `FreqMask` object by using

```
gm = FreqMask(mol,ec),
```

where `ec` is a user provide energy cutoff value.

3.6 Operator Overloading

Because the Kohn-Sham Hamiltonian $H(X)$ and wavefunction X are viewed as matrices in the algorithms to be described in the next chapter, it is desirable to allow `Hamilt` and `Wavefun` objects to be manipulated in KSSOLV as if they are matrices. This feature is made possible in KSSOLV by overloading some basic algebraic operations for a `Wavefun` object. These

overloaded operations are listed in Table 3.4. One should be careful about the use of some of these operators. For example, since the wavefunctions used in the SCF and DCM calculation all have the same dimension, the multiplication operator `*` is never used between two `Wavefun` objects except when the first `Wavefun` object is transposed or conjugate transposed, i.e., it is valid to perform `x'*y` or `x.'*y`, and the multiplication returns a standard MATLAB matrix object. The overloaded multiplication operator `*` for `Wavefun` objects allows the second operand to be a standard matrix object with proper dimension. The result of the multiplication is a `Wavefun` object.

Operations	Description
`x + y`	Add two wavefunctions
`x - y`	Subtract one wavefunction from another
`x * y`	Multiply two wavefunctions and return a matrix
`x * a`	Multiply several wavefunctions with a matrix
`x .* y`	Element-wise multiplication of two wavefunctions
`x . y`	Element-wise division of two wavefunctions
`x'`	Complex conjugate transpose of a wavefunction
`x.'`	Transpose of a wavefunction
`[x y]`	Horizontal concatenation of several wavefunctions
`x(:,i:j)`	Subscripted reference of wavefunctions

Table 3.4: Overload operations for `Wavefun` objects in KSSOLV

The multiplication operator is also overloaded for the `Hamilt` class so that the multiplication of a `Hamilt` object `H` and a `Wavefun` object `X` can be accomplished in KSSOLV by a simple expression

```
Y = H*X,
```

which hides all the complexity of the operation from the user.

Chapter 4

KSSOLV Methods

In this chapter, we will briefly describe the numerical methods employed in KSSOLV to obtain an approximate solution to the Kohn-Sham equations (1.1). We begin by discussing the planewave discretization scheme that turns the continuous nonlinear problem into a finite dimensional problem. The finite dimensional problem is expressed as a matrix nonlinear eigenvalue problem. We present two different approaches to solving the matrix nonlinear eigenvalue problem in sections 4.4 and 4.5.

4.1 Planewave Discretization

To solve the minimization problem (1.4) or the Kohn-Sham equation (1.1) numerically, we must first discretize the continuous problem. Standard discretization schemes such as finite difference, finite elements and other basis expansion (Ritz-Galerkin) methods [25] all have been used in practice. The discretization scheme we have implemented in the current version of KSSOLV is a Ritz type of method that expresses a single electron wavefunction $\psi(r)$ as a linear combination of planewaves $\{e^{-ig_j^T r}\}$, where $g_j \in \mathbb{R}^3$ $(j = 1, 2, ..., K)$ are wave numbers (frequency vectors) arranged in a lexicographical order. The planewave basis is a natural choice for studying periodic systems such as solids. It can also be applied to non-periodic structures (e.g., molecules) by embedding these structures in a ficticious supercell [18] that is periodically extended throughout an open domain. The use of the planewave basis has the additional advantage of making various energy calculations in density functional theory easy to implement. It is the most convenient choice for developing and testing numerical algorithms for solving the Kohn-Sham equations within the MATLAB environment, partly due to the availability of efficient fast Fourier transform (FFT) functions.

It is natural to assume that the potential for R-periodic atomistic systems is a periodic function with a period $R \equiv (R_1, R_2, R_3)$. Consequently, we can restrict ourselves to one canonical period often referred to as the primitive cell and impose periodic boundary condition on the restricted problem. It follows from the *Bloch*'s theorem [3, 4] that eigenfunctions of the restricted problem $\psi(r)$ can be periodically extended to the entire domain (to form the eigenfunction of the original Hamiltonian) by using the following formula:

$$\psi(r + R) = e^{ik^T R}\psi(r), \tag{4.1}$$

where $k = (k_1, k_2, k_3)$ is a frequency or wave vector that belongs to a primitive cell in the reciprocal space (e.g., the first *Brillouin* zone [3]). If the R-periodic system spans the entire

infinite open domain, the set of k's allowed in (4.1) forms a continuum in the first Brillouin zone. That is, each $\psi(r)$ generates an infinite number of eigenfunctions for the periodic structure. It can be shown that the corresponding eigenvalues form a continuous cluster in the spectrum of the original Hamiltonian [3]. Such a cluster is often referred to as an energy band in physics. Consequently, the complete set of eigenvectors of H can be indexed by the band number i and the Brillouin frequency vector k (often referred to as a k-point), i.e., $\psi_{i,k}$. In this case, the evaluation of the charge density must first be performed at each k-point by replacing $\psi_i(r)$ in (1.2) with $\psi_{i,k}$ to yield

$$\rho_k = \sum_{i=1}^{n_e} |\psi_{i,k}|^2.$$

The total charge density $\rho(r)$ can then be obtained by integrating over k, i.e.,

$$\rho(r) = \frac{|\Omega|}{(2\pi)^3} \int_{BZ} \rho_k(r) dk, \tag{4.2}$$

where $|\Omega|$ denotes the volume of the primitive cell in the first Brillouin zone. Furthermore, an integration with respect to k must also be performed for the kinetic energy term in (1.5).

When the primitive cell (or supercell) in real space is sufficiently large, the first Brillouin zone becomes so small that the integration with respect to k can be approximated by a single k-point calculation in (4.2).

To simplify our exposition, we will, from this point on, assume that a large primitive cell is chosen in the real space so that no integration with respect to k is necessary. Hence we will drop the index k in the following discussion and use $\psi(r)$ to represent an R-periodic single particle wavefunction. The periodic nature of $\psi(r)$ implies that it can be represented (under some mild assumptions) by a Fourier series, i.e.,

$$\psi(r) = \sum_{j=-\infty}^{\infty} c_j e^{ig_j^T r}, \tag{4.3}$$

where c_j is a Fourier coefficient that can be computed from

$$c_j = \int_{-R/2}^{R/2} \psi(r) e^{-ig_j^T r} dr.$$

In practice, the exact number of terms used in (4.3) is determined by a kinetic energy cutoff E_{cut}. Such a cutoff yields an approximation

$$\psi(r) = \sum_{j=1}^{n_g} c_j e^{ig_j^T r}, \tag{4.4}$$

where n_g is chosen such that

$$||g_j||^2 < 2E_{cut}, \tag{4.5}$$

for all $j = 1, 2, \ldots, n_g$. Although, the value of n_g will depend on many parameters such as the size and type of the system being studied, it is typically an order of magnitude smaller than $n = n_1 \times n_2 \times n_3$.

Once E_{cut} is chosen, the minimal number of samples of r along each Cartesian coordinate direction (n_1, n_2, n_3) required to represent $\psi(r)$ (without the aliasing effect) can be determined

from the sampling theorem [17]. That is, we must choose n_k ($k = 1, 2, 3$) sufficiently large so that

$$\frac{1}{2}\left(\frac{2\pi n_k}{R_k}\right) > 2\sqrt{2E_{cut}}, \tag{4.6}$$

is satisfied, i.e., n_k must satisfy $n_k > 2R_k\sqrt{2E_{cut}}/\pi$.

We will denote the uniformly sampled $\psi(r)$ by a vector $x \in \mathbb{R}^n$, where $n = n_1 n_2 n_3$ and the Fourier coefficients c_j in (4.4) by a vector $c \in \mathbb{C}^n$ with zero paddings used to ensure the length of c matches that of x. If the elements of x and c are ordered properly, these two vectors satisfy

$$c = Fx. \tag{4.7}$$

where $F \in \mathbb{C}^{n\times n}$ is a discrete Fourier transform matrix [28].

After a sampling grid has been properly defined, the approximation to the total energy can be evaluated by replacing the integrals in (1.5) with simple summations over the sampling grid.

4.2 Pseudopotentials

To solve the Kohn-Sham equations numerically, the Fourier series expansion (4.3) must be truncated to allow a finite number of terms only. If all electrons are treated equally, the number of terms required in (4.3) will be extremely large. This is due to the observation that the strong interaction between a nucleus and the inner electrons of an atom, which can be attributed to the presence of singularity in $V_{ion}(r)$ at the the nuclei position $\hat{r}_j$, must be accounted for by high frequency planewaves. However, because the inner electrons are held tightly to the nuclei, they are not active in terms of chemical reactions, and they usually do not contribute to chemical bonding or other types of interaction among different atoms. On the other hand, the valence electrons (electrons in atomic orbits that are not completely filled) can be represented by a relatively small number of low frequency planewaves. These electrons are the most interesting ones to study because they are responsible for a majority of the physical properties of the atomistic system. Hence, it is natural to focus only on these valence electrons and treat the inner electrons as part of an ionic core. An approximation scheme that formalizes this approach is called the *pseudo-potential* approximation [19, 20, 34]. The details of pseudopotential construction and their theoretical properties are beyond the scope of this user's guide. The user should just keep in mind that the use of pseudo-potentials allows us to

1. remove the singularity in V_{ion};
2. reduce the number of electrons n_e in (1.5) and (1.2) to the number of valence electrons;
3. represent the wavefunction associated with a valence electron by a small number of low frequency planewaves.

In KSSOLV, the ionic pseudopotential experienced by each single particle wavefunction is constructed from atomic pseudopotentials that have been previously constructed and stored in the `Pseudopot` directory. After a `Molecule` object `mol` has been properly created, the user can simply call

```
pseudovar = pseudoinit(mol)
```

which returns a MATLAB structure `pseudovar` (user defined variable name). The `pseudovar` contains various pieces of information required by KSSOLV to construct the ionic pseudopotential. In most cases, a user does not need know what is stored in `pseudovar`.

The ionic pseudopotential consists of two components. One of them is a local component $V_{local}(r)$ that operates pointwise on a single particle wavefunction $\psi(r)$. The other component is nonlocal. It is a projection operator that can be represented as

$$V_{nonlocal}(r) = \frac{1}{\Delta}\sum_{i=1}^{\ell} w_i(r)w_i(r)^*,$$

where Δ is a normalization factor, and the rank of the projection operator ℓ is the total number of quantum number triplets (n, l, m) associated with all atoms in the system, where n is the major electron shell number of and (l, m) represent a angular momentum.

In KSSOLV, the local ionic potential can be obtained by calling function `getvion`, i.e.

```
vlocal = getvion(pseudovar),
```

where `vlocal` is a 3D array that contains the potential define at each real space sampling grid points.

The nonlocal potential can be computed by calling the function `getwq`, i.e.

```
wqmat = getwq(pseudovar),
```

where `wq` is a n_g by ℓ matrix and n_g is the number of planewave basis used to represent each single particle wavefunction.

Both `vlocal` and `wqmat` are computed when a `Ham` object is constructed initially using

```
H = Ham(mol);
```

for some properly defined `Molecule` object `mol`. These arrays are saved as different attributes of `H`.

4.3 The finite dimensional Kohn-Sham problem

If we let $X \equiv (x_1, x_2, ..., x_{n_e}) \in \mathbb{C}^{n\times n_e}$ be a matrix that contains n_e discretized wavefunctions, the approximation to the kinetic energy (1.5) can also be expressed by

$$\hat{E}_{kin} = \frac{1}{2}\text{trace}(X^*LX), \tag{4.8}$$

where L is a finite-dimensional representation of the Laplacian operator in the planewave basis. Due to the periodic boundary condition imposed in our problem, L is a block circulant matrix with circulant blocks that can be decomposed as

$$L = F^*D_gF, \tag{4.9}$$

where F is the discrete Fourier transform matrix used in (4.7), and D_g is a diagonal matrix with $||g_j||^2$ on the diagonal [5]. It follows from (4.7) and (4.9) that the kinetic energy term (4.8 can be evaluated alternatively as

$$\frac{1}{2}\sum_{\ell=1}^{n_e}\sum_{j=1}^{n_g}||g_jc_j^{(\ell)}||^2.$$

In the planewave basis, the convolution that appears in the third term of (1.5) may be viewed as the $L^{-1}\rho(X)$, where $\rho(X) = \mathrm{diag}(XX^*)$. (To simplify notation, we will drop X in $\rho(X)$ in the following.) However, since L is singular (due to the periodic boundary condition), its inverse does not exist. Similar singularity issues appear in the planewave representation of the pseudopotential and the calculation of the ion-ion interaction energy. However, it can be shown that the net effects of these singularities cancel out for a system that is electrically neutral [10, 21]. Thus, one can simply remove these singularities by replacing $L^{-1}\rho$ with $L^{\dagger}\rho$, where $L^{\dagger}$ is the pseudo-inverse of L defined as

$$L^{\dagger} = F^* D_g^{\dagger} F,$$

where $D_g^{\dagger}$ is a diagonal matrix whose diagonal entries (d_j) are

$$d_j = \begin{cases} ||g_j||^{-2} & \text{if } g_j \neq 0; \\ 0 & \text{otherwise.} \end{cases}$$

Consequently, the third term in (1.5), which corresponds to an approximation to the Coulomb potential, can be evaluated as

$$\hat{E}_{coul} = \rho^T L^{\dagger} \rho = [F\rho]^* D_g^{\dagger} [F\rho].$$

However, removing these singularities results in a constant shift of the total energy, for which a compensation must be made. It has been shown in [10] that this compensation can be calculated by adding a term E_{rep} that measures the degree of repulsiveness of the local pseudopotential with a term that corresponds to the nonsingular part of ion-ion potential energy. Because the second term can be evaluated efficiently by using a technique originally developed by Ewald [6], it is denoted by E_{Ewald}. Both E_{rep} and E_{Ewald} can be computed once and for all in a DFT calculation. We will not go into further details of how they are computed since they do not play any role in the algorithms we will examine in this section. In KSSOLV, the E_{Ewald} calculation is performed in the function `getewald.m` and E_{rep} is calculated in the function `getealpht.m` Both functions are call once at the beginning of `scf.m`, `dcm.m`, `trdcm.m` etc.

To summarize, the use of a planewave basis allows us to define a finite-dimensional approximation to the total energy functional (1.5) as

$$\hat{E}_{total}(X) = \mathrm{trace}[X^*(\frac{1}{2}L + \hat{V}_{ion})X] + \frac{1}{2}\rho^T L^{\dagger}\rho + \rho^T \epsilon_{xc}(\rho) + E_{\mathrm{Ewald}} + E_{\mathrm{rep}}, \tag{4.10}$$

where $\hat{V}_{ion}$ denotes the ionic pseudopotentials sampled on the suitably chosen Cartesian grid of size $n_1 \times n_2 \times n_3$.

It is easy to verify that the KKT condition associated with the constrained minimization problem

$$\min_{X^*X=I} \hat{E}_{total}(X) \tag{4.11}$$

is

$$\begin{aligned} H(X)X - X\Lambda_{n_e} &= 0, \\ X^*X &= I, \end{aligned} \tag{4.12}$$

where

$$H(X) = L + \hat{V}_{ion} + \mathrm{Diag}(L^{\dagger}\rho) + \mathrm{Diag}(\mu_{xc}(\rho)), \tag{4.13}$$

$\mu_{xc}(\rho) = d\epsilon_{xc}(\rho)/d\rho$, and Λ_{n_e} is a $n_e \times n_e$ symmetric matrix of Lagrangian multipliers. Because $\hat{E}_{total}(X) = \hat{E}_{total}(XQ)$ for any orthogonal matrix $Q \in \mathbb{C}^{n_e \times n_e}$, we can always choose a particular Q such that Λ_{n_e} is diagonal. In this case, Λ_{n_e} contains n_e eigenvalues of $H(X)$. We are interested the n_e smallest eigenvalues and the invariant subspace X associated with these eigenvalues. Because the Hamiltonian matrix H depends on the invariant subspace to be computed, (4.12) is often called a nonlinear eigenvalue problem.

4.4 SCF

Currently, the most widely used algorithm for solving (4.12) is the self-consistent field (SCF) iteration which we outline in Figure 4.1 for completeness.

SCF Iteration

Input: An initial guess of the wavefunction $X^{(0)} \in \mathbb{C}^{n \times n_e}$, pseudopotential;

Output: $X \in \mathbb{C}^{n \times n_e}$ such that $X^*X = I_{n_e}$ and columns of X spans the invariant subspace associated with the smallest n_e eigenvalues of $H(X)$ defined in (4.13).

1. for $i = 1, 2, \ldots$ until convergence
2. Form $H^{(i)} = H(X^{(i-1)})$;
3. Compute $X^{(i)}$ such that $H^{(i)}X^{(i)} = X^{(i)}\Lambda^{(i)}$, and $\Lambda^{(i)}$ contains the n_e smallest eigenvalues of $H^{(i)}$;
4. end for

Figure 4.1: The SCF iteration

In [33], we viewed the SCF iteration as an indirect way to minimize $\hat{E}_{total}$ through the minimization of a sequence of quadratic surrogate functions of the form

$$q(X) = \frac{1}{2}\text{trace}(X^*H^{(i)}X), \tag{4.14}$$

on the manifold $X^*X = I_{n_e}$. This constrained minimization problem is solved in KSSOLV by running a small number of locally optimal preconditioned conjugate gradient (LOBPCG) iterations [13].

Since the surrogate function share the same gradient with $\hat{E}_{total}$ at $X^{(i)}$, i.e.,

$$\nabla \hat{E}_{total}(X)_{|X=X^{(i)}} = H^{(i)}X^{(i)} = \nabla q(X)_{|X=X^{(i)}},$$

moving along a descent direction associated with $q(X)$ is likely to produce a reduction in $\hat{E}_{total}$. However, because gradient information is local, there is no guarantee that the minimizer of $q(X)$, which may be far from $X^{(i)}$, will yield a lower $\hat{E}_{total}$ value. This observation partially explains why SCF often fails to converge. It also suggests at least two ways to improve the convergence of SCF.

4.4.1 Charge mixing

One possible improvement is to replace the simple gradient-matching surrogate $q(X)$ with another quadratic function whose minimizer is more likely to yield a reduction in $\hat{E}_{total}$. In practice, this alternative quadratic function is often constructed by replacing the charge density $\rho^{(i)}$ in (4.13) with a linear combination of m previously computed charge densities, i.e.,

$$\rho_{mix} = \sum_{j=0}^{m-1} \alpha_j \rho^{(i-j)},$$

where $a = (\alpha_0, \alpha_2, ..., \alpha_{i-m+1})$ is chosen to reduce the lack of self consistency between the input charge density used to define the surrogate function and the output charge density obtained from the solution of the trust region subproblem using the definition

$$\rho = \mathrm{diag}(XX^*),$$

where $\mathrm{diag}(A)$ denotes the diagonal of the matrix A.

$$\min_{a^T e=1} \|Ra\|^2 \tag{4.15}$$

where $R = (\Delta\rho^{(i)} \;\; \Delta\rho^{(i-1)} \;\; ... \;\; \Delta\rho^{(m-1)})$ and $\Delta\rho^{(i)} = \rho^{(i)} - \rho^{(i-1)}$. This technique is often called *charge mixing*. The particular mixing scheme defined by the solution to (4.15) is called *Pulay mixing* because it was first proposed by Pulay for Hartree-Fock calculations [22, 23]. (In computational chemistry, Pulay mixing is referred to as the method of *direct inversion of iterative subspace* or simply DIIS). Other mixing scheme include *Kerker mixing* [11], *Thomas-Fermi mixing* [24] and *Broyden mixing* [15]. Charge mixing is often quite effective in practice for improving the convergence SCF even though its convergence property is still not well understood. In some cases, charge mixing may fail also.

KSSOLV supports a number of charge mixing schemes. The default choice is the Anderson mixing scheme originally developed in [1] and further analyzed in [7, 8]. To change the default choice of charge mixing, one must first construct an option structure by using

```
option = setksopt;
```

This option structure can be modified by using, for example,

```
option = setksopt('mixtype','broyden');
```

which replace the default charge mixing scheme by a Broyden mixing scheme [16].

Need to write more about charge mixing

4.4.2 Trust Region

Another way to improve the convergence of the SCF iteration is to impose an additional constraint to the surrogate minimization problem (4.14) so that the wavefunction update can be restricted within a small neighborhood of the gradient matching point $X^{(i)}$, thereby ensuring a reduction of the total energy function as we minimize the surrogate function. In [33], we showed that the following constraint

$$\|XX^* - X^{(i)}X^{(i)*}\|_F^2 \leq \Delta$$

is preferred because it is rotationally invariant (i.e., post-multiplying X by an unitary matrix does not change the constraint), and because adding such a constraint does not increase the complexity of solving the surrogate minimization problem. It is not difficult to show [33] that that solving the following constrained minimization

$$\begin{aligned} &\min q(X) \\ &XX^* = I \\ &\|XX^* - X^{(i)}X^{(i)^*}\|_F^2 \leq \Delta \end{aligned} \tag{4.16}$$

is equivalent to solving a low rank perturbed linear eigenvalue problem

$$\left[H(X^{(i)}) - \sigma X^{(i)}X^{(i)^*}\right]X = X\Lambda, \tag{4.17}$$

where σ is essentially the Lagrange multiplier for the inequality constraint in (4.16) and Λ is a diagonal matrix that contains the n_e smallest eigenvalues of the low rank perturbed $H^{(i)}$. When σ is sufficiently large (which corresponds to a trust region radius Δ that is sufficiently small), the solution to (4.17) is guaranteed to produce a reduction in $\hat{E}_{total}(X)$.

4.4.3 LOBPCG for Solving Linear Eigenvalue Problems

As we mentioned earlier, Step 3 of the SCF iteration in which one must compute approximations to desired eigenvectors of a fixed Hamiltonian must is solved by the LOBPCG algorithm [13]. The algorithm is an iterative procedure that seeks the minimum of the quadratic surrogate function (4.14) by projecting the function into a sequence of low dimensional subspaces. The minimizer within each subspace, which can be obtained by solving a small generalized eigenvalue problem produces the optimal search direction and "step length" simultaneously. We will refer users to [13] for details of the algorithm. We would like to point out that a user can easily experiment with this algorithm in KSSOLV after a `Ham` object `H` has been properly constructed. To invoke LOBPCG, one can use

```
[X,lambda,LVEC,RVEC] = lobpcg(H, X0, prec, tol, maxit),
```

where `H` is a predefined `Ham` object, `X0` is a `Wavefun` object that contains the initial guess to the minimizer of (4.14), `prec` is a preconditioner (which should be constructed as a `Wavefun` object), `tol` is the convergence tolerance a user must set, and `maxit` is the maximum number of LOBPCG iterations allowed. Upon completion, `lobpcg` returns the approximation to the desired wavefunction `X`, the corresponding eigenvalue approximation stored in the array `lambda`, and residual norms (`RVEC`) and eigenvalues approximation (`LVEC`) computed at each iteration.

4.5 DCM

Instead of focusing on Kohn-Sham equations (4.12) and minimizing the total energy indirectly in the SCF iteration, we can minimize the total energy directly in an iterative procedure that involves finding a sequence of search directions along which $\hat{E}_{total}(X)$ decreases and computing an appropriate step length. In most of the earlier direct minimization methods developed in [2, 9, 15, 18, 26, 29, 30], the search direction and step length computations are carried out separately. This separation sometimes results in slow convergence. We recently developed a new direct constrained minimization (DCM) algorithm [32, 33] in which the search direction

and step length are obtained simultaneously in each iteration by minimizing the total energy within a subspace spanned by columns of

$$Y = \left(X^{(i)}, M^{-1}R^{(i)}, P^{(i-1)} \right),$$

where $X^{(i)}$ is the approximation to X obtained at the ith iteration, $R^{(i)} = H^{(i)}X^{(i)} - X^{(i)}\Lambda^{(i)}$, M is a hermitian positive definite preconditioner, and $P^{(i-1)}$ is the search direction obtained in the previous iteration. It was shown in [32] that solving the subspace minimization problem is equivalent to computing the eigenvectors G associated with the n_e smallest eigenvalues of the following nonlinear eigenvalue problem

$$\hat{H}(G)G = BG\Omega, \quad G^*BG = I, \tag{4.18}$$

where

$$\hat{H}(G) = Y^* \left[\frac{1}{2}L + V_{ion} + \mathrm{Diag}\left(L^\dagger \rho(YG) \right) + \mathrm{Diag}\left(\mu_{xc}(\rho(YG)) \right) \right] Y, \tag{4.19}$$

and $B = Y^*Y$.

Because the dimension of $\hat{H}(G)$ is at most $3n_e \times 3n_e$, which is normally much smaller than that of $H(X)$, it is relatively easy to solve (4.18) by, for example, a trust region enabled SCF (TRSCF) iteration. We should note that it is not necessary to solve (4.18) to full accuracy in the early stage of the DCM algorithm because all we need is a G that yields sufficient reduction in the objective function.

Once G is obtained, we can update the wave function by

$$X^{(i+1)} \leftarrow YG.$$

The search direction associated with this update is defined, using the MATLAB submatrix notation, to be

$$P^{(i)} \equiv Y(:, n_e + 1 : 3n_e)G(n_e + 1 : 3n_e, :).$$

A complete description of the constrained minimization algorithm is shown in Figure 4.2. We should point out that solving the projected optimization problem in Step 7 of the algorithm requires us to evaluate the projected Hamiltonian $\hat{H}(G)$ repeatedly as we search for the best G. However, since the first two terms of $\hat{H}$ do not depend on G, they can be computed and stored in advance. Only the last two terms of (4.19) need to be updated. These updates require the charge density, the Coulomb and the exchange-correlation potentials to be recomputed.

4.6 Options for SCF and DCM functions

KSSOLV allows one to change a number of parameters used in either SCF or DCM functions. These parameters can be changed by defining and modifying an option structure that can be created by simply calling

```
options = setksopt;
```

This call creates a structure named `options` that contains a number of default parameters used in SCF and DCM functions. To see these parameters, one can simply type `options` at the MATLAB prompt without a semicolon at the end.

Algorithm: A Constrained Minimization Algorithm for Total Energy Minimization

Input: initial set of wave functions $X^{(0)} \in \mathbb{C}^{n \times n_e}$; ionic pseudopotential; a preconditioner M;

Output: $X \in \mathbb{C}^{n \times k}$ such that the Kohn-Sham total energy functional $E_{total}(X)$ is minimized and $X^*X = I_k$.

1. Orthonormalize $X^{(0)}$ such that ${X^{(0)}}^* X^{(0)} = I_k$;

2. for $i = 0, 1, 2, \ldots$ until convergence

3. Compute $\Theta = {X^{(i)}}^* H^{(i)} X^{(i)}$;

4. Compute $R = H^{(i)} X^{(i)} - X^{(i)} \Theta$,

5. if $(i > 1)$ then
$Y \leftarrow (X^{(i)}, M^{-1}R, P^{(i-1)})$
else
$Y \leftarrow (X^{(i)}, M^{-1}R)$;
endif

6. $B \leftarrow Y^*Y$;

7. Find $G \in \mathbb{C}^{2n_e \times n_e}$ or $\mathbb{C}^{3n_e \times n_e}$ that minimizes $E_{total}(YG)$ subject to the constraint $G^*BG = I_{n_e}$;

8. Set $X^{(i+1)} = YG$;

9. if $(i > 1)$ then
$P^{(i)} \leftarrow Y(:, n_e + 1 : 3n_e)G(n_e + 1 : 3n_e, :)$;
else
$P^{(i)} \leftarrow Y(:, n_e + 1 : 2n_e)G(n_e + 1 : 2n_e, :)$;
endif

10. end for

Figure 4.2: A Direct Constrained Minimization Algorithm for Total Energy Minimization

```
options =

       verbose: 'on'
    maxscfiter: 10
    maxdcmiter: 10
    maxinerscf: 3
     maxcgiter: 10
        scftol: 1.0000e-08
        dcmtol: 1.0000e-08
         cgtol: 1.0000e-06
       mixtype: 'anderson'
        mixdim: 9
       betamix: 0.8000
         brank: 1
            X0: []
          rho0: []
        degree: 10
```

Each field can be modified by simply resetting the value of the field through an assignment statement. For example, to change the maximum number of SCF iterations allowed, we can simply use

```
opts.maxscfiter = 20.
```

This parameter can also be modified by calling `setksopt` again with appropriate argument pairs. For example,

```
opts = setksopt(opts,'maxscfiter',20);
```

achieves the same effect. The advantage of calling `setksopt` is that the function checks the validity of the structure field and its value.

Table 4.1 lists all options that can be defined by `setksopt` and explains what they are used for.

option name	allowed value	purpose
`verbose`	'on' or 'off'	display diagonostic info
`maxscfiter`	positive integer	maximum SCF iterations allowed
`maxdcmiter`	positive integer	maximum DCM iterations allowed
`maxinerscf`	positive integer	maximum number of inner SCF iterations allowed in the DCM algorithm
`maxcgiter`	positive integer	maximum LOBPCG iterations allowed
`scftol`	positive float	convergence tolerance for SCF
`dcmtol`	positive float	convergence tolerance for DCM
`cgtol`	positive float	convergence tolerance for LOBPCG
`mixtype`	'anderson' 'broyden' 'broyden1' 'pulay' 'kerker' 'pulay+kerker'	charge or potential mixing types
`mixdim`	positive integer	maximum number of previous potentials to be mixed
`betamix`	positive float	The damping parameter used in Anderson and Broyden mixing
`brank`	positive integer	The rank of the Broyden update
`X0`	Wavefun object	An initial guess of the wavefunctions
`rho0`	3-D array	An initial guess of the charge. To restart an SCF or DCM calculation using previously computed results One should pass in both the previously obtained wavefunction and charge.
`degree`	positive integer	The degree of the Chebyshev polynomial used in chebyscf.

Table 4.1: Options that can be defined by calling `setksopt`

Bibliography

[1] D. G. Anderson. Iterative procedures for nonlinear integral equations. *J. Assoc. Comput. Mach.*, 12:547, 1965.

[2] T. A. Arias, M. C. Payne, and J. D. Joannopoulos. Ab initio molecular dynamics: Analytically continued energy functionals and insights into iterative solutions. *Phys. Rev. Lett.*, 69:1077–1080, 1992.

[3] N. W. Ashcroft and N. D. Mermin. *Solid State Physics*. Brooks Cole, Pacific Grove, CA, 1976.

[4] Felix Bloch. Über die Quantenmechanik der Elektronen in Kristallgittern. *Z. Physik*, 52:555–600, 1928.

[5] P. J. Davis. *Circulant Matrices*. Wiley, New York, 1979.

[6] P. P. Ewald. Die Berchnung optischer und elektrostatischer Gitterpotentiale. *Ann. Phys.*, 64:253–287, 1921.

[7] V. Eyert. A comparison study on methods for convergence acceleration of iterative vector sequences. *J. Comp. Phys.*, 124:271–285, 1996.

[8] H. R. Fang and Y. Saad. Two classes of multisecant methods for nonlinear acceleration. *Numerical Linear Algebra and Its Applications*, to appear, 2009.

[9] M. J. Gillan. Calculation of the vacancy formation in aluminum. *J. Phys. Condens. Matter*, 1:689–711, 1989.

[10] J. Ihm, A. Zunger, and M. L. Cohen. Momentum-space formalism for the total energy of solids. *J. Phys. C: Solid State Physics*, 12:4409–4422, 1979.

[11] G. P. Kerker. Efficient iteration scheme for self-consistent pseudopotential calculations. *Phys Rev. B*, 23:3082–3084, 1981.

[12] L. Kleinman and D. M. Bylander. Efficacious from for model pseudopotentials. *Phys. Rev. Lett.*, 48:1425, 1982.

[13] A. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM J. Sci. Comput.*, 22(2):517–541, 2001.

[14] W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Phys. Rev.*, 140(4A):A1133–A11388, 1965.

[15] G. Kresse and J. Furthmüller. Efficiency of ab initio total energy calculations for metals and semiconductors using a plane-wave basis set. *Computational Materials Science*, 6:15–50, 1996.

[16] L. Marks and R. Luke. Robust mixing for *ab initio* quantum mechanical calculations. *Phys. Rev. B*, 78:075114–1–12, 2008.

[17] H. Nyquist. Certain topics in telegraph transmission theory. *Trans. AIEE*, 47:617–644, 1928.

[18] M. C. Payne, M. P. Teter, D. C. Allen, T. A. Arias, and J. D. Joannopoulos. Iterative minimization techniques for *ab initio* total energy calculation: molecular dynamics and conjugate gradients. *Reviews of Modern Physics*, 64(4):1045–1097, 1992.

[19] J. C. Phillips. Energy-band interpolation scheme based on a pseudopotential. *Phys. Rev.*, 112(3):685–695, 1958.

[20] J. C. Phillips and L. Kleinman. New method for calculating wave functions in crystals and molecules. *Phys. Rev.*, 116(2):287–294, 1958.

[21] W. E. Pickett. Pseudopotential methods in condensed matter applications. *Computer Physics Report*, 9:115–197, 1989.

[22] P. Pulay. Convergence acceleration of iterative sequences: The case of SCF iteration. *Chemical Physics Letters*, 73(2):393–398, 1980.

[23] P. Pulay. Improved SCF convergence acceleration. *Journal of Computational Chemistry*, 3(4):556–560, 1982.

[24] D. Raczkowski, A. Canning, and L. W. Wang. Thomas-Fermi charge mixing for obtaining self-consistency in density functional calculations. *Physical Review B*, 64:121101–1–4, 2001.

[25] W. Ritz. Ueber eine neue methode zur lösung gewisser variationsproblem der mathematischen physik. *J. Reine Angew. Math*, 135:1–61, 1908.

[26] M. P. Teter, M. C. Payne, and D. C. Allan. Solution of Schrödinger's equation for large systems. *Physical Review B*, 40(18):12255–12263, 1989.

[27] N. Troullier and J. L. Martins. Efficient pseudopotentials for plane-wave calculations. *Phys. Rev. B*, 43:1993–2005, 1991.

[28] C. Van Loan. *Computational Frameworks for the Fast Fourier Transform.* SIAM, Philadelphia, PA, 1987.

[29] J. VandeVondele and Jurg Hutter. An efficient orbital transformation method for electronic structure calculations. *Journal of Chem. Phys.*, 118:4365–4369, 2003.

[30] T. Van Voorhis and M. Head-Gordon. A geometric approach to direct minimization. *Molecular Physics*, 100(11):1713–1721, 2002.

[31] C. Yang, J. C. Meza, B. Lee, and L-W. Wang. KSSOLV: A MATLAB toolbox for solving the Khon-Sham equations. *ACM Trans. Math. Soft.*, to appear, 2009.

[32] C. Yang, J. C. Meza, and L. W. Wang. A constrained optimization algorithm for total energy minimization in electronic structure calculation. *Journal of Computational Physics*, 217:709–721, 2005.

[33] C. Yang, J. C. Meza, and L. W. Wang. A trust region direct constrained minimization algorithm for the Kohn-Sham equation. *SIAM J. Sci. Comp.*, 29(5):1854–1875, 2007.

[34] M. T. Yin and M. L. Cohen. Theory of *ab initio* pseudopotential calculations. *Phys. Rev. B*, 25(12):7403–7412, 1982.