PURPOSE:

This subroutine solves Kohn-Sham equations by minimizing total energy functional using conjugate gradient algorithm. The algorithm is based on T.A. Arias notes.

AUTHOR:

Fadjar Fathurrahman

MODIFIES:

Global variables KS_evecs and E_total

NOTES:

ILU0 preconditioner from SPARSKIT is used as preconditioner.

```
SUBROUTINE KS_solve_Emin_pcg( alpha_t, restart )
```

The following variables are imported.

```
  USE m_LF3d, ONLY : Npoints => LF3d_Npoints
  USE m_states, ONLY : Nstates, &
                       Focc, &
                       v => KS_evecs
  USE m_energies, ONLY : Etot => E_total

  USE m_options, ONLY : I_CG_BETA, Emin_NiterMax, Emin_ETOT_CONV_THR
  USE m_options, ONLY : T_WRITE_RESTART

  IMPLICIT NONE
  !
  REAL(8) :: alpha_t  ! step size
  LOGICAL :: restart
  REAL(8), ALLOCATABLE :: g(:,:), g_old(:,:), g_t(:,:)
  REAL(8), ALLOCATABLE :: d(:,:), d_old(:,:)
  REAL(8), ALLOCATABLE :: Kg(:,:), Kg_old(:,:) ! preconditioned
  REAL(8), ALLOCATABLE :: tv(:,:)
  REAL(8) :: alpha, beta, denum, Etot_old
  !
  INTEGER :: iter, ist
```

Display several informations about the algorithm

```
  CALL info_KS_solve_Emin_pcg( alpha_t, restart )

  ALLOCATE( g(Npoints,Nstates) )  ! gradient
  ALLOCATE( g_old(Npoints,Nstates) ) ! old gradient
  ALLOCATE( g_t(Npoints,Nstates) )  ! trial gradient
  ALLOCATE( d(Npoints,Nstates) )  ! direction
  ALLOCATE( d_old(Npoints,Nstates) )  ! old direction

  ALLOCATE( Kg(Npoints,Nstates) )  ! preconditioned gradient
  ALLOCATE( Kg_old(Npoints,Nstates) ) ! old preconditioned gradient

  ALLOCATE( tv(Npoints,Nstates) )  ! temporary vector for calculating trial gradient


  ! Read starting eigenvectors from file
  ! FIXME: This is no longer relevant
  IF( restart ) THEN
    READ(112) v   ! FIXME Need to use file name
  ENDIF

  CALL calc_Rhoe( Focc, v )
  CALL update_potentials()
```

```
  CALL calc_betaNL_psi( Nstates, v )
  CALL calc_energies( v )

  Etot_old = Etot

  alpha = 0.d0
  beta  = 0.d0

  g(:,:)     = 0.d0
  g_t(:,:)   = 0.d0
  d(:,:)     = 0.d0
  d_old(:,:) = 0.d0
  Kg(:,:)    = 0.d0
  Kg_old(:,:) = 0.d0

  DO iter = 1, Emin_NiterMax
    !
    ! Evaluate gradient at current trial vectors
    CALL calc_grad( Nstates, v, g )
    ! Precondition
    DO ist = 1, Nstates
      CALL prec_ilu0( g(:,ist), Kg(:,ist) )
    ENDDO
    !
    ! set search direction
    IF( iter /= 1 ) THEN
      SELECT CASE ( I_CG_BETA )
      CASE(1)
        ! Fletcher-Reeves
        beta = sum( g * Kg ) / sum( g_old * Kg_old )
      CASE(2)
        ! Polak-Ribiere
        beta = sum( (g-g_old)*Kg ) / sum( g_old * Kg_old )
      CASE(3)
        ! Hestenes-Stiefel
        beta = sum( (g-g_old)*Kg ) / sum( (g-g_old)*d_old )
      CASE(4)
        ! Dai-Yuan
        beta = sum( g * Kg ) / sum( (g-g_old)*d_old )
      END SELECT
    ENDIF
    IF( beta < 0 ) THEN
      WRITE(*,'(1x,A,F18.10,A)') 'beta is smaller than zero: ', beta, ': setting it to zero'
    ENDIF
    beta = max( 0.d0, beta )
    d(:,:) = -Kg(:,:) + beta*d_old(:,:)
    !
    ! Evaluate gradient at trial step
    tv(:,:) = v(:,:) + alpha_t * d(:,:)
    CALL orthonormalize( Nstates, tv )

    CALL calc_Rhoe( Focc, tv )
    CALL update_potentials()  ! Now global vars on m_hamiltonian are changed
    CALL calc_betaNL_psi( Nstates, tv )
    CALL calc_grad( Nstates, tv, g_t )
    !
    ! Compute estimate of best step and update current trial vectors
    denum = sum( (g - g_t) * d )
    IF( denum /= 0.d0 ) THEN  ! FIXME: use abs ?
      alpha = abs( alpha_t * sum( g * d )/denum )
```

```
    ELSE
      alpha = 0.d0
    ENDIF
    !WRITE(*,*) 'iter, alpha_t, alpha, beta', iter, alpha_t, alpha, beta

    v(:,:) = v(:,:) + alpha * d(:,:)
    CALL orthonormalize( Nstates, v )

    CALL calc_Rhoe( Focc, v )
    CALL update_potentials()
    CALL calc_betaNL_psi( Nstates, v )
    CALL calc_energies( v )
    !
    WRITE(*,'(1x,I5,F18.10,ES18.10)') iter, Etot, Etot_old-Etot
    !
    IF( abs(Etot - Etot_old) < Emin_ETOT_CONV_THR ) THEN
      WRITE(*,*) 'KS_solve_Emin_pcg converged in iter', iter
      EXIT
    ENDIF
    !
    Etot_old = Etot
    g_old(:,:) = g(:,:)
    d_old(:,:) = d(:,:)
    Kg_old(:,:) = Kg(:,:)
    flush(6)
  ENDDO

  IF( T_WRITE_RESTART ) THEN
    WRITE(111) v
  ENDIF

  DEALLOCATE( g, g_old, g_t, d, d_old, tv, Kg, Kg_old )
END SUBROUTINE


SUBROUTINE info_KS_solve_Emin_pcg( alpha_t, restart )
  USE m_options, ONLY : I_CG_BETA, Emin_NiterMax, Emin_ETOT_CONV_THR
  USE m_LF3d, ONLY : Npoints => LF3d_Npoints
  USE m_states, ONLY : Nstates
  IMPLICIT NONE
  REAL(8) :: alpha_t
  LOGICAL :: restart
  !
  REAL(8) :: memGB

  memGB = Npoints*Nstates*8d0 * 8d0 / (1024d0*1024d0*1024.d0)

  WRITE(*,*)
  WRITE(*,*) 'Minimizing KS total energy functional using PCG algorithm'
  WRITE(*,*) '------------------------------------------------------------'
  WRITE(*,*)
  WRITE(*,'(1x,A,I8)')     'NiterMax = ', Emin_NiterMax
  WRITE(*,'(1x,A,ES10.3)') 'alpha_t  = ', alpha_t
  WRITE(*,*)               'restart  = ', restart
  WRITE(*,'(1x,A,ES10.3)') 'conv_thr = ', Emin_ETOT_CONV_THR
  WRITE(*,*)
  IF( I_CG_BETA == 1 ) THEN
    WRITE(*,*) 'Using Fletcher-Reeves formula'
  ELSEIF( I_CG_BETA == 2 ) THEN
    WRITE(*,*) 'Using Polak-Ribiere formula'
```

```
  ELSEIF( I_CG_BETA == 3 ) THEN
    WRITE(*,*) 'Using Hestenes-Stiefel formula'
  ELSEIF( I_CG_BETA == 4 ) THEN
    WRITE(*,*) 'Using Dai-Yuan formula'
  ELSE
    WRITE(*,*) 'XXXXX WARNING: Unknown I_CG_BETA: ', I_CG_BETA
  ENDIF
  WRITE(*,*)
  WRITE(*,'(1x,A,F18.10)') 'KS_solve_Emin_pcg: memGB = ', memGB
  WRITE(*,*)

END SUBROUTINE
```