Learning QE programming in 18 easy steps-2nd Day

P. Umari
Università degli Studi di Padova, Italy
CNR-IOM Democritos, Trieste, Italy

Step 9: Reading wfcs and checking orthonormality in G, space: Γ-point only case with US pseudo-potentials

We have the relation:

$$\langle \psi_i | \, S \, | \psi_j \rangle = \delta_{ij}$$

Pseudo species of atom *I*

$$S = 1 + \sum_{I,\alpha,\beta} \left| \beta^{S(I)}_{\mathbf{R}_I \alpha} \right\rangle Q^{S(I)}_{\alpha\beta} \left\langle \beta^{S(I)}_{\mathbf{R}_I \beta} \right|$$

atom-index

angular momentum channels

Step 9: Reading wfcs and checking orthonormality in G, space: Γ-point only case with US pseudo-potentials

okvan LOGICAL .true. if we have US pseudos
nkb INTEGER total number of projectors (including NC pseudos)
vkb COMPLEX (npwx,nkb) betae projectors (already allocated)

xk REAL (3,nks)
coordinate in crystal units of k-points

```
USE uspp,     ONLY : nkb, vkb,okvan
USE becmod,       ONLY : bec_type, becp, calbec,allocate_bec_type, deallocate_bec_type
USE klist,            ONLY : xk

...
if(okvan) then
        CALL allocate_bec_type (nkb,nbnd,becp)
         IF ( nkb > 0 ) &
            CALL init_us_2( npw, igk, xk(1,1), vkb )
         CALL calbec( npw, vkb, evc, becp )
       endif

...
if(okvan) CALL deallocate_bec_type ( becp )
```

bec_type TYPE descriptor for betae/psi scalar products
becp TYPE(bec_type) global variable for beate/psi products

allocate_bec_type deallocate_bec_type
allocation deallocation routines

init_us_2
betae depend on k-point and must be initialized (also for NC pseudos)

calbec
calculates the betae/psi scalar products

Step 9: Reading wfcs and checking orthonormality in G, space: Γ-point only case with US pseudo-potentials

nat INTEGER number of atoms
nsp INTEGER number of different species (pseudos)
ityp INTEGER (nat) correspondence table atom->pseudo species

```
USE ions_base,  ONLY : nat, nsp, ityp
USE uspp_param, ONLY : upf, nh,nhm
USE uspp,     ONLY :  qq

•••
!US part
      if(okvan) then
         ijkb0 = 0
         DO nt = 1, nsp
           IF ( upf(nt)%tvanp ) THEN
             DO na = 1, nat
               IF ( ityp(na) == nt ) THEN

                 DO jh = 1, nh(nt)
                   jkb = ijkb0 + jh
                   DO ih = 1, nh(nt)
                     ikb = ijkb0 + ih
                     do i=1,nbnd
                       do j=1,nbnd
                         omat(i,j)=omat(i,j)+qq(ih,jh,nt)*becp%r(ikb,i)*becp%r(jkb,j)
                       enddo
                     enddo

                   END DO
                 END DO
                 ijkb0 = ijkb0 + nh(nt)
               END IF
             END DO
           ELSE
             DO na = 1, nat
               IF ( ityp(na) == nt ) ijkb0 = ijkb0 + nh(nt)
             END DO
           END IF
         END DO

      endif

•••
```

upf TYPE(pseudo_upf) (nat) descriptor for pseudo at each atom
nh INTEGER (nsp) number of betae projectors for each species (pseudos)
nhm INTEGER (nsp) max number of betae projectors for any species (pseudos)

Text

qq REAL(nhm,nhm,nsp) Q terms for each US pseudo

becp%r REAL(nkb,nbnd) betae/psi scalar products

Look at the correct order!

Step 10: Reading wfcs and checking orthonormality in G, space: k-points case with US pseudo-potentials

Analogous to Step 9 with just two differences:

Note dependence on *ik*-th k-point

```
....

if(okvan) then
        IF ( nkb > 0 ) &
            CALL init_us_2( npw, igk, xk(1,ik), vkb )
        CALL calbec( npw, vkb, evc, becp )
endif

...
zmat(i,j)=zmat(i,j)+qq(ih,jh,nt)*conjg(becp%k(ikb,i))*becp%k(jkb,j)
...
```

becp%k COMPLEX(nkb,nbnd) betae/psi scalar products

Step 11: Reading wfcs and checking orthonormality in R, space: Γ-point only case with US pseudo-potentials

$$\langle\psi_i|\ \widetilde{\mathbf{r}\rangle\langle\mathbf{r}}\ |\psi_j\rangle = \psi_i^*(\mathbf{r})\,\psi_j(\mathbf{r}) + \sum_{I,\alpha,\beta}\langle\psi_i\ \left|\beta^{S(I)}_{\mathbf{R}_I\alpha}\right\rangle q^{S(I),\mathbf{R}_I}_{\alpha\beta}(\mathbf{r})\left\langle\beta^{S(I)}_{\mathbf{R}_I\beta}\right|\ \psi_j\rangle$$

Position operator in US formalism

Augmentation charge

Step 11: Reading wfcs and checking orthonormality in R, space: Γ-point only case with US pseudo-potentials

US pseudo-potentials require a denser grid for the CHARGE which is defined by *ecutrho*

```
USE fft_base,         ONLY : dffts,dfftp
...
real(kind=DP), allocatable :: grids(:),gridd(:)
allocate(grids(dffts%nnr),gridd(dfftp%nnr))
...
CALL init_us_1
...
 do i=1,nbnd
        do j=1,nbnd
          grids(1:dffts%nnr)=rwfcs(1:dffts%nnr,i)*rwfcs(1:dffts%nnr,j)
          call interpolate (gridd, grids, 1)
          call addusterm(gridd,becp,i,j,nbnd)
...
    end do
end do
...
```

USE fft_base, ONLY : dfftp TYPE (fft_dlay_descriptor) descriptor for dense grid

call interpolate (gridd, grids, 1)
interpolate from smooth grid *grids* to dense grid *gridd*
call interpolate (gridd, grids, -1)
interpolate from dense grid *gridd* to smooth grid *grids*

CALL init_us_1
Initialization of US stuff

We must add the augmentation charge, you should write this routine, similar to PW/src/addusdens.f90

Step 11: Reading wfcs and checking orthonormality in R space

```
!-----------------------------------------------------------------------
subroutine addusterm(gridd,bcf,i,j,nbands)
  !-----------------------------------------------------------------------
  !
  !  This routine adds to the charge density the part which is due to
  !  the US augmentation.
  !
  USE kinds,                ONLY : DP
  USE ions_base,            ONLY : nat, ntyp => nsp, ityp
  USE fft_base,             ONLY : dfftp
  USE fft_interfaces,       ONLY : invfft
  USE gvect,                ONLY : ngm, nl, nlm, gg, g, &
                                   eigts1, eigts2, eigts3, mill
  USE uspp,                 ONLY : okvan
  USE uspp_param,           ONLY : upf, lmaxq, nh,nhm
  USE control_flags,        ONLY : gamma_only
  USE wavefunctions_module, ONLY : psic
  USE becmod,       ONLY : bec_type
  USE io_global, ONLY : stdout
  USE cell_base,    ONLY : omega
 !
  implicit none
  !
  REAL(kind=dp), intent(inout) :: gridd(dfftp%nnr)
  TYPE(bec_type), intent(in) :: bcf
  INTEGER, intent(in) :: i,j
  INTEGER, intent(in) :: nbands
  !
  !    here the local variables
  !
  integer :: ig, na, nt, ih, jh, is
  ! counters

  real(DP), allocatable :: qmod (:), ylmk0 (:,:)
  ! the modulus of G
  ! the spherical harmonics

  complex(DP) :: skk
  complex(DP), allocatable ::  aux (:), qgm(:)
   ! Fourier transform of q
  integer :: ijkb0,ikb,jkb,ii
  real(kind=DP), allocatable :: bec2(:,:,:)

  if (.not.okvan) return

  allocate(bec2(nhm,nat,nbands))
  call start_clock ('addusdens')

  allocate (aux ( ngm))
  allocate (qmod( ngm))
  allocate (qgm( ngm))
  allocate (ylmk0( ngm, lmaxq * lmaxq))

  aux (:) = (0.d0, 0.d0)
  call ylmr2 (lmaxq * lmaxq, ngm, g, gg, ylmk0)
```

Set up spherical harmonics

arrange order of beate/psi products

Calculate q(r)

Multiply with scalar products and with structure factors

Do fft to bring in real space!

```
do ig = 1, ngm
   qmod (ig) = sqrt (gg (ig) )
 enddo
 bec2=0.d0
 ijkb0 = 0
  DO nt = 1, ntyp
    IF ( upf(nt)%tvanp ) THEN
      DO na = 1, nat
        IF ( ityp(na) == nt ) THEN
          DO jh = 1, nh(nt)
            jkb = ijkb0 + jh
            do ii=1,nbands
              bec2(jh,na,ii)=bcf%r(jkb,ii)
            enddo
          END DO
          ijkb0 = ijkb0 + nh(nt)
        END IF
      END DO
    ELSE
      DO na = 1, nat
        IF ( ityp(na) == nt ) ijkb0 = ijkb0 + nh(nt)
      END DO
    END IF
  END DO
 do nt = 1, ntyp
   if ( upf(nt)%tvanp ) then

     do ih = 1, nh (nt)
       do jh = 1,nh (nt)

         call qvan2 (ngm, ih, jh, nt, qmod, qgm, ylmk0)


           do na = 1, nat

             if (ityp (na) .eq.nt) then


               do ig = 1, ngm
                 skk = eigts1 (mill (1,ig), na) * &
                       eigts2 (mill (2,ig), na) * &
                       eigts3 (mill (3,ig), na)
                 aux(ig)=aux(ig) + qgm(ig)*skk*omega*bec2(ih,na,i)*bec2(jh,na,j)
               enddo

             endif
           enddo
         enddo
       enddo
     endif
   enddo
 !
   deallocate (ylmk0)
   deallocate (qgm)
   deallocate (qmod)
 !
 !    convert aux to real space and add to the charge density
 !
   psic(:) = (0.d0, 0.d0)
   psic( nl(1:ngm) ) = aux(1:ngm)
   if (gamma_only) psic( nlm(1:ngm) ) = CONJG(aux(1:ngm))
   CALL invfft ('Dense', psic, dfftp)
   gridd(1:dfftp%nnr) = gridd(1:dfftp%nnr) +  DBLE (psic (1:dfftp%nnr) )

  deallocate (aux)
  deallocate(bec2)
  call stop_clock ('addusdens')
  return
end subroutine addusterm
```

Step 12: Reading wfcs and checking orthonormality in R, space: k-points case with US pseudo-potentials

```
•••
complex(kind=DP), allocatable :: gridc(:)

•••
 do i=1,nbnd
         do j=1,nbnd
           grids(1:dffts%nnr)=dble(conjg(cwfcs(1:dffts%nnr,i))*cwfcs(1:dffts%nnr,j))
           call interpolate (gridd, grids, 1)
           gridc(1:dfftp%nnr)=gridd(1:dfftp%nnr)
           grids(1:dffts%nnr)=dimag(conjg(cwfcs(1:dffts%nnr,i))*cwfcs(1:dffts%nnr,j))
           call interpolate (gridd, grids, 1)
           gridc(1:dfftp%nnr)=gridc(1:dfftp%nnr)+(0.d0,1.d0)*gridd(1:dfftp%nnr)
           call addustermk(gridc,becp,i,j,nbnd)

•••
   enddo
enddo

•••
```

Now *gridc* is complex, but interpolate works with REAL(DP) arrays

We must add the augmentation charge, you should write this routine, similar to PW/src/addusdens.f90

Step 12:continued

```
!-----------------------------------------------------------------------
subroutine addustermk(gridd,bcf,i,j,nbands)
 !-----------------------------------------------------------------------
 !
 !  This routine adds to the charge density the part which is due to
 !  the US augmentation.
 !
 USE kinds,                ONLY : DP
 USE ions_base,            ONLY : nat, ntyp => nsp, ityp
 USE fft_base,             ONLY : dfftp
 USE fft_interfaces,       ONLY : invfft
 USE gvect,                ONLY : ngm, nl, nlm, gg, g, &
                                  eigts1, eigts2, eigts3, mill
 USE uspp,                 ONLY : okvan
 USE uspp_param,           ONLY : upf, lmaxq, nh,nhm
 USE control_flags,        ONLY : gamma_only
 USE wavefunctions_module, ONLY : psic
 USE becmod,       ONLY : bec_type
 USE io_global, ONLY : stdout
 USE cell_base,    ONLY : omega
!
 implicit none
 !
 COMPLEX(kind=dp), intent(inout) :: gridd(dfftp%nnr)
 TYPE(bec_type), intent(in) :: bcf
 INTEGER, intent(in) :: i,j
 INTEGER, intent(in) :: nbands
 !
 !    here the local variables
 !

 integer :: ig, na, nt, ih, jh, is
 ! counters

 real(DP), allocatable :: qmod (:), ylmk0 (:,:)
 ! the modulus of G
 ! the spherical harmonics

 complex(DP) :: skk
 complex(DP), allocatable ::  aux (:), qgm(:)
  ! Fourier transform of q
 integer :: ijkb0,ikb,jkb,ii
 complex(kind=DP), allocatable :: bec2(:,:,:)

 if (.not.okvan) return

 allocate(bec2(nhm,nat,nbands))
 call start_clock ('addusdens')

 allocate (aux ( ngm))
 allocate (qmod( ngm))
 allocate (qgm( ngm))
 allocate (ylmk0( ngm, lmaxq * lmaxq))

 aux (:) = (0.d0, 0.d0)
 call ylmr2 (lmaxq * lmaxq, ngm, g, gg, ylmk0)
```

Note the difference
wrt the Gamma-only case

```
do ig = 1, ngm
   qmod (ig) = sqrt (gg (ig) )
enddo
bec2=0.d0
ijkb0 = 0
 DO nt = 1, ntyp
    IF ( upf(nt)%tvanp ) THEN
       DO na = 1, nat
          IF ( ityp(na) == nt ) THEN
             DO jh = 1, nh(nt)
                jkb = ijkb0 + jh
                do ii=1,nbands
                   bec2(jh,na,ii)=bcf%k(jkb,ii)
                enddo
             END DO
             ijkb0 = ijkb0 + nh(nt)
          END IF
       END DO
    ELSE
       DO na = 1, nat
          IF ( ityp(na) == nt ) ijkb0 = ijkb0 + nh(nt)
       END DO
    END IF
 END DO

 do nt = 1, ntyp
    if ( upf(nt)%tvanp ) then

       do ih = 1, nh (nt)
          do jh = 1,nh (nt)

             call qvan2 (ngm, ih, jh, nt, qmod, qgm, ylmk0)


             do na = 1, nat

                if (ityp (na) .eq.nt) then


                   do ig = 1, ngm
                      skk = eigts1 (mill (1,ig), na) * &
                            eigts2 (mill (2,ig), na) * &
                            eigts3 (mill (3,ig), na)
                      aux(ig)=aux(ig) + qgm(ig)*skk*omega*conjg(bec2(ih,na,i))*bec2(jh,na,j)
                   enddo

                endif
             enddo
          enddo
       enddo
    endif
 enddo
!
 deallocate (ylmk0)
 deallocate (qgm)
 deallocate (qmod)
!
!    convert aux to real space and add to the charge density
!

 psic(:) = (0.d0, 0.d0)
 psic( nl(1:ngm) ) = aux(1:ngm)
 CALL invfft ('Dense', psic, dfftp)
 gridd(1:dfftp%nnr) = gridd(1:dfftp%nnr) +  psic (1:dfftp%nnr)

 deallocate (aux)
 deallocate(bec2)
 call stop_clock ('addusdens')
 return
end subroutine addustermk
```