Electronic Structure Calculation using Plane Wave Basis Set

Fadjar Fathurrahman

Contents

Chapter 1

Introduction

This document is part of package `ffr-ElectronicStructure.jl` using plane wave basis set.

1.1 Outline

First, I will describe the top level description about what the a typical electronic structure calculation based on density functional theory is carried out. After that, I will break it apart into smaller pieces which are hopefully easier to implement and be understood.
In a typical DFT calculation we are trying to solve the so-called Kohn-Sham equations:

$$H_{KS}\psi_{KS} = e_{KS}\psi_{KS} \tag{1.1}$$

Tasks:

- Setting up data structure for PW basis set
- Solving Poisson equation using FFT
- Solving Schrodinger equation: diagonalization, energy minimization
- Solving KS equation: SCF and energy minimzation

Chapter 2

Solving Poisson and Schrodinger equation

In this chapter, three basics ingredients for our density functional theory calculations:

1. plane wave basis set
2. solving Poisson equation
3. solving Schrodinger equation

2.1 First implementation of plane wave basis: `PWGrid_01.jl`

To describe a plane wave basis, we need to define our periodic simulation box by specifying three lattice vectors. We also need to specify number sampling points for each lattice vector. We will store the lattice vector in 3×3 matrix. (add convention for lattice vectors, probably using the same convention as PWSCF input file).
In file `PWGrid_01.jl`, we give an implementation of plane wave basis set, which is encapsulated in a user-defined type `PWGrid`. An instance of `PWGrid` can be initialize via code like this:

```
Ns = [40, 40, 40]   # sampling points
LatVecs = 10*diagm(ones(3))   # lattice vectors for cubic system
pw = PWGrid( Ns, LatVecs )
```

2.1.1 Details of `PWGrid`

Let's look into details of `PWGrid`.
`PWGrid` is defined like this:

```
type PWGrid
  Ns::Array{Int64}
  LatVecs::Array{Float64,2}
  RecVecs::Array{Float64,2}
  Npoints::Int
  Ω::Float64
  r::Array{Float64,2}
  G::Array{Float64,2}
  G2::Array{Float64}
end
```

Some explanation about these fields follow:

- `Ns` is an integer array which defines number of sampling points in each lattice vectors.
- `LatVecs` is 3×3 matrix which defines lattice vectors of unit cell in real space.

- `RecVecs` is 3×3 matrix which defines lattice vectors of unit cell in reciprocal space. It is calculated according to (3.2).
- `Npoints` Total number of sampling points
- `Ω` Unit cell volume in real space
- `r` Real space grid points
- `G` **G**-vectors
- `G2` Magnitude of **G**-vectors

The constructor for `PWGrid` is defined as follow.

```
function PWGrid( Ns::Array{Int,1}, LatVecs::Array{Float64,2} )
  Npoints = prod(Ns)
  RecVecs = 2*pi*inv(LatVecs')
  Ω = det(LatVecs)
  R,G,G2 = init_grids( Ns, LatVecs, RecVecs )
  return PWGrid( Ns, LatVecs, RecVecs, Npoints, Ω, R, G, G2 )
end
```

The function `init_grid()` is defined as follow. It takes `Ns`, `LatVecs`, and `RecVecs` as the arguments.

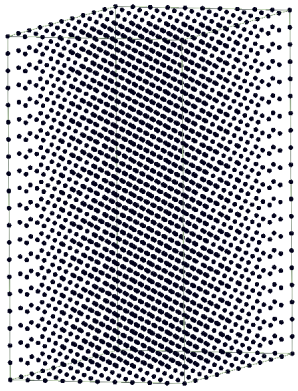
```
function init_grids( Ns, LatVecs, RecVecs )
```

First, grid points in real space are initialized:

```
  Npoints = prod(Ns)
  r = Array(Float64,3,Npoints)
  ip = 0
  for k in 0:Ns[3]-1
  for j in 0:Ns[2]-1
  for i in 0:Ns[1]-1
    ip = ip + 1
    r[1,ip] = LatVecs[1,1]*i/Ns[1] + LatVecs[2,1]*j/Ns[2]
              + LatVecs[3,1]*k/Ns[3]
    r[2,ip] = LatVecs[1,2]*i/Ns[1] + LatVecs[2,2]*j/Ns[2]
              + LatVecs[3,2]*k/Ns[3]
    r[3,ip] = LatVecs[1,3]*i/Ns[1] + LatVecs[2,3]*j/Ns[2]
              + LatVecs[3,3]*k/Ns[3]
  end
  end
  end
```

In the next step, grid points in reciprocal space, or **G**-vectors and also their squared values are initialized

```
  G  = Array(Float64,3,Npoints)
  G2 = Array(Float64,Npoints)
  ip    = 0
  for k in 0:Ns[3]-1
  for j in 0:Ns[2]-1
  for i in 0:Ns[1]-1
    gi = mm_to_nn( i, Ns[1] )
    gj = mm_to_nn( j, Ns[2] )
    gk = mm_to_nn( k, Ns[3] )
    ip = ip + 1
```



```
    G[1,ip] = RecVecs[1,1]*gi + RecVecs[2,1]*gj + RecVecs[3,1]*gk
    G[2,ip] = RecVecs[1,2]*gi + RecVecs[2,2]*gj + RecVecs[3,2]*gk
    G[3,ip] = RecVecs[1,3]*gi + RecVecs[2,3]*gj + RecVecs[3,3]*gk
    G2[ip] = G[1,ip]^2 + G[2,ip]^2 + G[3,ip]^2
  end
  end
  end
```

The function `mm_to_nn` defines mapping from real space to Fourier space:

```
function mm_to_nn(mm::Int,S::Int)
  if mm > S/2
    return mm - S
  else
    return mm
  end
end
```

Finally, the variables `r`, `G`, and `G2` are returned.

```
  return r,G,G2
```

2.1.2 Visualizing real-space grid points

In the directory `pwgrid_01`, we visualize grid points in real space using Xcrysden program. Originally Xcrysden, is meant to visualize crystalline structure, however, we also can use it to visualize grid points, taking periodic boundary conditions into consideration. This is useful to check whether grid points are generated correctly or not.

2.2 Solving Poisson equation

Poisson equation is relatively easy to solve in periodic boundary condition using Fourier method (FFT). This is different from other discretization method, such as finite difference or Lagrange function.
An example a program to solve Poisson equation is given in directory `poisson_01`. In this program, a charge density is constructed from difference between two Gaussian charge density. Total charge (integrated charge density) is restricted to zero. From this charge density, we calculate the electrostatic (Hartree) potential by solving Poisson equation.
Function to generate vector `dr`:

```
function gen_dr( r, center )
  Npoints = size(r)[2]
  dr = Array(Float64,Npoints)
```

```
  for ip=1:Npoints
    dx2 = ( r[1,ip] - center[1] )^2
    dy2 = ( r[2,ip] - center[2] )^2
    dz2 = ( r[3,ip] - center[3] )^2
    dr[ip] = sqrt( dx2 + dy2 + dz2 )
  end
  return dr
end
```

Function to generate charge density:

```
function gen_rho( dr, σ1, σ2 )
  Npoints = size(dr)[1]
  rho = Array( Float64, Npoints )
  c1 = 2*σ1^2
  c2 = 2*σ2^2
  cc1 = sqrt(2*pi*σ1^2)^3
  cc2 = sqrt(2*pi*σ2^2)^3
  for ip=1:Npoints
    g1 = exp(-dr[ip]^2/c1)/cc1
    g2 = exp(-dr[ip]^2/c2)/cc2
    rho[ip] = g2 - g1
  end
  return rho
end
```

Function to solve Poisson equation:

```
function solve_poisson( pw_grid::PWGrid, rho )
  Ω  = pw_grid.Ω
  G2 = pw_grid.G2
  Ns = pw_grid.Ns
  Npoints = pw_grid.Npoints
  ctmp = 4.0*pi*R_to_G( Ns, rho )
  for ip = 2:Npoints
    ctmp[ip] = ctmp[ip] / G2[ip]
  end
  ctmp[1] = 0.0
  phi = real( G_to_R( Ns, ctmp ) )
  return phi
end
```

2.3 Solving Schrodinger equation

Using energy minimization:
Introduction to minimization
simple 2D minimization, using steepest-descent and conjugate gradient method
Using iterative diagonalization: Davidson and LOBPCG
background information about iterative diagonalization Eigenvalue problems

Chapter 3

Formulae

Plane wave basis $b_\alpha(\mathbf{r})$:

$$b_\alpha(\mathbf{r}) = \frac{1}{\sqrt{\Omega}} e^{\mathbf{G}_\alpha \cdot \mathbf{r}} \tag{3.1}$$

Lattice vectors of unit cell in reciprocal space:

$$\mathbf{b} = 2\pi \left(a^T\right)^{-1} \tag{3.2}$$

G-vectors:

$$\mathbf{G} = i\mathbf{b}_1 + j\mathbf{b}_2 + k\mathbf{b}_3 \tag{3.3}$$

Structure factor:

$$S_I(\mathbf{G}) = \sum_{\mathbf{G}} e^{-\mathbf{G}\cdot\mathbf{X}_I} \tag{3.4}$$