# User Guide for `ffr-LFDFT`

Fadjar Fathurrahman

## Contents

## 1 Introduction

Welcome to `ffr-LFDFT` documentation.

`ffr-LFDFT` is a poor man's program (or collection of subroutines, as of now) to carry out electronic structure calculations based on density functional theory and Lagrange basis set.

How to compile

How to use

input parameters ...

subroutines ... (implementation)

Add tutorial on how to use m_LF3d module to solve Schrodinger equation in 1d.

In LF3d periodic, only gamma-point sampling is used.

## 2 Installation

There is no need for installation, actually. What is meant by installation here is compiling the library `libmain.a` and linking the main executable of `ffr-LFDFT`, namely `ffr_LFDFT.x`

A manually written `Makefile` is provided. On the topmost part of the `Makefile` you need to specify which `make.inc` file you want to use. This `make.inc` file contains definition of compiler executable, compiler flags, and libraries used in the linking process. Several `make.inc` files that I used can be found in the directory `platform`. You need to decide which compiler to use if there are more than one compiler in your system. For a typical Linux system, `make.inc.gfortran` is sufficient. You can manually edit the compiler options in the corresponding `make.inc` files.

Currently, `ffr-LFDFT` is tested using the following compilers on Linux system:

- GNU Fortran compiler, executable: `gfortran`
- G95 Fortran compiler, executable: `g95`
- Intel Fortran compiler, executable: `ifort`
- PGI Fortran compiler, executable: `pgf90` or `pgf95`
- Sun (now part of Oracle) Fortran compiler: `sunf95`

There following external libraries are required to build `ffr-LFDFT`

- BLAS
- LAPACK
- FFTW3

Typing the command

```
make
```

will build the library `libmain.a` and typing the command

```
make main
```

will build the main executable `ffr_LFDFT.x`.

## 3 Usage

`ffr-LFDFT` main executable, `ffr_LFDFT.x` supports a subset of PWSCF input file.

The following input file is for LiH molecule:

```
&CONTROL
  pseudo_dir = '../../HGH'
  etot_conv_thr = 1.0d-6
/
&SYSTEM
  ibrav = 8
  nat = 2
  ntyp = 2
  A = 8.4668d0
  B = 8.4668d0
  C = 8.4668d0
  nr1 = 45
  nr2 = 45
  nr3 = 45
/
&ELECTRONS
  KS_Solve = 'Emin_pcg'
  cg_beta = 'DY'
  electron_maxstep = 150
  mixing_beta = 0.1
  diagonalization = 'LOBPCG'
/
ATOMIC_SPECIES
Li   3.0  Li_sc.hgh
H    1.0  H.hgh
ATOMIC_POSITIONS angstrom
Li   0.0  0.0  0.0
H    1.0  0.0  0.0
```

Special to `ffr-LFDFT`, not found in PWSCF, in `ELECTRONS`

- `KS_Solve`: method to solve Kohn-Sham equation, accepted values:
  - `'SCF'`: using diagonalization-based self-consistent iterations
  - `'Emin-pcg'`: using direct minimization based on nonlinear conjugate gradient algorithm
- `cg_beta`: method to calculate parameter $\beta$ in nonlinear CG minimization used in direct Kohn-Sham energy minimization.
  - `'FR'`: using Fletcher-Reeves formula
  - `'PR'`: using Polak-Ribiere formula
  - `'HS'`: using Hestenes-Stiefel formula
  - `'DY'`: using Dai-Yuan formula

## 4 Kohn-Sham equation

Within LDA, Kohn-Sham energy functional can be written as:

$$E_{\mathrm{LDA}}\left[\{\psi_i(\mathbf{r})\}\right] = E_{\mathrm{kin}} + E_{\mathrm{ion}} + E_{\mathrm{Ha}} + E_{\mathrm{xc}} \tag{1}$$

with the following energy terms.
(1) kinetic energy:

$$E_{\mathrm{kin}} = -\frac{1}{2}\sum_i \int \psi_i^*(\mathbf{r})\, \nabla^2\, \psi_i(\mathbf{r})\, \mathrm{d}\mathbf{r} \tag{2}$$

(2) ion-electron interaction energy:

$$E_{\mathrm{ion}} = \int V_{\mathrm{ion}}(\mathbf{r})\, \rho(\mathbf{r})\, \mathrm{d}\mathbf{r} \tag{3}$$

(3) Hartree (electrostatic) energy:

$$E_{\mathrm{Ha}} = \int \frac{1}{2}\frac{\rho(\mathbf{r})\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}\mathrm{d}\mathbf{r}\mathrm{d}\mathbf{r}' \tag{4}$$

(4) Exchange-correlation energy (using LDA):

$$E_{\mathrm{xc}} = \int \epsilon_{\mathrm{xc}} \left[\rho(\mathbf{r})\right] \rho(\mathbf{r})\, \mathrm{d}\mathbf{r} \tag{5}$$

Central to the density functional theory is the so-called Kohn-Sham equation. This equation can be written as:

$$\left[ -\frac{1}{2}\nabla^2 + V_{\mathrm{KS}}(\mathbf{r}) \right] \psi_i(\mathbf{r}) = \epsilon_i \psi_i(\mathbf{r}) \tag{6}$$

where $\epsilon i$ and $\psi_i(\mathbf{r})$ is known as Kohn-Sham eigenvalues and eigenvectors (orbitals). Quantity $V_{\mathrm{KS}}$ is called the Kohn-Sham potential, which can be written as sum of several potentials:

$$V_{\mathrm{KS}}(\mathbf{r}) = V_{\mathrm{ion}}(\mathbf{r}) + V_{\mathrm{Ha}}(\mathbf{r}) + V_{\mathrm{xc}}(\mathbf{r}) \tag{7}$$

$V_{\mathrm{ion}}$ denotes attractive potential between ion (or atomic nuclei) with electrons. This potential can be written as:

$$V_{\mathrm{ion}}(\mathrm{r}) = \sum_{I}^{N_{\mathrm{atoms}}} \frac{Z_I}{|\mathbf{r} - \mathbf{R}_I|} \tag{8}$$

This potential is Coulombic and has singularities at the ionic centers. It is generally difficult to describe this potential fully. It is common to replace the full Coulombic potential with softer potential which is known as pseudopotential. There are various types or flavors of pseudopotentials. In the current implementation, ion-electron potential, $V_{\mathrm{ion}}$ is treated by pseudopotential. HGH-type pseudopotential is employed due to the the availability of analytic forms both in real and reciprocal space.
$V_{\mathrm{Ha}}$ is the classical Hartree potential. It is defined as

$$V_{\mathrm{Ha}}(\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{\mathbf{r} - \mathbf{r}'}\, \mathrm{d}\mathbf{r}', \tag{9}$$

where $\rho(\mathbf{r})$ denotes electronic density:

$$\rho(\mathbf{r}) = \sum_{i}^{N_{\mathrm{occ}}} \psi_i^*(\mathbf{r})\psi_i(\mathbf{r}) \tag{10}$$

Alternatively, Hartree potential can also be obtained via solving Poisson equation:

$$\nabla^2 V_{\mathrm{Ha}}(\mathbf{r}) = -4\pi\rho(\mathbf{r}) \tag{11}$$

The last term in Equation (7) is exchange-correlation potential.

## 5 Implementation

`ffr-LFDFT` is implemented in simple Fortran language. I used global variables heavily, as opposed to using user-defined type to contained them. Currently, only one user-defined type is used in `ffr-LFDFT`, namely `Ps_HGH_Params_T` which is mainly used for convenience. I tried to make the code clear for those who are beginners in implemeting a density-functional calculations (such as myself).

### 5.1 Main program

Currently, the calculation flow of the main program of `ffr-LFDFT` is as follows:

- Getting program argument as input file and reading the input file

- Initializing molecular structure, pseudopotentials, and Lagrange basis functions, including grids
- Setting additional options if necessary based on the input file
- Initializing electronic states variables
- Setting up Hamiltonian: potential and kinetic operators.
- Solving the Kohn-Sham equation via direct minimization or self-consistent field

The appropriate subroutine calls is given below.

```
CALL getarg( 1, filein )
CALL read_input( filein )
CALL setup_from_input()
CALL setup_options()

CALL init_betaNL()
CALL init_states()
CALL init_strfact_shifted()
CALL calc_Ewald_qe()
CALL alloc_hamiltonian()
CALL init_V_ps_loc_G()
CALL init_nabla2_sparse()
CALL init_ilu0_prec()
CALL gen_guess_rho_gaussian()
CALL gen_random_evecs()
CALL gen_gaussian_evecs()

IF( I_KS_SOLVE == 1 ) THEN
  CALL KS_solve_Emin_pcg()
  CALL calc_evals( Nstates, Focc, evecs, evals )
ELSEIF( I_KS_SOLVE == 2 ) THEN
  CALL KS_solve_SCF_v2()
ENDIF
```

Subroutine `setup_from_input()` is a wrapper to three setup calls:

```
CALL setup_atoms()
CALL setup_PsPot()
CALL setup_LF3d()
```

The subroutine names is self-explanatory.

Subroutine `setup_options()` converts various optional input variables into global variables mainly defined in `m_options`.

Before going into futher details of the calculation, I will describe first the data structures used for describing grids, basis functions, atomic structure and pseudopotentials.

## 5.2 Description of LF basis set

Description of LF basis set in 3d is given in module `m_LF3d`. All global variables in this module is given prefix `LF3d`.

```
MODULE m_LF3d
  IMPLICIT NONE
  INTEGER, PARAMETER :: LF3d_PERIODIC = 1
  INTEGER, PARAMETER :: LF3d_CLUSTER  = 2
  INTEGER, PARAMETER :: LF3d_SINC     = 3
  INTEGER :: LF3d_TYPE
  INTEGER, DIMENSION(3) :: LF3d_NN
  REAL(8), DIMENSION(3) :: LF3d_LL, LF3d_AA, LF3d_BB, LF3d_hh
  INTEGER :: LF3d_Npoints
  REAL(8) :: LF3d_dVol
  REAL(8), ALLOCATABLE :: LF3d_grid_x(:), LF3d_grid_y(:), LF3d_grid_z(:)
  REAL(8), ALLOCATABLE :: LF3d_D1jl_x(:,:), LF3d_D1jl_y(:,:), LF3d_D1jl_z(:,:)
  REAL(8), ALLOCATABLE :: LF3d_D2jl_x(:,:), LF3d_D2jl_y(:,:), LF3d_D2jl_z(:,:)
  REAL(8), ALLOCATABLE :: LF3d_lingrid(:,:)
  INTEGER, ALLOCATABLE :: LF3d_xyz2lin(:,:,:)
  INTEGER, ALLOCATABLE :: LF3d_lin2xyz(:,:)
  REAL(8), ALLOCATABLE :: LF3d_G2(:), LF3d_Gv(:,:)
END MODULE
```

Variables in `m_LF3d` is initialized by calling the subroutine `init_LF3d_XX()`, where `XX` may be one of:

- `p`: periodic LFF
- `c`: cluster LF
- `sinc`: sinc L

```
SUBROUTINE init_LF3d_p( NN, AA, BB )
SUBROUTINE init_LF3d_c( NN, AA, BB )
SUBROUTINE init_LF3d_sinc( NN, hh )
```

In the above subroutines:

- `NN`: an array of 3 integers, specifying sampling points in $x$, $y$ and $z$ direction.
- `AA`: an array of 3 floats, specifying left ends of unit cell.
- `BB`: an array of 3 floats, specifying right ends of unit cell.
- `hh`: an array of 3 floats, specifying spacing between adjacent sampling points.

Note that for periodic and cluster LF we have to specify `NN`, `AA`, and `BB` while for sinc LF we have to specify `NN` and `hh`. Note that for periodic LF `NN` must be odd numbers.
Example:

```
NN = (/ 35, 35, 35 /)
AA = (/ 0.d0, 0.d0, 0.d0 /)
BB = (/ 6.d0, 6.d0, 6.d0 /)
CALL init_LF3d_p( NN, AA, BB )
```

## 5.3 Description of molecular or crystalline structure

Description of molecular or crystalline structure is given in module `m_atoms`. Note that unit cell for crystalline structure (currently only orthorombic structure is possible) is specified by `AA` and `BB` in call to `init_LF3d_p()`

```
MODULE m_atoms
  IMPLICIT NONE
  INTEGER :: Natoms
  INTEGER :: Nspecies
  REAL(8), ALLOCATABLE :: AtomicCoords(:,:)
  INTEGER, ALLOCATABLE :: atm2species(:)
  CHARACTER(5), ALLOCATABLE :: SpeciesSymbols(:)
  REAL(8), ALLOCATABLE :: AtomicValences(:)
  COMPLEX(8), ALLOCATABLE :: StructureFactor(:,:)
END MODULE
```

The global variables in module `m_atoms` can initialized from an XYZ file by calling the subroutine `init_atoms_xyz()`.

```
SUBROUTINE init_atoms_xyz( fil_xyz )
```

This subroutine takes one argument `fil_xyz` which is the path to XYZ file describing the molecular structure or crystalline structure.

In the main program, the variables are initialized by calling the subroutine `setup_atoms()`. This subroutine handles conversion from input data read to internal global variables in module `m_atoms`.

## 5.4 Pseudopotential

Module `m_PsPot`

```
MODULE m_PsPot
  USE m_Ps_HGH, ONLY : Ps_HGH_Params_T
  IMPLICIT NONE
  CHARACTER(128) :: PsPot_Dir = './HGH/'
  CHARACTER(128), ALLOCATABLE :: PsPot_FilePath(:)
  TYPE(Ps_HGH_Params_T), ALLOCATABLE :: Ps_HGH_Params(:)
  INTEGER :: NbetaNL
  REAL(8), ALLOCATABLE :: betaNL(:,:)
  INTEGER, ALLOCATABLE :: prj2beta(:,:,:,:)
  INTEGER :: NprojTotMax
END MODULE
```

We currently support HGH pseudopotential only. The HGH pseudopotential parameter is described by an array of type `Ps_HGH_Params_T` which is defined in `m_Ps_HGH`:

```
TYPE Ps_HGH_Params_T
  CHARACTER(5) :: atom_name
  INTEGER :: zval
  REAL(8) :: rlocal
  REAL(8) :: rc(0:3)
  REAL(8) :: c(1:4)
  REAL(8) :: h(0:3, 1:3, 1:3)
  REAL(8) :: k(0:3, 1:3, 1:3)
  INTEGER :: lmax
  INTEGER :: Nproj_l(0:3)  ! number of projectors for each AM
  REAL(8) :: rcut_NL(0:3)
END TYPE
```

The array `betaNL` and `prj2beta` are related to nonlocal pseudopotential calculation.

Except for the array `betaNL`, most variables in module `m_PsPot` are initialized by the call to subroutine `init_PsPot`.

```
ALLOCATE( PsPot_FilePath(Nspecies) )
ALLOCATE( Ps_HGH_Params(Nspecies) )
DO isp = 1,Nspecies
  PsPot_FilePath(isp) = trim(PsPot_Dir) // trim(SpeciesSymbols(isp)) // '.hgh'
  CALL init_Ps_HGH_Params( Ps_HGH_Params(isp), PsPot_FilePath(isp) )
  AtomicValences(isp) = Ps_HGH_Params(isp)%zval
ENDDO
```

Initialization of array `prj2beta`

```
ALLOCATE( prj2beta(1:3,1:Natoms,0:3,-3:3) )
prj2beta(:,:,:,:) = -1
NbetaNL = 0
DO ia = 1,Natoms
  isp = atm2species(ia)
  DO l = 0,Ps_HGH_Params(isp)%lmax
    DO iprj = 1,Ps_HGH_Params(isp)%Nproj_l(l)
      DO m = -l,l
        NbetaNL = NbetaNL + 1
        prj2beta(iprj,ia,l,m) = NbetaNL
      ENDDO ! m
    ENDDO ! iprj
  ENDDO ! l
ENDDO ! ia
```

## 5.5 Kinetic operator and energy

Using LF basis:

$$T_{\alpha,\beta,\gamma} = -\frac{1}{2}\sum_i f_i \left\langle \psi_i \,\middle|\, \nabla^2 \,\middle|\, \psi i \right\rangle = \sum_i f_i \sum_{\alpha\alpha'}\sum_{\beta\beta'}\sum_{\gamma\gamma'} C_{i,\alpha\beta\gamma} K_{\alpha\beta\gamma}^{\alpha'\beta'\gamma'} C_{i,\alpha'\beta'\gamma'} \tag{12}$$

Kinetic operator matrix has the following form:

$$K_{\alpha\beta\gamma}^{\alpha'\beta'\gamma'} = t_{\alpha\alpha'}\delta_{\beta\beta'}\delta_{\gamma\gamma'} + t_{\beta\beta'}\delta_{\alpha\alpha'}\delta_{\gamma\gamma'} + t_{\gamma\gamma'}\delta_{\alpha\alpha'}\delta_{\beta\beta'} \tag{13}$$

For periodic LF:

$$t_{nn'} = -\left(\frac{2\pi}{L}\right)^2 \frac{N}{6}\left(N+1\right)\delta_{nn'} - \frac{\left(\frac{2\pi}{L}\right)^2 (-1)^{n-n'} \cos\left[\frac{\pi(n-n')}{2N+1}\right]}{4\sin^2\left[\frac{\pi}{2N+1}\right]}(1-\delta_{nn'}) \tag{14}$$

An implementation of the equation (14) can be found in the file `init_deriv_matrix_p`.

```
! Diagonal elements
NPRIMED = (N-1)/2
DO jj = 1, N
  D1jl(jj,jj) = 0d0
  D2jl(jj,jj) = -( 2.d0*PI/L )**2.d0 * NPRIMED * (NPRIMED+1)/3.d0
ENDDO
! Off diagonal elements
DO jj = 1, N
  DO ll = jj+1, N
    nn = jj - ll
    tt1 = PI/L * (-1.d0)**nn
    tt2 = sin(PI*nn/N)
    tt3 = (2.d0*PI/L)**2d0 * (-1.d0)**nn * cos(PI*nn/N)
    tt4 = 2.d0*sin(PI*nn/N)**2d0
    D1jl(jj,ll) =  tt1/tt2
    D1jl(ll,jj) = -tt1/tt2
    D2jl(jj,ll) = -tt3/tt4
    D2jl(ll,jj) = -tt3/tt4
  ENDDO
ENDDO
```

Code for calculating kinetic term contribution to total energy:

```
DO ist = 1, Nstates_occ
  CALL op_nabla2( psi(:,ist), nabla2_psi(:) )
  E_kinetic = E_kinetic + Focc(ist) * &
              (-0.5d0) * ddot( Npoints, psi(:,ist),1, nabla2_psi(:),1 ) * dVol
ENDDO
```

There two ways to implement kinetic term contribution to wave function gradient:

- By building the matrix representation of kinetic operator in the sparse form
- Using matrix-free method

The following code build the matrix representation of kinetic operator:

```
! Number of nonzeros per column
nnzc = Nx + Ny + Nz - 2
! Total number of nonzeros
NNZ  = nnzc*Npoints
! Allocate arrays for CSC format
ALLOCATE( rowval(NNZ) )
ALLOCATE( nzval(NNZ) )
ALLOCATE( colptr(Npoints+1) )
! Initialize rowGbl pattern for x, y, and z components
ALLOCATE( rowGbl_x_orig(Nx) )
ALLOCATE( rowGbl_y_orig(Ny) )
ALLOCATE( rowGbl_z_orig(Nz) )
!
rowGbl_x_orig(1) = 1
DO ix = 2,Nx
  rowGbl_x_orig(ix) = rowGbl_x_orig(ix-1) + Ny*Nz
ENDDO
rowGbl_y_orig(1) = 1
DO iy = 2,Ny
  rowGbl_y_orig(iy) = rowGbl_y_orig(iy-1) + Nz
ENDDO
DO iz = 1,Nz
  rowGbl_z_orig(iz) = iz
ENDDO
ip = 0
DO colGbl = 1,Npoints
  ! Determine local column index for x, y, and z components
  !
  colLoc_x = ceiling( real(colGbl)/(Ny*Nz) )
  !
  yy = colGbl - (colLoc_x - 1)*Ny*Nz
  colLoc_y = ceiling( real(yy)/Nz )
  !
  izz = ceiling( real(colGbl)/Nz )
  colLoc_z = colGbl - (izz-1)*Nz
  ! Add diagonal element (only one element in any column)
  ip = ip + 1
  rowval(ip) = colGbl
  nzval(ip) = D2jl_x(colLoc_x,colLoc_x) + D2jl_y(colLoc_y,colLoc_y) + &
              D2jl_z(colLoc_z,colLoc_z)
  ! Add non-diagonal elements
  DO ix = 1,Nx
    IF ( ix /= colLoc_x ) THEN
      ip = ip + 1
      rowval(ip) = rowGbl_x_orig(ix) + colGbl - 1 - (colLoc_x - 1)*Ny*Nz
      nzval(ip) = D2jl_x(ix,colLoc_x)
    ENDIF
  ENDDO
  DO iy = 1,Ny
    IF ( iy /= colLoc_y ) THEN
      ip = ip + 1
      rowval(ip) = rowGbl_y_orig(iy) + colGbl - 1 - (izz-1)*Nz + (colLoc_x - 1)*Ny*Nz
      nzval(ip) = D2jl_y(iy,colLoc_y)
    ENDIF
  ENDDO
  DO iz = 1,Nz
    IF ( iz /= colLoc_z ) THEN
      ip = ip + 1
      rowval(ip) = rowGbl_z_orig(iz) + (izz-1)*Nz
      nzval(ip) = D2jl_z(iz,colLoc_z)
    ENDIF
  ENDDO
ENDDO
! Now colptr
colptr(1) = 1
DO ii = 2,Npoints+1
  colptr(ii) = colptr(ii-1) + nnzc
ENDDO
! Sort using subroutine csort from SPARSKIT
nwork = max( Npoints+1, 2*(colptr(Npoints+1)-colptr(1)) )
ALLOCATE( iwork( nwork ) )
CALL csort( Npoints, nzval, rowval, colptr, iwork, .TRUE. )
```

Multiplication of kinetic matrix with wave function can be done using SPARSKIT's sparse matrix-vector multiplication subroutine `amux`:

```
CALL amux( Npoints, v(:,ic), Hv(:,ic), nzval, rowval, colptr )
```

Alternatively, one can use matrix-free method (without building 3D Laplacian matrix)

```
DO ip = 1, Npoints
  i = lin2xyz(1,ip)
  j = lin2xyz(2,ip)
  k = lin2xyz(3,ip)
  Lv(ip) = 0.d0
  DO ii = 1, Nx
    Lv(ip) = Lv(ip) + D2jl_x(ii,i) * v( xyz2lin(ii,j,k) )
  ENDDO
  DO jj = 1, Ny
    Lv(ip) = Lv(ip) + D2jl_y(jj,j) * v( xyz2lin(i,jj,k) )
  ENDDO
  DO kk = 1, Nz
    Lv(ip) = Lv(ip) + D2jl_z(kk,k) * v( xyz2lin(i,j,kk) )
  ENDDO
ENDDO
```

ILU0 preconditioner based on kinetic matrix:

```
ALLOCATE( alu_ilu0(Npoints*(Nx+Ny+Nz-2)) )
ALLOCATE( jlu_ilu0(Npoints*(Nx+Ny+Nz-2)) )
ALLOCATE( ju_ilu0(Npoints) )
ALLOCATE( iw_ilu0(Npoints) )
CALL ilu0( Npoints, -0.5d0*nzval, rowval, colptr, alu_ilu0, jlu_ilu0, &
           ju_ilu0, iw_ilu0, ierr )
```

Apply preconditioner:

```
CALL lusol( Npoints, v, Kv, alu_ilu0, jlu_ilu0, ju_ilu0 )
```

## 5.6 Local pseudopotential

Local pseudopotential term is very simple because it is diagonal in real space. This term is represented by the global array `V_ps_loc` found in file `m_hamiltonian`. Despite its name, it can however be used to represent any local external potential such as harmonic potential.

It contribution to total energy is simply sum over grid points:

```
E_ps_loc = sum( Rhoe(:) * V_ps_loc(:) )*dVol
```

Its contribution to wave function gradient is simply obtained by point-wise multiplication with wave function expansion coefficients.

## 5.7 Hartree term: solution of Poisson equation

Hartree potential can be calculated by solving Poisson equation once the charge density has been calculated. There are several methods to solve Poisson equation. For periodic system, the most popular method is via FFT. In this method charge density is first transformed to reciprocal space or **G**-space. In this space, Hartree potential can be calculated by simply dividing charge density with magnitude of non-zero reciprocal vectors **G**.

To implement this method we need to generate reciprocal Gectors **G**. In the current implementation **G**-vectors are declared in module `m_LF3d`, namely `LF3d_Gv` and `LF3d_Gv2` for **G**-vectors and their magnitudes, respectively. The subroutine which is responsible to generate **G**-vectors is `init_gvec()`.

```
ALLOCATE( G2(Npoints) )
ALLOCATE( Gv(3,Npoints) )
ig = 0
DO k = 0, NN(3)-1
DO j = 0, NN(2)-1
DO i = 0, NN(1)-1
  ig = ig + 1
  ii = mm_to_nn( i, NN(1) )
  jj = mm_to_nn( j, NN(2) )
  kk = mm_to_nn( k, NN(3) )
  Gv(1,ig) = ii * 2.d0*PI/LL(1)
  Gv(2,ig) = jj * 2.d0*PI/LL(2)
  Gv(3,ig) = kk * 2.d0*PI/LL(3)
  G2(ig) = Gv(1,ig)**2 + Gv(2,ig)**2 + Gv(3,ig)**2
ENDDO
ENDDO
ENDDO
```

The function `mm_to_nn` describes mapping between real space grid and Fourier grid.

```
FUNCTION mm_to_nn( mm, S ) RESULT(idx)
  IMPLICIT NONE
  INTEGER :: idx
  INTEGER :: mm
  INTEGER :: S
  IF(mm > S/2) THEN
    idx = mm - S
  ELSE
    idx = mm
  ENDIF
END FUNCTION
```

Driver for solving Poisson equation via FFT is implemented in subroutine `Poisson_solve_fft()`

```
ALLOCATE( tmp_fft(Npoints) )
DO ip = 1, Npoints
  tmp_fft(ip) = cmplx( rho(ip), 0.d0, kind=8 )
ENDDO
! forward FFT
CALL fft_fftw3( tmp_fft, Nx, Ny, Nz, .false. )  ! now ‘tmp_fft = rho(G)‘
tmp_fft(1) = (0.d0,0.d0)  ! zero-G component
DO ip = 2, Npoints
  tmp_fft(ip) = 4.d0*PI*tmp_fft(ip) / G2(ip)
ENDDO  ! now ‘tmp_fft‘ = phi(G)
! Inverse FFT
CALL fft_fftw3( tmp_fft, Nx, Ny, Nz, .true. )
! Transform back to real space
DO ip = 1, Npoints
  phi(ip) = real( tmp_fft(ip), kind=8 )
ENDDO
```

Hartree energy is calculated according to the following equation:

$$E_{\mathrm{Ha}} = \frac{1}{2}\int \rho(\mathbf{r})V_{\mathrm{Ha}}(\mathbf{r})\,\mathrm{d}\mathbf{r} \tag{15}$$

This is done by simple summation over grid points:

```
E_Hartree = 0.5d0*sum( Rhoe(:) * V_Hartree(:) )*dVol
```

## 5.8 XC energy and potential

Currently the only supported form of XC functional is of Volko-Wilk-Nusair which is implemented in the file `LDA_VWN.f90`. In the future version, LibXC implementation will be implemented hopefully.

XC contribution to total energy is implemented as follows.

```
CALL excVWN( Npoints, Rhoe, epsxc )
E_xc = sum( Rhoe(:) * epsxc(:) )*dVol
```

**TODO**: Equation for $V_{XC}$ XC contribution to wavefunction gradient is treated the same way as local potential.

```
CALL excVWN( Npoints, Rhoe, epsxc )
CALL excpVWN( Npoints, Rhoe, depsxc )
V_xc(:) = epsxc(:) + Rhoe(:)*depsxc(:)
```

## 5.9 Nonlocal pseudopotential

Nonlocal HGH pseudopotential action can be defined as follows:

$$\hat{V}_{\mathrm{NL}}\psi = \sum_{i}^{N_{\mathrm{occ}}}\sum_{a}^{N_{\mathrm{atom}}}\sum_{l=0}\sum_{m=-l}^{+l}\sum_{i,j} h_{i,j}\beta_{ialm}^{\mathrm{NL}} \tag{16}$$

Contribution to total energy:

```
E_ps_NL = 0.d0
DO ist = 1,Nstates_occ
  enl1 = 0.d0
  DO ia = 1,Natoms
    isp = atm2species(ia)
    DO l = 0,Ps(isp)%lmax
    DO m = -l,l
```

```
      DO iprj = 1,Ps(isp)%Nproj_l(l)
      DO jprj = 1,Ps(isp)%Nproj_l(l)
        ibeta = prj2beta(iprj,ia,l,m)
        jbeta = prj2beta(jprj,ia,l,m)
        hij = Ps(isp)%h(l,iprj,jprj)
        enl1 = enl1 + hij*betaNL_psi(ia,ist,ibeta)*betaNL_psi(ia,ist,jbeta)
      ENDDO ! jprj
      ENDDO ! iprj
    ENDDO ! m
    ENDDO ! l
  ENDDO
  E_ps_NL = E_ps_NL + Focc(ist)*enl1
ENDDO
```

Action of nonlocal pseudopotential to wavefunction:

```
SUBROUTINE op_V_ps_NL( Nstates, Vpsi )
  USE m_LF3d, ONLY : Npoints => LF3d_Npoints
  USE m_PsPot, ONLY : NbetaNL, betaNL, prj2beta, Ps => Ps_HGH_Params
  USE m_atoms, ONLY : Natoms, atm2species
  USE m_hamiltonian, ONLY : betaNL_psi
  IMPLICIT NONE
  INTEGER :: Nstates
  REAL(8) :: Vpsi(Npoints,Nstates)
  INTEGER :: ia, isp, ist, ibeta, jbeta, iprj, jprj
  INTEGER :: l, m
  REAL(8) :: hij

  IF( NbetaNL <= 0 ) THEN
    RETURN
  ENDIF

  Vpsi(:,:) = 0.d0

  DO ist = 1,Nstates
    DO ia = 1,Natoms
      isp = atm2species(ia)
      DO l = 0,Ps(isp)%lmax
      DO m = -l,l
        DO iprj = 1,Ps(isp)%Nproj_l(l)
        DO jprj = 1,Ps(isp)%Nproj_l(l)
          ibeta = prj2beta(iprj,ia,l,m)
          jbeta = prj2beta(jprj,ia,l,m)
          hij = Ps(isp)%h(l,iprj,jprj)
          Vpsi(:,ist) = Vpsi(:,ist) + hij*betaNL(:,ibeta)*betaNL_psi(ia,ist,jbeta)
        ENDDO ! jprj
        ENDDO ! iprj
      ENDDO ! m
      ENDDO ! l
    ENDDO
  ENDDO

END SUBROUTINE
```

The array `betaNL` is defined initialized in subroutine `init_betaNL`:

```
SUBROUTINE init_betaNL()

  USE m_LF3d, ONLY : Npoints => LF3d_Npoints, &
                     lingrid => LF3d_lingrid, &
```

```
                    LL => LF3d_LL, &
                    dVol => LF3d_dVol
  USE m_PsPot, ONLY : betaNL, NbetaNL, &
                     Ps => Ps_HGH_Params
  USE m_atoms, ONLY : atpos => AtomicCoords, Natoms, atm2species
  USE m_Ps_HGH, ONLY : hgh_eval_proj_R
  IMPLICIT NONE
  INTEGER :: ia, isp, l, m, iprj
  INTEGER :: Np_beta, ip, ibeta
  REAL(8) :: dr_vec(3)
  REAL(8) :: dr
  REAL(8) :: Ylm_real
  REAL(8) :: nrm

  ALLOCATE( betaNL(Npoints,NbetaNL) )

  ! loop structure must be the same as in init_PsPot
  ibeta = 0
  DO ia = 1,Natoms
    isp = atm2species(ia)
    DO l = 0,Ps(isp)%lmax
      DO iprj = 1,Ps(isp)%Nproj_l(l)
        DO m = -l,l
          ibeta = ibeta + 1
          Np_beta = 0
          DO ip = 1,Npoints
            CALL calc_dr_periodic_1pnt( LL, atpos(:,ia), lingrid(:,ip), dr_vec )
            dr = sqrt( dr_vec(1)**2 + dr_vec(2)**2 + dr_vec(3)**2 )
            IF( dr <= Ps(isp)%rcut_NL(l) ) THEN
              Np_beta = Np_beta + 1
              betaNL(ip,ibeta) = hgh_eval_proj_R( Ps(isp), l, iprj, dr ) * Ylm_real( l, m,
↪  dr_vec )
            ENDIF
          ENDDO
          nrm = sum(betaNL(:,ibeta)**2)*dVol
          WRITE(*,'(1x,A,I5,I8,F18.10)') 'ibeta, Np_beta, integ = ', ibeta, Np_beta, nrm
        ENDDO ! iprj
      ENDDO ! m
    ENDDO ! l
  ENDDO

END SUBROUTINE
```

and `betaNL_psi` is calculated in `calc_betaNL_psi`:

```
SUBROUTINE calc_betaNL_psi( Nstates, psi )

  USE m_LF3d, ONLY : Npoints => LF3d_Npoints, &
                       dVol => LF3d_dVol
  USE m_PsPot, ONLY : NbetaNL, betaNL
  USE m_hamiltonian, ONLY : betaNL_psi
  USE m_atoms, ONLY : Natoms
  IMPLICIT NONE
  INTEGER :: Nstates
  REAL(8) :: psi(Npoints,Nstates)
  INTEGER :: ist, ibeta, ia
  REAL(8) :: ddot

  ! immediate return if no projectors are available
  IF( NbetaNL <= 0 ) THEN
```

```
    RETURN
  ENDIF

  betaNL_psi(:,:,:) = 0.d0

  DO ia = 1,Natoms
    DO ist = 1,Nstates
      DO ibeta = 1,NbetaNL
        betaNL_psi(ia,ist,ibeta) = ddot( Npoints, betaNL(:,ibeta),1, psi(:,ist),1 ) * dVol
      ENDDO
    ENDDO
  ENDDO

END SUBROUTINE
```

## A Lagrange basis function

### A.1 Periodic Lagrange function

For a given interval $[0, L]$, with $L > 0$, the grid points $x_i$ appropriate for periodic Lagrange function are given by:

$$x_i = \frac{L}{2}\frac{2i-1}{N} \tag{17}$$

with $i = 1, \ldots, N$. Number of points $N$ should be an odd number.
The periodic cardinal functions $L_i^{\text{per}}(x)$, defined at grid point $i$ are given by:

$$L_i^{\text{per}}(x) = \frac{1}{\sqrt{NL}} \sum_{n=1}^{N} \cos\left(\frac{\pi}{L}(2n - N - 1)(x - x_i)\right). \tag{18}$$

The expansion of periodic function in terms of Lagrange functions:

$$f(x) = \sum_{i=1}^{N} c_i L_i^{\text{per}}(x) \tag{19}$$

with expansion coefficients $c_i = \sqrt{L/N} f(x_i)$. When doing variational calculation, the cofficients $c_i$ are the variational parameters. The actual function values $f(x_i)$ at grid points $x_i$ is obtained by $f(x_i) = \sqrt{N/L} c_i$. The prefactor is sometimes abbreviated by $h = L/N$ and is also referred to as scaling factor.
Consider periodic potential in one dimension:

$$V(x + L) = V(x). \tag{20}$$

Floquet-Bloch theorem states that the wave function solution for periodic potentials can be written in the form:

$$\psi_k(x) = e^{\imath kx} \phi_k(x) \tag{21}$$

where function $\phi_k(x)$ and its first derivative $\phi_k'(x)$ have the same periodicity as $V(x)$ and $k$ is a constant called the crystal momentum. Substituting this expression to Schrodinger equation we obtain:

$$\left[-\frac{\hbar^2}{2m}\left(\frac{\mathrm{d}^2}{\mathrm{d}x^2} + 2\imath k\frac{\mathrm{d}}{\mathrm{d}x} - k^2\right) + V(x)\right]\phi_k(x) = E\phi_k(k). \tag{22}$$

An alternative way of enforcing periodicity of the wave function is to require that:

$$\psi_k(x+L) = e^{\imath kL}\psi_k(x). \tag{23}$$

This condition follows from:

$$\begin{aligned}\psi_k(x+L) &= e^{\imath k(x+L)}\phi_k(x+L)\\ &= e^{\imath k(x+L)}\phi_k(x)\\ &= e^{\imath kL}e^{\imath kx}\phi_k(x)\\ &= e^{\imath kL}\psi_k(x)\end{aligned}$$

Using periodic cardinal the Schrodinger equation for periodic potential can be written as:

$$\sum_{j=1}^{N}\left[-\frac{\hbar^2}{2m}\left(D^{(2)}_{jl} + 2\imath k D^{(1)}_{jl} - k^2\delta_{jl}\right) + V(j)\delta_{jl}\right]\phi(j) = E\phi(l) \tag{24}$$

with $l = 1,\ldots,N$. $D^{(1)}_{jl}$ are matrix elements of the first derivatives:

$$D^{(1)}_{jl} = \begin{cases} 0 & j = l \\ -\dfrac{2\pi}{L}(-1)^{j-l}\left(2\sin\dfrac{\pi(j-l)}{N}\right)^{-1} & j \neq l \end{cases} \tag{25}$$

and $D^{(2)}_{jl}$ are matrix elements of the second derivatives, $N' = (N-1)/2$:

$$D^{(2)}_{jl} = \begin{cases} -\left(\dfrac{2\pi}{L}\right)^2\dfrac{N'(N'+1)}{3} & j = l \\ -\left(\dfrac{2\pi}{L}\right)^2(-1)^{j-l}\dfrac{\cos\left(\pi(j-l)/N\right)}{2\sin^2\left[\pi(j-l)/N\right]} & j \neq l \end{cases} \tag{26}$$

Note that, $D^{(1)}_{jl}$ is not symmetric, but $D^{(1)}_{jl} = -D^{(1)}_{lj}$. Meanwhile, the second derivative matrix $D^{(2)}_{jl}$ is symetric, i.e. $D^{(2)}_{jl} = D^{(2)}_{lj}$. With the above expressions, first and second derivative of periodic cardinals can be expressed as

$$\frac{\mathrm{d}}{\mathrm{d}x}L^{\mathrm{per}}_i(x) = \sum_{j=1}^{N} D^{(1)}_{ji}L^{\mathrm{per}}_j(x) \tag{27}$$

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}L^{\mathrm{per}}_i(x) = \sum_{j=1}^{N} D^{(2)}_{ji}L^{\mathrm{per}}_j(x) \tag{28}$$

The previous approach also can be extended to periodic potential in 3D:

$$V(\mathbf{r}) = V(x,y,z) = V\left(x+L_x, y+L_y, z+L_z\right)$$

Using periodic LF, Schrodinger equation can be casted into the following form:

$$\left[-\frac{\hbar^2}{2m}\left(\nabla^2 + 2\imath\mathbf{k}\cdot\nabla - \mathbf{k}^2\right) + V(\mathbf{r})\right]\phi_{\mathbf{k}}(\mathbf{r}) = E\ \phi_{\mathbf{k}}(\mathbf{r}) \tag{29}$$

### A.2 Cluster Lagrange function

For a given interval $[A, B]$, with $B > A$, the grid points $x_i$ appropriate for cluster Lagrange function are given by:

$$x_i = A + \frac{B-A}{N+1} i$$

where $i = 1, \ldots, N$. Number of points $N$ can be either odd or even number.
The cluster Lagrange functions $L_i^{\text{clu}}(x)$, defined at grid point $i$ are given by:

$$L_i^{\text{clu}}(x) = \frac{2}{\sqrt{(N+1)(B-A)}} \sum_{n=1}^{N} \sin\left(k_n(x_i - A)\right) \sin\left(k_n(x - A)\right). \tag{30}$$

where $k_n = \pi n/(B-A)$. The expansion of a function $f(x)$ in terms of cluster Lagrange functions:

$$f(x) = \sum_{i=1}^{N} c_i L_i^{\text{clu}}(x) \tag{31}$$

with expansion coefficients $c_i = \sqrt{(B-A)/(N+1)} f(x_i)$. When doing variational calculation, the cofficients $c_i$ are the variational parameters. The actual function values $f(x_i)$ at grid points $x_i$ is obtained by $f(x_i) = \sqrt{(N+1)/(B-A)} c_i$.
Matrix elements $D_{jl}^{(2)}$ of the second derivatives for cluster Lagrange functions are

$$D_{jl}^{(2)} = \begin{cases} -\dfrac{1}{2}\left(\dfrac{\pi}{B-A}\right)^2 \dfrac{2(N+1)^2+1}{3} - \dfrac{1}{\sin^2\left[\pi j/(N+1)\right]} & j = l \\ -\dfrac{1}{2}\left(\dfrac{\pi}{B-A}\right)^2 (-1)^{j-l} \left[\dfrac{1}{\sin^2\left[\dfrac{\pi(j-l)}{2(N+1)}\right]} - \dfrac{1}{\sin^2\left[\dfrac{\pi(j+l)}{2(N+1)}\right]}\right] & j \neq l \end{cases} \tag{32}$$

For free or cluster boundary condition, we don't need $D_{jl}^{(1)}$.

## B HGH pseudopotential

HGH pseudopotential has analytic forms both in real space and reciprocal space.
Local component of pseudopotential in real space

$$\begin{aligned} V_{\text{loc}}(\mathbf{r}) = -\frac{Z_{\text{ion}}}{r} \text{erf}\left(\frac{r}{\sqrt{2} r_{\text{loc}}}\right) + \\ \exp\left[-\frac{1}{2}\left(\frac{r}{r_{\text{loc}}}\right)^2\right] \times \left[C_1 + C_2\left(\frac{r}{r_{\text{loc}}}\right)^2 + C_3\left(\frac{r}{r_{\text{loc}}}\right)^4 + C_4\left(\frac{r}{r_{\text{loc}}}\right)^6\right] \end{aligned} \tag{33}$$

with parameters: $r_{\text{loc}}$, $C_1$, $C_2$, $C_3$, and $C_4$.
Local component of local pseudopotential in **G**-space:

$$\begin{aligned} V_{\text{loc}}(\mathbf{G}) = -\frac{1}{\Omega}\frac{4\pi Z_{\text{ion}}}{G^2} \exp\left[-\frac{1}{2}(G r_{\text{loc}})^2\right] + \sqrt{8\pi^3}\frac{r_{\text{loc}}}{\Omega} \exp\left[-\frac{1}{2}(G r_{\text{loc}})^2\right] \times \\ \left\{C_1 + C_2\left[3 - (G r_{\text{loc}})^2\right] + C_3\left[15 - 10(G r_{\text{loc}})^2 (G r_{\text{loc}})^4\right]\right. \\ \left. + C_4\left[105 - 105(G r_{\text{loc}})^2 + 21(G r_{\text{loc}})^4 - (G r_{\text{loc}})^6\right]\right\} \end{aligned} \tag{34}$$

Nonlocal component of pseudopotential can be written as

$$V_l(\mathbf{r},\mathbf{r}') = \sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{m=-l}^{l} \beta_{ilm}(\mathbf{r})\, h_{ij}^{l}\, \beta_{jlm}^{*}(\mathbf{r}') \tag{35}$$

with atomic-centered functions projector functions as

$$\beta_{ilm}(\mathbf{r}) = p_i^l(r) Y_{lm}(\hat{\mathbf{r}}) \tag{36}$$

The radial projector functions have the following form in real space

$$p_i^l(r) = \frac{\sqrt{2} r^{l+2(i-1)} \exp\left(-\dfrac{r^2}{2r_l^2}\right)}{r_l^{l+(4i-1)/2}\sqrt{\Gamma\left(l+\dfrac{4i-1}{2}\right)}} \tag{37}$$

The radial projector functions satisfy the following normalization condition

$$\int_0^\infty p_i^l(r) p_i^l(r)\, r^2\, \mathrm{d}r = 1 \tag{38}$$

For $l = 0$, the Fourier transform of radial projector functions can be written as:

$$p_1^{l=0}(G) = \frac{4\sqrt{2r_0^3}\pi^{5/4}}{\sqrt{\Omega}\exp\left[(Gr_0)^2/2\right]} \tag{39}$$

$$p_2^{l=0}(G) = \frac{\sqrt{8\dfrac{2r_0^3}{15}}\pi^{5/4}\left(3-(Gr_0)^2\right)}{\sqrt{\Omega}\exp\left[(Gr_0)^2/2\right]} \tag{40}$$

$$p_3^{l=0}(G) = \frac{16\sqrt{\dfrac{2r_0^3}{105}}\pi^{5/4}\left(15-10(Gr_0)^2-(Gr_0)^4\right)}{3\sqrt{\Omega}\exp\left[(Gr_0)^2/2\right]} \tag{41}$$

For $l = 1$, the Fourier transform of radial projector functions can be written as

$$p_1^{l=1}(G) = \frac{8\sqrt{\dfrac{r_1^5}{3}}\pi^{5/4}G}{\sqrt{\Omega}\exp\left[(Gr_1)^2/2\right]} \tag{42}$$

$$p_2^{l=1}(G) = \frac{16\sqrt{\dfrac{r_1^5}{105}}\pi^{5/4}\left(5-(Gr_1)^2\right)G}{\sqrt{\Omega}\exp\left[(Gr_1)^2/2\right]} \tag{43}$$

$$p_3^{l=1}(G) = \frac{32\sqrt{\dfrac{r_1^5}{1155}}\pi^{5/4}\left(35-14(Gr_1)^2+(Gr_1)^4\right)G}{3\sqrt{\Omega}\exp\left[(Gr_1)^2/2\right]} \tag{44}$$

For $l = 2$, the Fourier transform of radial projector functions can be written as

$$p_1^{l=2}(G) = \frac{8\sqrt{\dfrac{2r_2^7}{15}}\pi^{5/4}G^2}{\sqrt{\Omega}\exp\left[(Gr_2)^2/2\right]} \tag{45}$$

$$p_2^{l=2}(G) = \frac{16\sqrt{\dfrac{2r_2^7}{105}}\pi^{5/4}\left(7-(Gr_2)^2\right)G^2}{3\sqrt{\Omega}\exp\left[(Gr_2)^2/2\right]} \tag{46}$$

For $l = 3$, the Fourier transform of radial projector function can be written as

$$p_1^{l=3}(G) = \frac{16\sqrt{\dfrac{2r_3^9}{105}}\pi^{5/4}G^3}{\sqrt{\Omega}\exp\left[(Gr_3)^2/2\right]} \tag{47}$$