1 Main program: ffr_LFDFT

This is the main program for ffr_LFDFT.

Start from here if you want to learn the call sequence of the main program. Most of the subroutines called here are wrapper subroutines which call various core computational subroutines.

If you want to quickly do a calculation for simple system or want to use directly the core computational subroutines, please see examples in directory tests.

```
PROGRAM ffr_LFDFT
  IMPLICIT NONE
```

Variables for timing stuffs Note that, for the moment the program only does timing for overall subroutines. It is very desirable, however, to measure timing of various steps in the program.

```
  INTEGER :: tstart, counts_per_second, tstop
```

Here begins the calls of wrapper subroutines.

```
!!
```

- The following subroutine will print some information about the program.

  ```
    CALL welcome()
  ```

- We start timing from here.

  ```
    CALL system_clock( tstart, counts_per_second )
  ```

- This subroutine handles reading input file, setting-up basis set and grid points, pseudopotential-related variables, etc.

  ```
    CALL setup_ffr_LFDFT()
  ```

- Preparation for solving Kohn-Sham equation

  ```
    CALL guess_KS_solutions()
  ```

- This soubroutine is the driver for primary subroutines to solve Kohn-Sham equation.

  ```
    CALL do_KS_solve()
  ```

- Free allocated memory

  ```
    CALL cleanup_ffr_LFDFT()
  ```

- Stop timing and display elapsed time

  ```
    CALL system_clock( tstop )
    WRITE(*,*)
    WRITE(*,'(1x,A,ES18.10,A)') 'Total elapsed time: ', &
             dble(tstop - tstart)/counts_per_second, ' second.'
  ```

- Display goodbye message

  ```
    CALL goodbye()
  END PROGRAM
  ```

2 Subroutine setup_ffr_LFDFT()

This subroutine prepares various tasks before actually solving the Kohn-Sham equation.

```
SUBROUTINE setup_ffr_LFDFT()
  USE m_options, ONLY : FREE_NABLA2, I_POISSON_SOLVE
  USE m_input_vars, ONLY : assume_isolated
  USE m_LF3d, ONLY : Npoints => LF3d_Npoints
  USE m_states, ONLY : Nstates, &
                       evals => KS_evals, &
                       evecs => KS_evecs
  !
  INTEGER :: Narg   ! number of argument
  INTEGER :: iargc  ! needed for several compilers
  CHARACTER(64) :: filein
```

- We first check the number of argument(s) give to the program using built-in function `iargc()` and save the result to variable `Narg`. Currently, we only support one argument, i.e. path to input file. The program will stop and display error message if `Narg` is not equal to one.

```
  Narg = iargc()
  IF( Narg /= 1 ) THEN
    WRITE(*,*)
    WRITE(*,*) 'ERROR: exactly one argument must be given: input file path'
    STOP
  ENDIF
```

- We get the actual argument using built-in subroutine `getarg()`.

```
  CALL getarg( 1, filein )
```

- We read the input file using subroutine `read_input()`

```
  CALL read_input( filein )
```

- The following subroutine will initialize global variables related to basis function and grid points, molecular or crystalline structures, and pseudopotentials.

```
  CALL setup_from_input()
```

- Various options, such as convergence criteria, choice of algorithms, etc which are given in the input file, will be converted to internal variables (mostly defined in module `m_options`).

```
  CALL setup_options()
```

- The following calls will output information about molecular or crystalline structures, pseudopotentials, and basis function and grid points.

```
  CALL info_atoms()
  CALL info_PsPot()
  CALL info_LF3d()
```

- This subroutine initialize nonlocal pseudopotential projectors. It must be called after variables from `m_LF3d` are initialized as they are defined on grid points.

  TODO/FIXME: To be consistent, this call should be made in pseudopotential setup. (probably via subroutine `setup_from_input()`)

```
  CALL init_betaNL()
```

- This call will determined number of occupied and unoccupied states. It will also initialize occupation numbers.

  Note: Memories for eigenvectors and eigenvalues of KS equations are not allocated here.

```
CALL init_states()
```

- Allocate KS eigenvectors and eigenvalues.

  This step should be done in some wrapper subroutine, however, for current use-case this is suffice.

```
ALLOCATE( evecs(Npoints,Nstates), evals(Nstates) )
```

- This call will initialize and calculate structure factor $S_f(\mathbf{G})$. This is required for periodic LF.

  TODO/FIXME: This call should be made in `setup_from_input` or anther subroutine.

```
CALL init_strfact_shifted()
```

- Ewald energy is calculated here.

  FIXME: Ewald energy calculation should be called everytime atomic positions are updated, for example in geometry optimization or molecular dynamics.

```
IF( assume_isolated == 'sinc' ) THEN
  CALL calc_E_NN()
ELSE
  CALL calc_Ewald_qe()
ENDIF
```

- The following call will initialize various global variables (arrays) needed to define Hamiltonian. It is mainly used for storing potential terms.

```
CALL alloc_hamiltonian()
```

- Allocate local pseudopotential. For periodic sinc LF the potential is constructed directly on real space grid. For periodic LF, the potential is first constructed on reciprocal space and then transformed to real space grid via inverse FFT.

```
IF( assume_isolated == 'sinc' ) THEN
  CALL init_V_ps_loc()
ELSE
  CALL init_V_ps_loc_G()
ENDIF
```

- Laplacian matrix ∇^2 is initialized here.

```
CALL init_nabla2_sparse()
```

- Here we construct ILU0 preconditioner based on kinetic matrix.

```
CALL init_ilu0_prec()
```

- If the option `FREE_NABLA2` is `.TRUE.`, then memory for storing Laplacian matrix is freed immediately. In this way, application of kinetic operator will be done by a matrix-free algorithm instead of using sparse-matrix multiplication.

```
IF( FREE_NABLA2 ) THEN
  CALL dealloc_nabla2_sparse()
ENDIF
```

- This is a special setup for Poisson solver in the case of non-periodic system. Currrently there are two different methods: (1) interpolating scaling function (ISF) method and (2) Direct Algorithm for Gravitation and Electrostatic (DAGE).

```
IF( I_POISSON_SOLVE == 1 ) THEN
  CALL init_Poisson_solve_ISF()
ELSEIF( I_POISSON_SOLVE == 2 ) THEN
  CALL init_Poisson_solve_DAGE()
ENDIF
```

```
END SUBROUTINE
```

3 Subroutine guess_KS_solutions

Generate guess solutions (density and orbitals) for `KS_solve_XXX` subroutines.

Note that the global arrays `KS_evals` and `KS_evecs` are allocated in this subroutine.

The algorithm used in this subroutine is not sophisticated. I have used ABINIT routine **`atmlength`** to generate gaussian charge density.

For direct minimization (using `KS_solve_Emin_XXX`), we need to generate initial wavefunction, so an iterative diagonalization step must be performed. This might cause significant additional time before the actual direct minimization step starts.

For SCF, we can simply generate random initial wavefunction which will be used for iterative diagonalization step.

```
SUBROUTINE guess_KS_solutions()
```

Here are the imported variables:

```
  USE m_input_vars, ONLY : startingwfc
```

- Generate gaussian charge density.

```
  ! FIXME: Need gen_guess_rho_gaussian for isolated system ?
  !        thus bypassing calculation of structure factors ?
  IF( startingwfc /= 'random' ) THEN
    CALL gen_guess_rho_gaussian()
  ENDIF
```

- Generate random initial wavefunction. This step is needed for both SCF and direct minimization method.

```
  CALL gen_random_evecs()  ! also needed for initial diagonalization routine
```

- If the input variable **`startingwfc`** is not set to **`'random'`** then we need to do one iterative diagonalization step. If it is set to random then we don't need to do iterative diagonalization step as random wavefunction is suffice and subroutine will immediately return.

```
  IF( startingwfc /= 'random' ) THEN
    ! This will call diagonalization routine
    CALL gen_gaussian_evecs()
  ELSE
    WRITE(*,*)
    WRITE(*,*) 'Using random starting wavefunction'
  ENDIF
END SUBROUTINE
```

4 Subroutine do_KS_solve

A driver routine for solving Kohn-Sham equations.

```
SUBROUTINE do_KS_solve()
  USE m_constants, ONLY : Ry2eV
  USE m_input_vars, ONLY : startingwfc
  USE m_options, ONLY : I_KS_Solve
  USE m_states, ONLY : Nstates, Focc, &
                       evals => KS_evals, &
                       evecs => KS_evecs
  IMPLICIT NONE
  INTEGER :: ist
```

We are using direct minimization (this is the default):

```
  IF( I_KS_SOLVE == 1 ) THEN
```

Call the main computational routine for direct minimization. Note that for the moment the parameter α_t is hardcoded to 3×10^{-5}

```
    CALL KS_solve_Emin_pcg( 3.d-5, .FALSE. )
```

Display total energy components

```
    CALL info_energies()
```

We need to calculate eigenvalues explicitly

```
    CALL calc_evals( Nstates, Focc, evecs, evals )
```

We are using the conventional SCF with electron density mixing

```
  ELSEIF( I_KS_SOLVE == 2 ) THEN
```

We need to calculate electron density and update the local potentials if starting from random wavefunction

```
    IF( startingwfc == 'random' ) THEN
      ! Initial Rhoe and potentials
      CALL calc_rhoe( Focc, evecs )
      ! FIXME: need this ?
      CALL update_potentials()
    ENDIF
```

Call the main computational routine for SCF

```
    CALL KS_solve_SCF()
    CALL info_energies()
  ENDIF
```

Eigenvalues are displayed here

```
  WRITE(*,*)
  WRITE(*,*) 'Final eigenvalues (Ha and eV)'
  WRITE(*,*)
  DO ist = 1,Nstates
    WRITE(*,'(1x,I8,2F18.10)') ist, evals(ist), evals(ist)*2.d0*Ry2eV
  ENDDO
```

Write restart data

```
  !FIXME Need tidy up
  CALL write_checkpoint()
  CALL write_KS_evecs('KS_evecs.dat')
END SUBROUTINE
```

5 Subroutine `cleanup_ffr_LFDFT`

This subroutine is the driver for various `dealloc_XX` subroutines.

```
SUBROUTINE cleanup_ffr_LFDFT()
```

Currently, only this option is needed.

```
  USE m_options, ONLY : I_POISSON_SOLVE
  IMPLICIT NONE
```

This call will deallocate Kohn-Sham eigenvalues and eigenvectors

```
  CALL dealloc_states()
  IF( I_POISSON_SOLVE == 1 ) THEN
    CALL dealloc_Poisson_solve_ISF()
  ELSEIF( I_POISSON_SOLVE == 2 ) THEN
    CALL dealloc_Poisson_solve_DAGE()
  ENDIF
  CALL dealloc_nabla2_sparse()
  CALL dealloc_ilu0_prec()
  CALL dealloc_hamiltonian()
  CALL dealloc_LF3d()
  CALL dealloc_PsPot()
  CALL dealloc_atoms()
END SUBROUTINE
```

6 Function `eval_LF1d_sinc`

Evaluate Lagrange-sinc function:

$$\phi_\alpha(x) = \frac{1}{\sqrt{h}} \frac{\sin\left[\pi(x - x_\alpha)/h\right]}{\pi(x - x_\alpha)/h} \tag{1}$$

where h is grid spacing.

```
FUNCTION eval_LF1d_sinc( N, grid, ibf, x ) RESULT(ff)
  USE m_constants, ONLY : PI
  IMPLICIT NONE
  !
  INTEGER :: N
  REAL(8) :: grid(N)
  REAL(8) :: ff, x, dx
  INTEGER :: ibf
  REAL(8), PARAMETER :: SMALL = 1.d-10
  !
  REAL(8) :: h

  h = grid(2) - grid(1)
  dx = x - grid(ibf)
  IF( abs(dx) < SMALL ) THEN
    dx = SMALL
  ENDIF
  ff = sin( PI*dx/h ) / (PI*dx) * h /sqrt(h)
END FUNCTION
```

7 Function `eval_LF1d_p`

Evaluate periodic Lagrange function:

$$\phi_\alpha(x) = \frac{1}{\sqrt{NL}} \sum_{i=1}^{N} \cos\left[k_i(x - x_\alpha)\right] \tag{2}$$

with $i = 1, 2, \ldots, N$ and

$$k_i = \frac{2\pi(i - N' - 1)}{L} \tag{3}$$

$N' = (N - 1)/2$ and N must be an odd number.

```
FUNCTION eval_LF1d_p(N, L, grid, ibf, x) RESULT(ff)
  USE m_constants, ONLY : PI
  IMPLICIT NONE
  ! arguments
```

```
  INTEGER :: N
  REAL(8) :: L
  REAL(8) :: grid(N)
  INTEGER :: ibf
  REAL(8) :: x, ff
  !
  REAL(8) :: pre1
  INTEGER :: ii
  !
  pre1 = 1.d0/sqrt(N*L)
  ff = 0.d0
  DO ii = 1,N
    ff = ff + cos( PI*( 2*ii - N - 1 )*( x - grid(ibf) )/ L )
  ENDDO
  ff = ff*pre1
END FUNCTION
```

8 Function `eval_LF1d_c`

Evaluate cluster Lagrange function:

$$\phi_\alpha(x) = \frac{2}{\sqrt{(N+1)L}} \sum_{i=1}^{N} \sin(k_i x) \sin(k_i x_\alpha) \tag{4}$$

with $i = 1, 2, \ldots, N$ and

$$k_i = \frac{\pi i}{L} \tag{5}$$

```
FUNCTION eval_LF1d_c( N, L, A, grid, ibf, x ) RESULT(ff)
  USE m_constants, ONLY : PI
  IMPLICIT NONE
  !
  INTEGER :: N
  REAL(8) :: L
  REAL(8) :: A
  REAL(8) :: grid(N)
  REAL(8) :: ff
  INTEGER :: ibf
  REAL(8) :: x
  !
  REAL(8) :: ki, pre
  INTEGER :: ii
  pre = 2.d0/sqrt( (N+1)*L )
  ff = 0.d0
  DO ii = 1, N
    ki = PI*ii/L
    ff = ff + sin( ki*(grid(ibf) - A) ) * sin( ki*(x - A) )
  ENDDO
  ff = ff*pre
END FUNCTION
```

9 Subroutine `calc_betaNL_psi`

Calculate dot product between $\beta_{\mathrm{NL}}(\mathbf{r})$ and $\psi(\mathbf{r})$

```
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
```