ecalj/fpgw/ code document

0.1

Generated by Doxygen 1.8.6

Fri Feb 5 2016 20:44:56

Contents

Chapter 1

Data Type Index

1.1 Data Types List

Here are the data types with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Type Documentation

3.1 m_freq Module Reference

Frequency mesh generator.

Public Member Functions

- subroutine getfreq (epsmode, realomega, imagomega, tetra, omg2max, nw_input, niw, ua, mpi__root)

  Get data set for m_freq. All arguments are input.

Public Attributes

- real(8), dimension(:), allocatable frhis
- real(8), dimension(:), allocatable freq_r
- real(8), dimension(:), allocatable freq_i
- real(8), dimension(:), allocatable wiw
- integer nwhis
- integer npm
- integer nw_i
- integer nw

3.1.1 Detailed Description

Frequency mesh generator.

- OUTPUT
  - fhris :histgram bins to accumlate im part
  - freq_r: omega along real axis
  - freq_i: omega along imag axis
  - wiw: integration weight along im axis
  - npm: npm=1 means only positive omega;npm=2 means positive and negative omega.
- NOTE: change of frequency mesh defined here may destroy consistency or not. Need check

Definition at line 9 of file m_freq.F.

3.1.2 Member Function/Subroutine Documentation

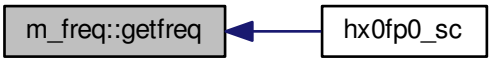
3.1.2.1 subroutine m_freq::getfreq (logical, intent(in) *epsmode,* logical, intent(in) *realomega,* logical, intent(in) *imagomega,* logical, intent(in) *tetra,* real(8), intent(in) *omg2max,* integer, intent(in) *nw_input,* integer, intent(in) *niw,* real(8), intent(in) *ua,* logical, intent(in) *mpi__root*)

Get data set for m_freq. All arguments are input.

- This read GWinput (dw,omg_c) and TimeReversal()
- All arguments are input

Definition at line 17 of file m_freq.F.

Here is the caller graph for this function:



3.1.3 Member Data Documentation

3.1.3.1 real(8), dimension(:), allocatable m_freq::freq_i

Definition at line 10 of file m_freq.F.

3.1.3.2 real(8), dimension(:), allocatable m_freq::freq_r

Definition at line 10 of file m_freq.F.

3.1.3.3 real(8), dimension(:), allocatable m_freq::frhis

Definition at line 10 of file m_freq.F.

3.1.3.4 integer m_freq::npm

Definition at line 11 of file m_freq.F.

3.1.3.5 integer m_freq::nw

Definition at line 11 of file m_freq.F.

3.1.3.6 integer m_freq::nw_i

Definition at line 11 of file m_freq.F.

3.1.3.7 integer m_freq::nwhis

Definition at line 11 of file m_freq.F.

3.1.3.8 real(8), dimension(:), allocatable m_freq::wiw

Definition at line 10 of file m_freq.F.

The documentation for this module was generated from the following file:

- gwsrc/m_freq.F

3.2 m_genallcf_v3 Module Reference

get basic settings of crystal structure and nlm info

Public Member Functions

- subroutine genallcf_v3 (nwin, efin, incwfx)

Public Attributes

- character(120) symgrp
- character(6), dimension(:), allocatable clabl
- integer, dimension(:), allocatable iclass
- real(8) alat
- real(8) ef
- real(8) diw
- real(8) dw
- real(8) delta
- real(8) deltaw
- real(8) esmr
- logical done_genallcf_v3 =.false.
- character(8), dimension(:), allocatable spid

3.2.1 Detailed Description

get basic settings of crystal structure and nlm info

- genallcf_v3(nwin,efin,incwfx) set data
- This is old routine. Confusing. We need to clean up.

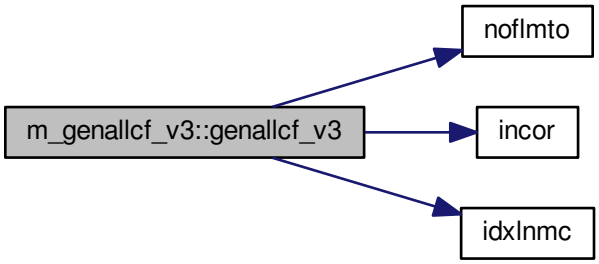
Definition at line 4 of file genallcf_mod.F.

3.2.2 Member Function/Subroutine Documentation

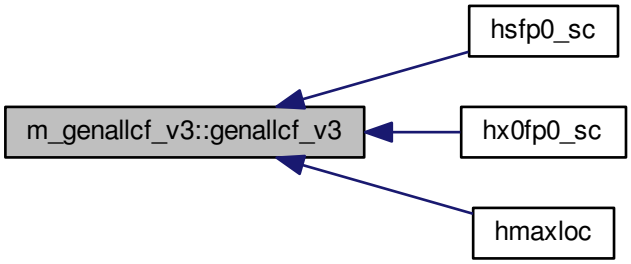
3.2.2.1 subroutine m_genallcf_v3::genallcf_v3 (integer(4) *nwin*, real(8) *efin*, integer(4) *incwfx*)

Definition at line 48 of file genallcf_mod.F.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.3 Member Data Documentation

3.2.3.1 real(8) m_genallcf_v3::alat

Definition at line 42 of file genallcf_mod.F.

3.2.3.2 character(6), dimension(:), allocatable m_genallcf_v3::clabl

Definition at line 27 of file genallcf_mod.F.

3.2.3.3 real(8) m_genallcf_v3::delta

Definition at line 42 of file genallcf_mod.F.

3.2.3.4 real(8) m_genallcf_v3::deltaw

Definition at line 42 of file genallcf_mod.F.

3.2.3.5 real(8) m_genallcf_v3::diw

Definition at line 42 of file genallcf_mod.F.

3.2.3.6 logical m_genallcf_v3::done_genallcf_v3 =.false.

Definition at line 43 of file genallcf_mod.F.

3.2.3.7 real(8) m_genallcf_v3::dw

Definition at line 42 of file genallcf_mod.F.

3.2.3.8 real(8) m_genallcf_v3::ef

Definition at line 42 of file genallcf_mod.F.

3.2.3.9 real(8) m_genallcf_v3::esmr

Definition at line 42 of file genallcf_mod.F.

3.2.3.10 integer, dimension(:), allocatable m_genallcf_v3::iclass

Definition at line 28 of file genallcf_mod.F.

3.2.3.11 character(8), dimension(:), allocatable m_genallcf_v3::spid

Definition at line 44 of file genallcf_mod.F.

3.2.3.12 character(120) m_genallcf_v3::symgrp

Definition at line 26 of file genallcf_mod.F.

The documentation for this module was generated from the following file:

- gwsrc/genallcf_mod.F

3.3 m_hamindex Module Reference

This is in lm7K/subs/m_hamindex.F and in fpgw/gwsrc/m_hamindex.F We will need to unify make system and source code in fpgw and lmf. norbtx is given in gwsrc/readeigen.F init_readeigen2.

Public Member Functions

- integer function getikt (qin)

  get index ikt such that for qin(:)=qq(:,ikt)

- subroutine writehamindex ()

  write info for wave rotation.
- subroutine readhamindex ()

  read info for wave rotation.

Public Attributes

- integer, parameter null =-999999
- integer ngrp =null
- integer lxx =null
- integer kxx =null
- integer norbmto =null
- integer norbtx =null
- integer imx =null
- integer nbas
- integer ndimham =null
- integer nqtt
- integer nqi
- integer nqnum
- integer ngpmx
- integer, dimension(:), allocatable ltab
- integer, dimension(:), allocatable ktab
- integer, dimension(:), allocatable offl
- integer, dimension(:), allocatable ispec
- integer, dimension(:), allocatable iclasst
- integer, dimension(:,:,:), allocatable offlrev
- integer, dimension(:), allocatable ibastab
- integer, dimension(:), allocatable iqimap
- integer, dimension(:), allocatable iqmap
- integer, dimension(:), allocatable igmap
- integer, dimension(:), allocatable invgx
- integer, dimension(:,:), allocatable miat
- integer, dimension(:), allocatable ibasindex
- real(8), dimension(:,:,:), allocatable symops
- real(8), dimension(:,:), allocatable ag
- real(8), dimension(:,:,:), allocatable tiat
- real(8), dimension(:,:), allocatable shtvg
- real(8), dimension(:,:,:,:), allocatable dlmm
- real(8), dimension(:,:), allocatable qq
- real(8), dimension(3, 3) plat
- real(8), dimension(3, 3) qlat
- real(8), dimension(:,:), allocatable qtt
- real(8), dimension(:,:), allocatable qtti
- integer, dimension(:,:,:), allocatable igv2

- integer, dimension(:), allocatable napwk
- integer, dimension(:,:,:,:), allocatable igv2rev
- integer napwmx =null
- integer lxxa =null

Private Attributes

- logical, private debug =.false.

3.3.1 Detailed Description

This is in lm7K/subs/m_hamindex.F and in fpgw/gwsrc/m_hamindex.F We will need to unify make system and source code in fpgw and lmf. norbtx is given in gwsrc/readeigen.F init_readeigen2.

Definition at line 4 of file m_hamindex.F.

3.3.2 Member Function/Subroutine Documentation

3.3.2.1 integer function m_hamindex::getikt (real(8), dimension(3) *qin*)

get index ikt such that for qin(:)=qq(:,ikt)

Definition at line 21 of file m_hamindex.F.

3.3.2.2 subroutine m_hamindex::readhamindex ( )

read info for wave rotation.

Definition at line 62 of file m_hamindex.F.

3.3.2.3 subroutine m_hamindex::writehamindex ( )

write info for wave rotation.

Definition at line 40 of file m_hamindex.F.

3.3.3 Member Data Documentation

3.3.3.1 real(8), dimension(:,:), allocatable m_hamindex::ag

Definition at line 12 of file m_hamindex.F.

3.3.3.2 logical, private m_hamindex::debug =.false. [private]

Definition at line 17 of file m_hamindex.F.

3.3.3.3 real(8), dimension(:,:,:,:), allocatable m_hamindex::dlmm

Definition at line 12 of file m_hamindex.F.

3.3.3.4 integer, dimension(:), allocatable m_hamindex::ibasindex

Definition at line 11 of file m_hamindex.F.

3.3.3.5 integer, dimension(:), allocatable m_hamindex::ibastab

Definition at line 10 of file m_hamindex.F.

3.3.3.6 integer, dimension(:), allocatable m_hamindex::iclasst

Definition at line 10 of file m_hamindex.F.

3.3.3.7 integer, dimension(:), allocatable m_hamindex::igmap

Definition at line 11 of file m_hamindex.F.

3.3.3.8 integer, dimension(:,:,:), allocatable m_hamindex::igv2

Definition at line 15 of file m_hamindex.F.

3.3.3.9 integer, dimension(:,:,:,:), allocatable m_hamindex::igv2rev

Definition at line 15 of file m_hamindex.F.

3.3.3.10 integer m_hamindex::imx =null

Definition at line 8 of file m_hamindex.F.

3.3.3.11 integer, dimension(:), allocatable m_hamindex::invgx

Definition at line 11 of file m_hamindex.F.

3.3.3.12 integer, dimension(:), allocatable m_hamindex::iqimap

Definition at line 11 of file m_hamindex.F.

3.3.3.13 integer, dimension(:), allocatable m_hamindex::iqmap

Definition at line 11 of file m_hamindex.F.

3.3.3.14 integer, dimension(:), allocatable m_hamindex::ispec

Definition at line 10 of file m_hamindex.F.

3.3.3.15 integer, dimension(:), allocatable m_hamindex::ktab

Definition at line 10 of file m_hamindex.F.

3.3.3.16 integer m_hamindex::kxx =null

Definition at line 7 of file m_hamindex.F.

3.3.3.17 integer, dimension(:), allocatable m_hamindex::ltab

Definition at line 10 of file m_hamindex.F.

3.3.3.18 integer m_hamindex::lxx =null

Definition at line 7 of file m_hamindex.F.

3.3.3.19 integer m_hamindex::lxxa =null

Definition at line 16 of file m_hamindex.F.

3.3.3.20 integer, dimension(:,:), allocatable m_hamindex::miat

Definition at line 11 of file m_hamindex.F.

3.3.3.21 integer, dimension(:), allocatable m_hamindex::napwk

Definition at line 15 of file m_hamindex.F.

3.3.3.22 integer m_hamindex::napwmx =null

Definition at line 16 of file m_hamindex.F.

3.3.3.23 integer m_hamindex::nbas

Definition at line 8 of file m_hamindex.F.

3.3.3.24 integer m_hamindex::ndimham =null

Definition at line 8 of file m_hamindex.F.

3.3.3.25 integer m_hamindex::ngpmx

Definition at line 9 of file m_hamindex.F.

3.3.3.26 integer m_hamindex::ngrp =null

Definition at line 7 of file m_hamindex.F.

3.3.3.27 integer m_hamindex::norbmto =null

Definition at line 7 of file m_hamindex.F.

3.3.3.28 integer m_hamindex::norbtx =null

Definition at line 7 of file m_hamindex.F.

3.3.3.29 integer m_hamindex::nqi

Definition at line 9 of file m_hamindex.F.

3.3.3.30 integer m_hamindex::nqnum

Definition at line 9 of file m_hamindex.F.

3.3.3.31 integer m_hamindex::nqtt

Definition at line 9 of file m_hamindex.F.

3.3.3.32 integer, parameter m_hamindex::null =-999999

Definition at line 6 of file m_hamindex.F.

3.3.3.33 integer, dimension(:), allocatable m_hamindex::offl

Definition at line 10 of file m_hamindex.F.

3.3.3.34 integer, dimension(:,:,:), allocatable m_hamindex::offlrev

Definition at line 10 of file m_hamindex.F.

3.3.3.35 real(8), dimension(3,3) m_hamindex::plat

Definition at line 13 of file m_hamindex.F.

3.3.3.36 real(8), dimension(3,3) m_hamindex::qlat

Definition at line 13 of file m_hamindex.F.

3.3.3.37 real(8), dimension(:,:), allocatable m_hamindex::qq

Definition at line 12 of file m_hamindex.F.

3.3.3.38 real(8), dimension(:,:), allocatable m_hamindex::qtt

Definition at line 14 of file m_hamindex.F.

3.3.3.39 real(8), dimension(:,:), allocatable m_hamindex::qtti

Definition at line 14 of file m_hamindex.F.

3.3.3.40 real(8), dimension(:,:), allocatable m_hamindex::shtvg

Definition at line 12 of file m_hamindex.F.

3.3.3.41 real(8), dimension(:,:,:), allocatable m_hamindex::symops

Definition at line 12 of file m_hamindex.F.

3.3.3.42 real(8), dimension(:,:,:), allocatable m_hamindex::tiat

Definition at line 12 of file m_hamindex.F.

The documentation for this module was generated from the following file:

- gwsrc/m_hamindex.F

3.4 m_q0p Module Reference

Q0P (offset Gamma points) generator.

Public Member Functions

- subroutine getq0p (newoffsetG, alat, plat, qlat, n1q, n2q, n3q, alp, alpv, ngcxx, ngcx, nqbz, nqibz, nstbz, qbz, qibz, symops, ngrp, ngvect)

Public Attributes

- real(8), dimension(:,:), allocatable q0i
- real(8), dimension(:), allocatable wt
- integer nq0i

Private Attributes

- integer, private nq0x
- integer, private nmm

3.4.1 Detailed Description

Q0P (offset Gamma points) generator.

Definition at line 2 of file mkqg.F.

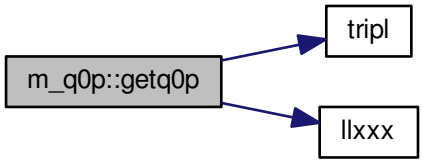
3.4.2 Member Function/Subroutine Documentation

3.4.2.1 subroutine m_q0p::getq0p (logical, intent(in) *newoffsetG,* real(8), intent(in) *alat,* real(8), dimension(3,3), intent(in) *plat,* real(8), dimension(3,3), intent(in) *qlat,* integer, intent(in) *n1q,* integer, intent(in) *n2q,* integer, intent(in) *n3q,* real(8), intent(in) *alp,* real(8), dimension(3), intent(in) *alpv,* integer, intent(in) *ngcxx,* integer, dimension(nqbz), intent(in) *ngcx,* integer, intent(in) *nqbz,* integer, intent(in) *nqibz,* integer, dimension(*), intent(in) *nstbz,* real(8), dimension(3,nqbz), intent(in) *qbz,* real(8), dimension(3,nqibz), intent(in) *qibz,* real(8), dimension(3,3,ngrp), intent(in) *symops,* integer, intent(in) *ngrp,* integer, dimension(3,ngcxx,nqbz), intent(in) *ngvect*)

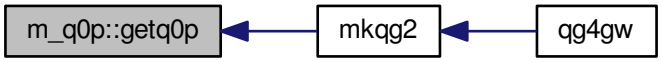
- Q0P data set is given for 'getq0p'

Definition at line 10 of file mkqg.F.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3 Member Data Documentation

3.4.3.1 integer, private m_q0p::nmm [private]

Definition at line 6 of file mkqg.F.

3.4.3.2 integer m_q0p::nq0i

Definition at line 5 of file mkqg.F.

3.4.3.3 integer, private m_q0p::nq0x [private]

Definition at line 6 of file mkqg.F.

3.4.3.4 real(8), dimension(:,:), allocatable m_q0p::q0i

Definition at line 4 of file mkqg.F.

3.4.3.5 real(8), dimension(:), allocatable m_q0p::wt

Definition at line 4 of file mkqg.F.

The documentation for this module was generated from the following file:

- gwsrc/mkqg.F

3.5 m_readqg Module Reference

Return QGcou and QGpsi ===.

Public Member Functions

- subroutine readngmx (key, ngmx)
- subroutine readqg (key, qin, ginv, qu, ngv, ngvec)

  Get ngv and ngvec(3,ngv) for given qin(3) key=='QGcou' or 'QGpsi'.
- subroutine readqg0 (key, qin, ginv, qu, ngv)

  Get ngv key=='QGcou' or 'QGpsi'.
- subroutine init_readqg (ifi, ginv)

  initialization. readin QGpsi or QGcou.
- subroutine tabkk (kkin, kktable, n, nout)
- subroutine iqindx2qg (q, ifi, iqindx, qu)

  Find index as q=qq(:,iq) with modulo of premitive vector. ginv is the inverse of plat (premitive translation vector). Use kk1,kk2,kk3,nkey(1:3),iqkkk to get iqindx.
- subroutine sortea (ea, ieaord, n, isig)

  mini-sort routine.
- subroutine iswap (i, j)

Private Attributes

- real(8), dimension(:,:), allocatable, target, private qc
- real(8), dimension(:,:), allocatable, target, private qp
- logical, dimension(2), private init =.true.
- real(8), private qpgcut_cou
- real(8), private qpgcut_psi
- integer(4), target, private nqnumc
- integer(4), target, private nqnump
- integer(4), target, private ngcmx
- integer(4), target, private ngpmx
- integer(4), dimension(:,:,:), allocatable, private ngvecp
- integer(4), dimension(:), allocatable, private ngp
- integer(4), dimension(:,:,:), allocatable, private ngvecc
- integer(4), dimension(:), allocatable, private ngc
- integer, pointer, private nqtt
- real(8), dimension(:,:), pointer, private qtt

- real(8), private epsd =1d-7
- integer, dimension(:), pointer, private nkey
- integer, dimension(:), pointer, private kk1
- integer, dimension(:), pointer, private kk2
- integer, dimension(:), pointer, private kk3
- integer, dimension(:,:,:), pointer, private iqkkk
- integer, dimension(3), target, private nkeyp
- integer, dimension(3), target, private nkeyc
- integer, dimension(:,:), allocatable, target, private keyp
- integer, dimension(:), allocatable, target, private kk1p
- integer, dimension(:), allocatable, target, private kk2p
- integer, dimension(:), allocatable, target, private kk3p
- integer, dimension(:,:,:), allocatable, target, private iqkkkp
- integer, dimension(:,:), allocatable, target, private keyc
- integer, dimension(:), allocatable, target, private kk1c
- integer, dimension(:), allocatable, target, private kk2c
- integer, dimension(:), allocatable, target, private kk3c
- integer, dimension(:,:,:), allocatable, target, private iqkkkc
- real(8), dimension(3, 3), private ginv_

3.5.1 Detailed Description

Return QGcou and QGpsi ===.

Definition at line 23 of file readqg.F.

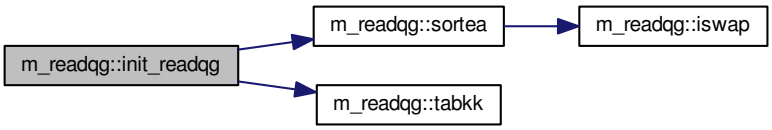
3.5.2 Member Function/Subroutine Documentation

3.5.2.1 subroutine m_readqg::init_readqg (integer(4), intent(in) *ifi,* real(8), dimension(3,3), intent(in) *ginv*)

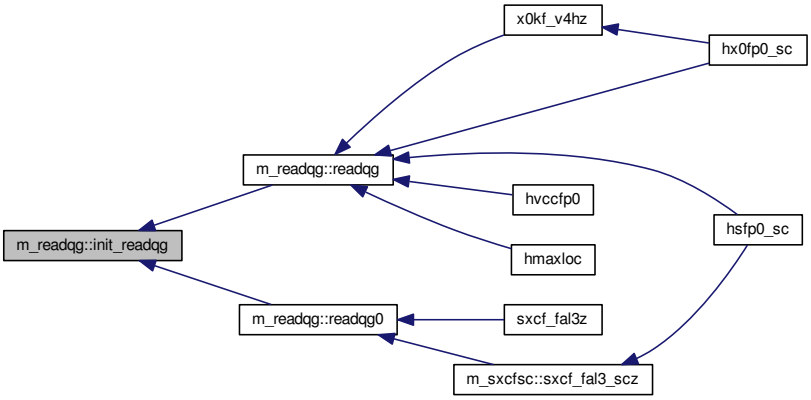
initialization. readin QGpsi or QGcou.

Definition at line 132 of file readqg.F.

Here is the call graph for this function:



Here is the caller graph for this function:

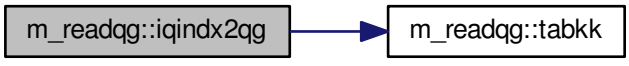


3.5.2.2 subroutine m_readqg::iqindx2qg (real(8), dimension(3), intent(in) *q,* integer, intent(in) *ifi,* integer, intent(out) *iqindx,* real(8), dimension(3), intent(out) *qu*)

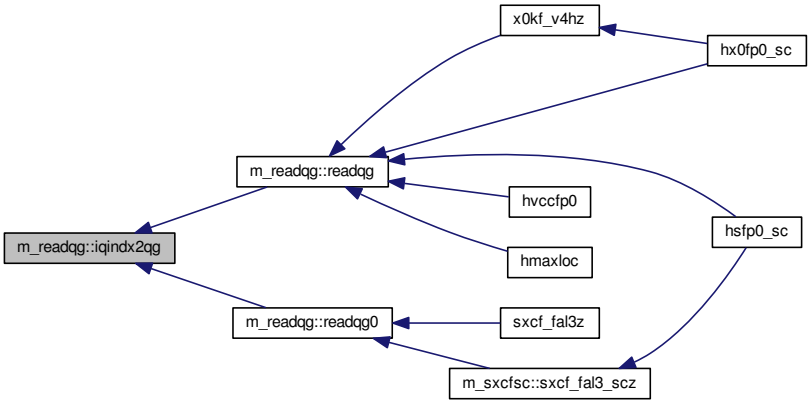
Find index as q=qq(:,iq) with modulo of premitive vector. ginv is the inverse of plat (premitive translation vector). Use kk1,kk2,kk3,nkey(1:3),iqkkk to get iqindx.

Definition at line 296 of file readqg.F.

Here is the call graph for this function:



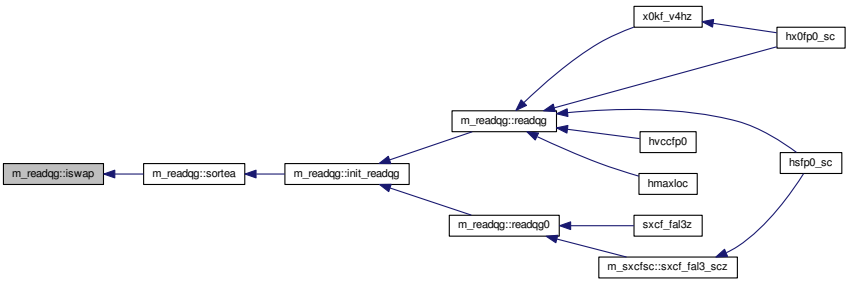
Here is the caller graph for this function:



3.5.2.3 subroutine m_readqg::iswap (integer, intent(inout) *i,* integer, intent(inout) *j*)

Definition at line 361 of file readqg.F.

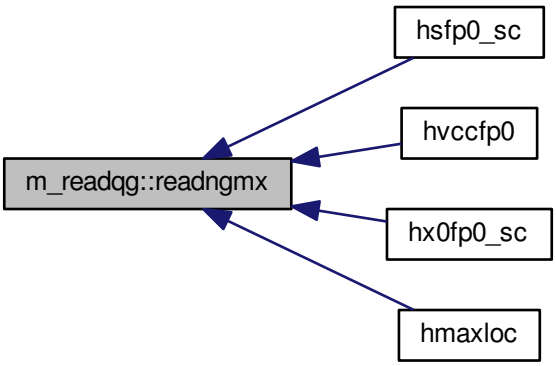
Here is the caller graph for this function:



3.5.2.4 subroutine m_readqg::readngmx (character∗(∗) *key,* integer(4) *ngmx*)

Definition at line 40 of file readqg.F.

Here is the caller graph for this function:

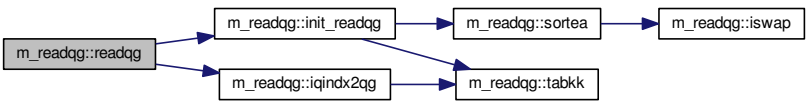


3.5.2.5 subroutine m_readqg::readqg (character∗(∗), intent(in) *key,* real(8), dimension(3), intent(in) *qin,* real(8), dimension(3,3), intent(in) *ginv,* real(8), dimension(3), intent(out) *qu,* integer(4), intent(out) *ngv,* integer(4), dimension(3,∗), intent(out) *ngvec*)

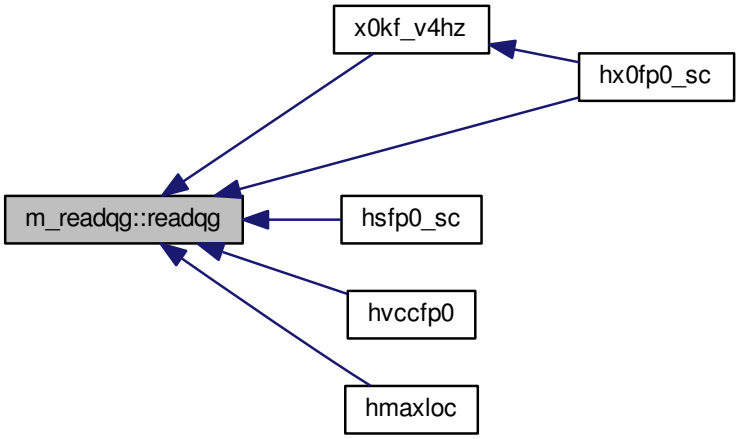
Get ngv and ngvec(3,ngv) for given qin(3) key==’QGcou’ or ’QGpsi’.

Definition at line 61 of file readqg.F.

Here is the call graph for this function:



Here is the caller graph for this function:

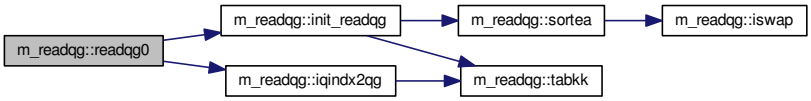


3.5.2.6 subroutine m_readqg::readqg0 (character∗(∗), intent(in) *key,* real(8), dimension(3), intent(in) *qin,* real(8), dimension(3,3), intent(in) *ginv,* real(8), dimension(3), intent(out) *qu,* integer(4), intent(out) *ngv*)

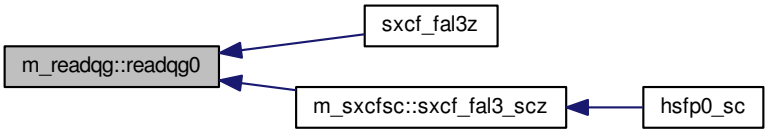
Get ngv key==’QGcou’ or ’QGpsi’.

Definition at line 98 of file readqg.F.

Here is the call graph for this function:



Here is the caller graph for this function:



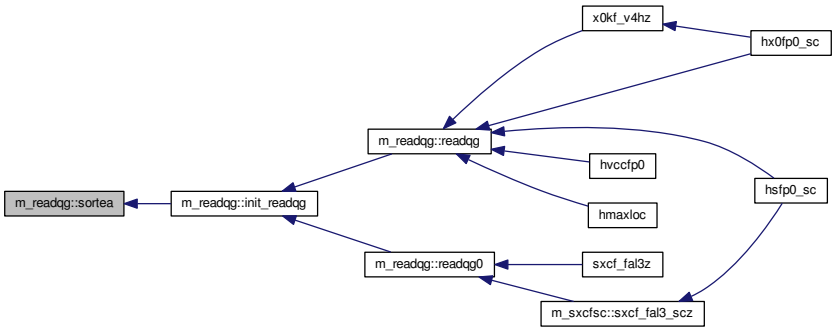
3.5.2.7 subroutine m_readqg::sortea (real(8), dimension(n), intent(in) *ea,* integer(4), dimension(n), intent(inout) *ieaord,* integer, intent(in) *n,* integer, intent(out) *isig*)

mini-sort routine.

Definition at line 340 of file readqg.F.

Here is the call graph for this function:

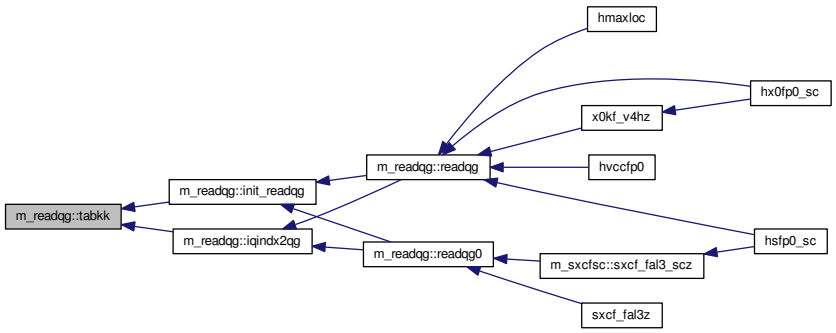
Here is the caller graph for this function:



3.5.2.8 subroutine m_readqg::tabkk (integer *kkin,* integer, dimension(n) *kktable,* integer *n,* integer *nout*)

Definition at line 248 of file readqg.F.

Here is the caller graph for this function:



3.5.3 Member Data Documentation

3.5.3.1 real(8), private m_readqg::epsd =1d-7 [private]

Definition at line 32 of file readqg.F.

3.5.3.2 real(8), dimension(3,3), private m_readqg::ginv_ [private]

Definition at line 37 of file readqg.F.

3.5.3.3 logical, dimension(2), private m_readqg::init =.true. [private]

Definition at line 26 of file readqg.F.

3.5.3.4 integer, dimension(:,:,:), pointer, private m_readqg::iqkkk [private]

Definition at line 33 of file readqg.F.

3.5.3.5 integer, dimension(:,:,:), allocatable, target, private m_readqg::iqkkkc [private]

Definition at line 36 of file readqg.F.

3.5.3.6 integer, dimension(:,:,:), allocatable, target, private m_readqg::iqkkkp [private]

Definition at line 35 of file readqg.F.

3.5.3.7 integer, dimension(:,:), allocatable, target, private m_readqg::keyc [private]

Definition at line 36 of file readqg.F.

3.5.3.8 integer, dimension(:,:), allocatable, target, private m_readqg::keyp [private]

Definition at line 35 of file readqg.F.

3.5.3.9 integer, dimension(:), pointer, private m_readqg::kk1 [private]

Definition at line 33 of file readqg.F.

3.5.3.10 integer, dimension(:), allocatable, target, private m_readqg::kk1c [private]

Definition at line 36 of file readqg.F.

3.5.3.11 integer, dimension(:), allocatable, target, private m_readqg::kk1p [private]

Definition at line 35 of file readqg.F.

3.5.3.12 integer, dimension(:), pointer, private m_readqg::kk2 [private]

Definition at line 33 of file readqg.F.

3.5.3.13 integer, dimension(:), allocatable, target, private m_readqg::kk2c [private]

Definition at line 36 of file readqg.F.

3.5.3.14 integer, dimension(:), allocatable, target, private m_readqg::kk2p [private]

Definition at line 35 of file readqg.F.

3.5.3.15 integer, dimension(:), pointer, private m_readqg::kk3 [private]

Definition at line 33 of file readqg.F.

3.5.3.16 integer, dimension(:), allocatable, target, private m_readqg::kk3c [private]

Definition at line 36 of file readqg.F.

3.5.3.17 integer, dimension(:), allocatable, target, private m_readqg::kk3p [private]

Definition at line 35 of file readqg.F.

3.5.3.18 integer(4), dimension(:), allocatable, private m_readqg::ngc [private]

Definition at line 29 of file readqg.F.

3.5.3.19 integer(4), target, private m_readqg::ngcmx [private]

Definition at line 28 of file readqg.F.

3.5.3.20 integer(4), dimension(:), allocatable, private m_readqg::ngp [private]

Definition at line 29 of file readqg.F.

3.5.3.21 integer(4), target, private m_readqg::ngpmx [private]

Definition at line 28 of file readqg.F.

3.5.3.22 integer(4), dimension(:,:,:), allocatable, private m_readqg::ngvecc [private]

Definition at line 29 of file readqg.F.

3.5.3.23 integer(4), dimension(:,:,:), allocatable, private m_readqg::ngvecp [private]

Definition at line 29 of file readqg.F.

3.5.3.24 integer, dimension(:), pointer, private m_readqg::nkey [private]

Definition at line 33 of file readqg.F.

3.5.3.25 integer, dimension(3), target, private m_readqg::nkeyc [private]

Definition at line 34 of file readqg.F.

3.5.3.26 integer, dimension(3), target, private m_readqg::nkeyp [private]

Definition at line 34 of file readqg.F.

3.5.3.27 integer(4), target, private m_readqg::nqnumc [private]

Definition at line 28 of file readqg.F.

3.5.3.28 integer(4), target, private m_readqg::nqnump [private]

Definition at line 28 of file readqg.F.

3.5.3.29 integer, pointer, private m_readqg::nqtt [private]

Definition at line 30 of file readqg.F.

3.5.3.30 real(8), dimension(:,:), allocatable, target, private m_readqg::qc [private]

Definition at line 25 of file readqg.F.

3.5.3.31 real(8), dimension(:,:), allocatable, target, private m_readqg::qp [private]

Definition at line 25 of file readqg.F.

3.5.3.32 real(8), private m_readqg::qpgcut_cou [private]

Definition at line 27 of file readqg.F.

3.5.3.33 real(8), private m_readqg::qpgcut_psi [private]

Definition at line 27 of file readqg.F.

3.5.3.34 real(8), dimension(:,:), pointer, private m_readqg::qtt [private]

Definition at line 31 of file readqg.F.

The documentation for this module was generated from the following file:

- gwsrc/readqg.F

3.6 m_sxcfsc Module Reference

this module is only because name=name argument binding. No data

Public Member Functions

- subroutine sxcf_fal3_scz (kount, qip, itq, ntq, ef, esmr, nsp, isp, qbas, ginv, qibz, qbz, wk, nstbz, irkip, nrkip, freq_r, nw_i, nw, freqx, wx, dw, ecore, nlmto, nqibz, nqbz, nctot, nbloch, ngrp, niw, nq, nblochpmx, ngpmx, ngcmx, wgt0, nq0i, q0i, symgg, alat, nband, ifvcfpout, exchange, screen, cohtest, ifexsp, nbmx, ebmx, wklm, lxklm, eftrue, jobsw, hermitianW, zsec, coh, nbandmx)
- subroutine weightset4intreal (nctot, esmr, omega, ekc, freq_r, nw_i, nw, ntqxx, nt0m, nt0p, ef, nwx, nwxi, nt_max, wfaccut, wtt, we_, wfac_, ixss, ititpskip, iirx)

3.6.1 Detailed Description

this module is only because name=name argument binding. No data

Definition at line 2 of file sxcf_fal2.sc.F.

3.6.2 Member Function/Subroutine Documentation

3.6.2.1 subroutine m_sxcfsc::sxcf_fal3_scz (integer, dimension(nqibz,nq), intent(in) *kount,* real(8), dimension(3,nq), intent(in) *qip,* integer, dimension(ntq), intent(in) *itq,* integer, intent(in) *ntq,* real(8), intent(in) *ef,* real(8), intent(in) *esmr,* integer, intent(in) *nsp,* integer, intent(in) *isp,* real(8), dimension(3,3), intent(in) *qbas,* real(8), dimension(3,3), intent(in) *ginv,* real(8), dimension(3,nqibz), intent(in) *qibz,* real(8), dimension(3,nqbz), intent(in) *qbz,* real(8), dimension(nqbz), intent(in) *wk,* integer, dimension(nqbz), intent(in) *nstbz,* integer, dimension(nqibz,ngrp,nq), intent(in) *irkip,* integer, dimension(nqibz,ngrp,nq), intent(in) *nrkip,* real(8), dimension(nw_i:nw), intent(in) *freq_r,* integer *nw_i,* integer *nw,* real(8), dimension(niw), intent(in) *freqx,* real(8), dimension(niw), intent(in) *wx,* real(8), intent(in) *dw,* real(8), dimension(nctot), intent(in) *ecore,* integer, intent(in) *nlmto,* integer, intent(in) *nqibz,* integer, intent(in) *nqbz,* integer, intent(in) *nctot,* integer, intent(in) *nbloch,* integer, intent(in) *ngrp,* integer, intent(in) *niw,* integer, intent(in) *nq,* integer, intent(in) *nblochpmx,* integer, intent(in) *ngpmx,* integer, intent(in) *ngcmx,* real(8), dimension(nq0i,ngrp), intent(in) *wgt0,* integer, intent(in) *nq0i,* real(8), dimension(1:3,1:nq0i), intent(in) *q0i,* real(8), dimension(3,3,ngrp), intent(in) *symgg,* real(8), intent(in) *alat,* integer, intent(in) *nband,* integer, intent(in) *ifvcfpout,* logical, intent(in) *exchange,* logical, intent(in) *screen,* logical, intent(in) *cohtest,* integer, intent(in) *ifexsp,* integer, dimension(2), intent(in) *nbmx,* real(8), dimension(2), intent(in) *ebmx,* real(8), dimension((lxklm+1)∗∗2), intent(in) *wklm,* integer, intent(in) *lxklm,* real(8), intent(in) *eftrue,* integer, intent(in) *jobsw,* logical *hermitianW,* complex(8), dimension(ntq,ntq,nq), intent(out), optional *zsec,* complex(8), dimension(ntq,nq), intent(out), optional *coh,* integer, dimension(nq), intent(in) *nbandmx*)

Calcualte full simga_ij(e_i)= <i|Re[Sigma](e_i)|j>

Parameters

exchange	• T : Calculate the exchange self-energy • F : Calculate correlated part of the self-energy
zsec	• S_ij= <i|Re[S](e_i)|j> • Note that S_ij itself is not Hermite becasue it includes e_i. i and j are band indexes
coh	dummy
screen	dummy

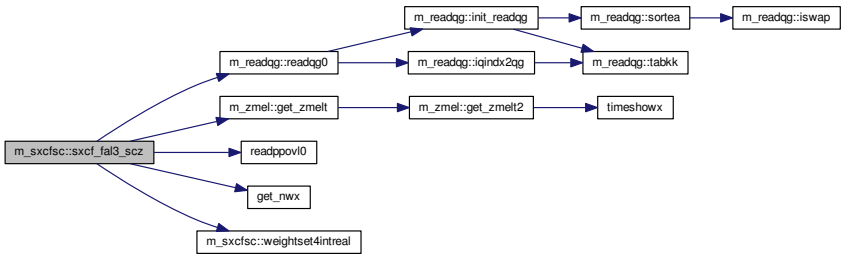
Remarks

```
Jan2013: eftrue is added.
  ef=eftrue(true fermi energy) for valence exchange and correlation mode.
  but ef is not the true fermi energy for core-exchange mode.

Jan2006
```

```
    "zsec from im-axis integral part"  had been symmetrized as
    &        wtt*.5d0*(   sum(zwzi(:,itp,itpp))+ !S_{ij}(e_i)
    &        dconjg( sum(zwzi(:,itpp,itp)) )   ) !S_{ji}^*(e_j)= S_{ij}(e_j)
    However, I now do it just the 1st term.
    &        wtt* sum(zwzi(:,itp,itpp))   !S_{ij}(e_i)
    This is OK because the symmetrization is in hqpe.sc.F
    Now zsec given in this routine is simply written as <i|Re[S](e_i)|j>.
    ( In the version until Jan2006 (fpgw032f8), only the im-axis part was symmetrized.
    But it was not necessary from the begining because it was done in hqpe.sc.F

    (Be careful as for the difference between
    <i|Re[S](e_i)|j> and transpose(dconjg(<i|Re[S](e_i)|j>)).
    &mdash;because e_i is included.
    The symmetrization (hermitian) procedure is inlucded in hqpe.sc.F

    NOTE: matrix element is given by "call get_zmelt". It returns  zmelt or zmeltt.

jobsw switch
 1-5 scGW mode.
  diag+@EF      jobsw==1 SE_nn'(ef)+delta_nn'(SE_nn(e_n)-SE_nn(ef))
  xxx modeB (Not Available now)  jobsw==2 SE_nn'((e_n+e_n')/2)  !we need to recover comment out for jobs
  mode A        jobsw==3 (SE_nn'(e_n)+SE_nn'(e_n'))/2 (Usually usued in QSGW).
  @Ef           jobsw==4 SE_nn'(ef)
  diagonly      jobsw==5 delta_nn' SE_nn(e_n) (not efficient memoryuse; but we don't use this mode so of

Output file in hsfp0 should contain hermitean part of SE
   ( hermitean of SE_nn'(e_n) means SE_n'n(e_n')^* )
            we use that zwz(itp,itpp)=dconjg( zwz(itpp,itp) )
Caution! npm=2 is not examined enough...

Calculate the exchange part and the correlated part of self-energy.
T.Kotani started development after the analysis of F.Aryasetiawan's LMTO-ASA-GW.
We still use some of his ideas in this code.

See paper
[1]T. Kotani and M. van Schilfgaarde, ??Quasiparticle self-consistent GW method:
    A basis for the independent-particle approximation, Phys. Rev. B, vol. 76, no. 16, p. 165106[24pages
[2]T. Kotani, Quasiparticle Self-Consistent GW Method Based on the Augmented Plane-Wave
   and Muffin-Tin Orbital Method, J. Phys. Soc. Jpn., vol. 83, no. 9, p. 094711 [11 Pages], Sep. 2014.

=========================================================================
Omega integral for SEc
  The integral path is deformed along the imaginary-axis, but together with contribution of poles.
  See Fig.1 and around in Ref.[1].

&mdash;Integration along imaginary axis.&mdash;
  ( Current version for it, wintzsg_npm, do not assume time-reversal when npm=2.)
  Integration along the imaginary axis: --------------&mdash;
   (Here is a memo by F.Aryasetiawan.)
    (i/2pi) < [w'=-inf,inf] Wc(k,w')(i,j)/(w'+w-e(q-k,n) >
   Gaussian integral along the imaginary axis.
   transform: x = 1/(1+w')
    this leads to a denser mesh in w' around 0 for equal mesh x
   which is desirable since Wc and the lorentzian are peaked around w'=0
    wint = - (1/pi) < [x=0,1] Wc(iw') (w-e)x^2/{(w-e)^2 + w'^2} >

    the integrand is peaked around w'=0 or x=1 when w=e
    to handel the problem, add and substract the singular part as follows:
    wint = - (1/pi) < [x=0,1] { Wc(iw') - Wc(0)exp(-a^2 w'^2) }
    * (w-e)/{(w-e)^2 +w'^2}x^2 >
    - (1/2) Wc(0) sgn(w-e) exp(a^2 (w-e)^2) erfc(a|w-e|)

    the second term of the integral can be done analytically, which
    results in the last term a is some constant

    when w = e, (1/pi) (w-e)/{(w-e)^2 + w'^2} ==> delta(w') and
    the integral becomes -Wc(0)/2
    this together with the contribution from the pole of G (s.u.)
    gives the so called static screened exchange -Wc(0)

&mdash;Integration along real axis (contribution from the poles of G: SEc(pole))
   See Eq.(34),(55), and (58) and around in Ref.[1]. We now use Gaussian Smearing.
=========================================================================
```

Definition at line 4 of file sxcf_fal2.sc.F.

Here is the call graph for this function:

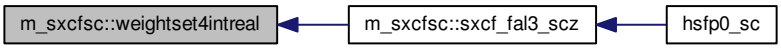


Here is the caller graph for this function:

3.6.2.2 **subroutine m_sxcfsc::weightset4intreal (integer, intent(in) *nctot,* real(8), intent(in) *esmr,* real(8), dimension(ntqxx), intent(in) *omega,* real(8), dimension(ntqxx), intent(in) *ekc,* real(8), dimension(nw_i:nw), intent(in) *freq_r,* integer, intent(in) *nw_i,* integer, intent(in) *nw,* integer, intent(in) *ntqxx,* integer, intent(in) *nt0m,* integer *nt0p,* real(8), intent(in) *ef,* integer, intent(in) *nwx,* integer, intent(in) *nwxi,* integer, intent(in) *nt_max,* real(8), intent(in) *wfaccut,* real(8), intent(in) *wtt,* real(8), dimension(nt_max,ntqxx), intent(out) *we_,* real(8), dimension(nt_max,ntqxx), intent(out) *wfac_,* integer, dimension(nt_max,ntqxx), intent(out) *ixss,* logical, dimension(nt_max,ntqxx), intent(out) *ititpskip,* integer, dimension(ntqxx), intent(out) *iirx*)**

Definition at line 1239 of file sxcf_fal2.sc.F.

Here is the caller graph for this function:



The documentation for this module was generated from the following file:

- gwsrc/sxcf_fal2.sc.F

3.7 m_tetwt Module Reference

Get the weights and index for tetrahedron method for the Lindhard function.

Public Member Functions

- subroutine gettetwt (q, iq, is, isf, nwgt)

Public Attributes

- real(8), dimension(:), allocatable whw
- integer, dimension(:,:,:), allocatable ihw
- integer, dimension(:,:,:), allocatable nhw
- integer, dimension(:,:,:), allocatable jhw
- integer, dimension(:,:,:,:), allocatable ibjb
- integer nbnbx
- integer nhwtot
- integer, dimension(:,:,:), allocatable n1b
- integer, dimension(:,:,:), allocatable n2b
- integer, dimension(:,:), allocatable nbnb

3.7.1 Detailed Description

Get the weights and index for tetrahedron method for the Lindhard function.

- nbnb = total number of weight.
- n1b = band index for occ. 1 n1b nband+nctot. "Valence index->core index" ordering(Core index follows valence index).
- n2b = band index for unocc. 1 n2b nband
- wwk(ibib,...) = (complex)weight for the pair for n1b(ibib...),n2b(ibib...).

NOTE: 'call getbzdata1' generates nteti,ntetf,... See mkqg.F about how to call it.

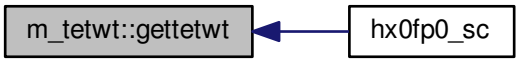
Definition at line 10 of file m_tetwt.F.

3.7.2 Member Function/Subroutine Documentation

3.7.2.1 subroutine m_tetwt::gettetwt (real(8), dimension(3), intent(in) *q,* integer, intent(in) *iq,* integer, intent(in) *is,* integer, intent(in) *isf,* integer, dimension(*), intent(in) *nwgt*)

Definition at line 18 of file m_tetwt.F.

Here is the caller graph for this function:



3.7.3 Member Data Documentation

3.7.3.1 integer, dimension(:,:,:,:), allocatable m_tetwt::ibjb

Definition at line 12 of file m_tetwt.F.

3.7.3.2 integer, dimension(:,:,:), allocatable m_tetwt::ihw

Definition at line 12 of file m_tetwt.F.

3.7.3.3 integer, dimension(:,:,:), allocatable m_tetwt::jhw

Definition at line 12 of file m_tetwt.F.

3.7.3.4 integer, dimension(:,:,:), allocatable m_tetwt::n1b

Definition at line 14 of file m_tetwt.F.

3.7.3.5 integer, dimension(:,:,:), allocatable m_tetwt::n2b

Definition at line 14 of file m_tetwt.F.

3.7.3.6 integer, dimension(:,:), allocatable m_tetwt::nbnb

Definition at line 14 of file m_tetwt.F.

3.7.3.7 integer m_tetwt::nbnbx

Definition at line 13 of file m_tetwt.F.

3.7.3.8 integer, dimension(:,:,:), allocatable m_tetwt::nhw

Definition at line 12 of file m_tetwt.F.

3.7.3.9 integer m_tetwt::nhwtot

Definition at line 13 of file m_tetwt.F.

3.7.3.10 real(8), dimension(:), allocatable m_tetwt::whw

Definition at line 11 of file m_tetwt.F.

The documentation for this module was generated from the following file:

- gwsrc/m_tetwt.F

3.8 m_zmel Module Reference

Get the matrix element zmel = ZO^-1 <MPB psi|psi> , where ZO is ppovlz. To use this module, set data in this module, and call "call get_zmelt" or "call get_zmelt2". Then we have matrix elements zmel (exchange=F for correlation) or zmeltt (exchange=T). In future, they may be unified...

Public Member Functions

- subroutine get_zmelt (exchange, q, kx, kvec, irot, rkvec, kr, isp, ngc, ngb, nmmax, nqmax, nctot, ncc)
- subroutine get_zmelt2 (exchange,

Public Attributes

- integer, parameter null =-99999
- integer, dimension(:,:), allocatable miat
- real(8), dimension(:,:,:), allocatable tiat
- real(8), dimension(:,:), allocatable shtvg
- integer nband =NULL
- integer ngcmx =NULL
- integer ngpmx =NULL
- integer ntq =NULL
- integer, dimension(:), allocatable itq
- real(8), dimension(:,:,:), allocatable ppbir
- complex(8), dimension(:,:), allocatable, target ppovlz
- complex(8), dimension(:,:,:), allocatable zmel
- complex(8), dimension(:,:,:), allocatable zmeltt

Private Attributes

- real(8), dimension(3, 3), private qbasinv
- real(8), dimension(3), private q_bk =1d10
- real(8), dimension(3), private qk_bk =1d0
- logical, private init =.true.
- complex(8), dimension(:,:), allocatable, private cphiq
- complex(8), dimension(:,:), allocatable, private cphim

- real(8), dimension(:,:,:), allocatable, private rmelt
- real(8), dimension(:,:,:), allocatable, private cmelt
- integer, private kxold =-9999

3.8.1 Detailed Description

Get the matrix element zmel = ZO^-1 <MPB psi|psi> , where ZO is ppovlz. To use this module, set data in this module, and call "call get_zmelt" or "call get_zmelt2". Then we have matrix elements zmel (exchange=F for correlation) or zmeltt (exchange=T). In future, they may be unified...

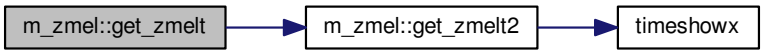
Definition at line 5 of file m_zmel.F.

3.8.2 Member Function/Subroutine Documentation

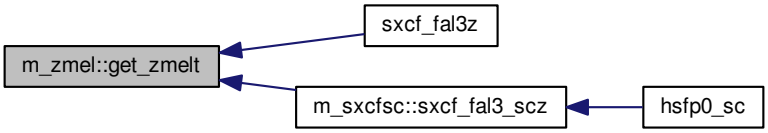
3.8.2.1 subroutine m_zmel::get_zmelt (logical *exchange,* real(8), dimension(3) *q,* integer *kx,* real(8), dimension(3) *kvec,* integer *irot,* real(8), dimension(3) *rkvec,* integer *kr,* integer *isp,* integer *ngc,* integer *ngb,* integer *nmmax,* integer *nqmax,* integer *nctot,* integer *ncc*)

Definition at line 60 of file m_zmel.F.

Here is the call graph for this function:



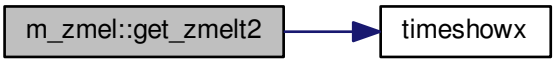
Here is the caller graph for this function:



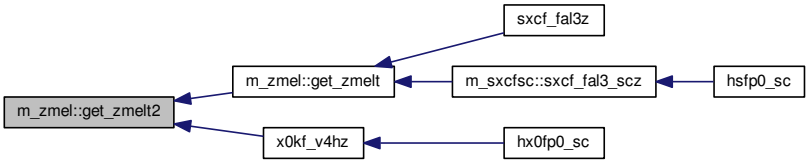
3.8.2.2 subroutine m_zmel::get_zmelt2 (logical *exchange*)

Definition at line 113 of file m_zmel.F.

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.3 Member Data Documentation

3.8.3.1 real(8), dimension(:,:,:), allocatable, private m_zmel::cmelt [private]

Definition at line 55 of file m_zmel.F.

3.8.3.2 complex(8), dimension(:,:), allocatable, private m_zmel::cphim [private]

Definition at line 54 of file m_zmel.F.

3.8.3.3 complex(8), dimension(:,:), allocatable, private m_zmel::cphiq [private]

Definition at line 54 of file m_zmel.F.

3.8.3.4 logical, private m_zmel::init =.true. [private]

Definition at line 53 of file m_zmel.F.

3.8.3.5 integer, dimension(:), allocatable m_zmel::itq

Definition at line 43 of file m_zmel.F.

3.8.3.6 integer, private m_zmel::kxold =-9999 [private]

Definition at line 56 of file m_zmel.F.

3.8.3.7 integer, dimension(:,:), allocatable m_zmel::miat

Definition at line 39 of file m_zmel.F.

3.8.3.8 integer m_zmel::nband =NULL

Definition at line 42 of file m_zmel.F.

3.8.3.9 integer m_zmel::ngcmx =NULL

Definition at line 42 of file m_zmel.F.

3.8.3.10 integer m_zmel::ngpmx =NULL

Definition at line 42 of file m_zmel.F.

3.8.3.11 integer m_zmel::ntq =NULL

Definition at line 42 of file m_zmel.F.

3.8.3.12 integer, parameter m_zmel::null =-99999

Definition at line 37 of file m_zmel.F.

3.8.3.13 real(8), dimension(:,:,:), allocatable m_zmel::ppbir

Definition at line 44 of file m_zmel.F.

3.8.3.14 complex(8), dimension(:,:), allocatable, target m_zmel::ppovlz

Definition at line 45 of file m_zmel.F.

3.8.3.15 real(8), dimension(3), private m_zmel::q_bk =1d10 [private]

Definition at line 52 of file m_zmel.F.

3.8.3.16 real(8), dimension(3,3), private m_zmel::qbasinv [private]

Definition at line 52 of file m_zmel.F.

3.8.3.17 real(8), dimension(3), private m_zmel::qk_bk =1d0 [private]

Definition at line 52 of file m_zmel.F.

3.8.3.18 real(8), dimension(:,:,:), allocatable, private m_zmel::rmelt [private]

Definition at line 55 of file m_zmel.F.

3.8.3.19 real(8), dimension(:,:), allocatable m_zmel::shtvg

Definition at line 40 of file m_zmel.F.

3.8.3.20 real(8), dimension(:,:,:), allocatable m_zmel::tiat

Definition at line 40 of file m_zmel.F.

3.8.3.21 complex(8), dimension(:,:,:), allocatable m_zmel::zmel

Definition at line 49 of file m_zmel.F.

3.8.3.22 complex(8), dimension(:,:,:), allocatable m_zmel::zmeltt

Definition at line 49 of file m_zmel.F.

The documentation for this module was generated from the following file:

- gwsrc/m_zmel.F

Chapter 4

File Documentation

4.1 exec/gwsc File Reference

Variables

- if [$#-ne 4][$2!="-np"]
- then echo An example of usage
- then echo An example of where means iterations exit fi n

4.1.1 Variable Documentation

4.1.1.1 if[$#-ne 4][$2!="-np"]

Definition at line 7 of file gwsc.

4.1.1.2 then echo An example of where means iterations exit fi n

Initial value:

```
=$0
nfpgw=`dirname $0`
TARGET=$4
MPI_SIZE=$3
NO_MPI=0
ITER=$1
lx0_para_option=""  #set lx0_para_option='-nq 4 -ns 1'


source $nfpgw/run_arg #define echo_run and serial_run in run_arg

$echo_run echo "### START gwsc: ITER= "$ITER
```

Definition at line 11 of file gwsc.

4.1.1.3 then echo An example of usage

Definition at line 8 of file gwsc.

4.2 gwsc

```
00001 #!/bin/bash
```

```
00002 # ------------------------------------------------------
00003 # QP self-consistent GW itteration using MPI. Using run_arg
00004 ### you may need to set echo_run and serial_run in /run_arg for cray machine
00005 # ------------------------------------------------------
00006
00007 if [ $# -ne 4 ] || [ $2 != "-np" ] ; then
00008     echo "An example of usage: gwsc 5 -np 4 si, where 5 means 5+1 iterations"
00009     exit 101
00010 fi
00011 n=$0
00012 nfpgw=`dirname $0`
00013 TARGET=$4
00014 MPI_SIZE=$3
00015 NO_MPI=0
00016 ITER=$1
00017 lx0_para_option=""  #set lx0_para_option='-nq 4 -ns 1'
00018
00019 ### Read funcitons run_arg and run_arg_tee defined in a file run_arg ###
00020 source $nfpgw/run_arg #define echo_run and serial_run in run_arg
00021
00022 $echo_run echo "### START gwsc: ITER= "$ITER, "MPI size= " $MPI_SIZE, "TARGET= "$TARGET
00023
00024 # ##### rm and mkdir ##############
00025 # if [ -e NoCore ]; then #backword compatibility not so meaningful now.
00026 #   rm -f NoCore
00027 # fi
00028 # if [ -e QPU ]; then
00029 #  rm -f QP[UD]
00030 # fi
00031 if [ ! -e SEBK ]; then
00032     mkdir SEBK
00033 fi
00034 if [ ! -e STDOUT ]; then
00035     mkdir STDOUT
00036 fi
00037 ## mv sigm or simg.$TARGET to sigm. And make softlink to simg.$TARGET.
00038 ## sigm is prior to simg.$TARGET.
00039 if [ -e sigm ]; then
00040     if [ -e sigm.$TARGET ]; then
00041         mv sigm.$TARGET sigm.$TARGET.bakup
00042         ln -s -f sigm sigm.$TARGET
00043         $echo_run echo '--- sigm is used. sigm.$TARGET is softlink to it  ---'
00044     fi
00045 else
00046     if [ -e sigm.$TARGET ]; then
00047         mv sigm.$TARGET sigm
00048         ln -s -f sigm sigm.$TARGET
00049         $echo_run echo '--- sigm.$TARGET is moved to sigm. sigm.$TARGET is softlink now.  ---'
00050     else
00051         $echo_run echo '--- No sigm nor sigm.$TARGET files for starting ---'
00052     fi
00053 fi
00054
00055 ##### itteration loop start ##########################################
00056 for ix in `seq 0 ${ITER}`
00057 do
00058     ##### self-consistent calculation for given Sigma(self-energy ###
00059     $echo_run echo " ---- goto sc calculation for given sigma-vxc --- ix=",$ix
00060     if [ $ix == 0 ]; then # ix=0 is for iteration from LDA.
00061         if [ -e sigm.$TARGET ] ; then
00062             $echo_run echo " we have sigm already, skip iter=0"
00063             continue # goto ix=1
00064         fi
00065         $echo_run echo "No sigm ---> LDA caculation for eigenfunctions "
00066         rm -f llmf
00067         run_arg '---' $MPI_SIZE $nfpgw          /lmf-MPIK llmf_lda $TARGET
00068         cp rst.$TARGET rst.$TARGET.lda
00069     else
00070         run_arg '---' $MPI_SIZE $nfpgw          /lmf-MPIK llmf $TARGET
00071     fi
00072     rm -f ewindow.${TARGET}* qbyl.${TARGET}* eigze*.${TARGET}* mixm.${TARGET}
00073     argin=0; run_arg $argin $NO_MPI  $nfpgw  /lmfgw  llmfgw00 $TARGET
00074     argin=1; run_arg $argin $NO_MPI  $nfpgw  /qg4gw lqg4gw   #Generate requied q+G v
00075
00076     ### eigenvalues for micro-tetrahedron method. Rarely used.
00077     if [ -e Qmtet ]; then
00078         mv Qmtet Qeigval
00079         argin=5; run_arg $argin $MPI_SIZE $nfpgw /lmfgw-MPIK llmfgw_eigval $TARGET
00080         mv eigval eigmtet
00081     fi
00082     argin=1; run_arg $argin $MPI_SIZE $nfpgw /lmfgw-MPIK llmfgw01 $TARGET
00083     run_arg  '---' $NO_MPI   $nfpgw /lmf2gw     llmf2gw  #reform data for gw
00084
00085     mv normchk* STDOUT
00086
00087     ##### main stage of gw #################
00088     argin=0; run_arg $argin $NO_MPI   $nfpgw /rdata4gw_v2 lrdata4gw_v2 #prepare files
```

```
00089     rm gw1* gw2* gwa* gwb*
00090     if [ $ix == 0 ]; then
00091         cp evec.$TARGET evec0  # used in hqpe_sc for isigma_en==5
00092     fi
00093     if [ -e ANFcond ]; then
00094         cp EVU EVD  # This is for ANFcond. Rarely used recently
00095     fi
00096     argin=1; run_arg $argin $NO_MPI $nfpgw   /heftet leftet # A file EFERMI for hx0fp0
00097     argin=1; run_arg $argin $NO_MPI $nfpgw   /hchknw lchknw # A file NW, containing nw
00098     ### Core part of the self-energy (exchange only) ###
00099     argin=3; run_arg $argin $NO_MPI $nfpgw   /hbasfp0 lbasC  # Product basis generation
00100     argin=3; run_arg $argin $MPI_SIZE $nfpgw /hvccfp0 lvccC # Coulomb matrix for lbasC
00101     argin=3; run_arg $argin $MPI_SIZE $nfpgw /hsfp0_sc lsxC # Sigma from core1
00102     mv stdout* STDOUT
00103     ### Valence part of the self-energy Sigma ###
00104     argin=0; run_arg $argin $NO_MPI $nfpgw   /hbasfp0 lbas # Product basis generation
00105     argin=0; run_arg $argin $MPI_SIZE $nfpgw /hvccfp0 lvcc # Coulomb matrix for lbas
00106     argin=1; run_arg $argin $MPI_SIZE $nfpgw /hsfp0_sc lsx # Exchange Sigma
00107     mv stdout* STDOUT
00108     if [ -e WV.d ]; then
00109         rm -f WV*
00110     fi
00111     # following two runs are most expensive #
00112     argin=11; run_arg $argin $MPI_SIZE $nfpgw /hx0fp0_sc lx0 $lx0_para_option #x0 part
00113     mv stdout* STDOUT
00114     argin=2;  run_arg $argin $MPI_SIZE $nfpgw /hsfp0_sc lsc #correlation Sigma
00115     mv stdout* STDOUT
00116     argin=0;  run_arg $argin $NO_MPI $nfpgw   /hqpe_sc lqpe #all Sigma are combined.
00117     $echo_run echo -n 'OK! --> '
00118     ### final part of iteration loop. Manupulate files ###
00119     cp evec.$TARGET evec_prev_iter
00120     ln -s -f sigm sigm.$TARGET
00121     mv SEX* SEC* XC* SEBK
00122     for file in sigm QPU QPD TOTE.UP TOTE.DN lqpe lsc lsx lx0 llmfgw01 evecfix.chk llmf ESEAVR
00123     do
00124         if [ -e $file ]; then
00125             cp $file $file.${ix}run
00126         fi
00127     done
00128     if [ $ix == 0 ] && [ ${ITER} != 0 ]; then
00129         mkdir RUN0
00130         run_arg '---' $MPI_SIZE $nfpgw           /lmf-MPIK llmf_oneshot $TARGET
00131         cp ctrl.$TARGET rst.$TARGET sigm.$TARGET llmf_oneshot save.$TARGET RUN0
00132     fi
00133
00134     mkdir RUN.ITER${ix}
00135     cp ctrl.$TARGET rst.$TARGET sigm.$TARGET GWinput save.$TARGET RUN.ITER${ix}
00136
00137     $echo_run echo == $ix 'iteration over =='
00138 done
00139 ################## end of loop ######################
00140
00141 ### finally we have llmf_gwscend ###
00142 run_arg '---' $MPI_SIZE $nfpgw             /lmf-MPIK llmf_gwscend.${ITER} $TARGET
00143 rm -f ewindow.${TARGET}* qbyl.${TARGET}* eigze*.${TARGET}* _IN_
00144 if [ ${ITER} == 0 ]; then
00145     mkdir RUN0
00146     cp ctrl.$TARGET rst.$TARGET sigm.$TARGET llmf_gwscend.${ITER} save.$TARGET RUN0
00147 fi
00148 $echo_run echo OK! ==== All calclation finished for  gwsc $argv ====
00149 exit
```

4.3 exec/makefile File Reference

Variables

- PLATFORM
- doxygen
- cd latex
- make echo fpgw latex refman
  pdf generated init

4.3.1 Variable Documentation

4.3.1.1 doxygen

Definition at line 63 of file makefile.

4.3.1.2 make echo fpgw latex refman pdf generated init

Definition at line 657 of file makefile.

4.3.1.3 cd latex

Definition at line 63 of file makefile.

4.3.1.4 PLATFORM

Definition at line 9 of file makefile.

4.4 makefile

```
00001 ### I think that you don't needs to modify this file. ###
00002 ### This file is not machine-dependent. #####
00003 ### Machine dependence in make.inc
00004
00005
00006 # ---- Machine-specific compiler flags ---
00007 #include make.inc.ifort_asahi_kino
00008 #include make.inc.thinkpad_gfortran_tkotani
00009 PLATFORM=gfortran
00010 LIBMATH=/usr/lib/x86_64-linux-gnu/libfftw3.so.3 /usr/lib/liblapack.so.3gf /usr/lib/libblas.so.3gf
00011
00012 #PLATFORM=ifort
00013 #LIBMATH=-mkl
00014
00015
00016 include make.inc.$(PLATFORM)
00017
00018 BINDIR = $(HOME)/bin
00019
00020 #-----------------------------------------------------------
00021 # src directories
00022 gwsrc    = ../gwsrc/
00023 main     = ../main/
00024 nfpsrc   = ../nfpsrc/
00025 slatsmlib  = ../slatsmlib/
00026 tote = ../tote/
00027 #maxloc = ../Miyake/maxloc/
00028 # tag directory
00029 tags   = ../
00030
00031 #progs  = hbasfp0 hvccfp0 hx0fp0 hsfp0 hef hqpe hchknw qg4gw gwinit heftet hmergewv hparainfo hbndout
       rdata4gw_v2 convgwin hx0fp0_sc hsfp0_sc hqpe_sc kino_input_test hecor eout eout2 h_uumatrix hsigmconv
00032 # lmf_exec
00033 #progs  = hbasfp0 hvccfp0 hx0fp0 hsfp0 hef hqpe hchknw qg4gw gwinit heftet hmergewv hparainfo hbndout
       rdata4gw_v2  hx0fp0_fal hx0fp1
00034
00035 progs  = hbasfp0 hvccfp0 hx0fp0 hsfp0 hef hqpe hchknw qg4gw gwinit heftet hmergewv hbndout
      rdata4gw_v2 convgwin hx0fp0_sc hsfp0_sc hqpe_sc kino_input_test hecor eout eout2
00036
00037 # progs  = hbasfp0 hvccfp0 hx0fp0 hsfp0 hef hqpe hchknw qg4gw gwinit heftet hmergewv hbndout rdata4gw_v2
       convgwin hx0fp0_sc hsfp0_sc hqpe_sc kino_input_test hecor eout eout2 h_uumatrix hsigmconv hwmat hmaxloc huumat
       qpwf hpsig hnocc_mlw hx0fp0_mlw hphig
00038
00039 # hmaxloc1D
00040 progs2 = $(progs) $(tags)TAGS
00041 #checkmod
00042
00043 #script = cleargw* dqpu dtote eps* ex* gw* hqpemetal* inf* lmgw* plotg save* tote_lmfh2 xqp mkG*
00044 script = cleargw* dqpu eps* gw* mkG*
00045
00046 #### You can choose these options. all is default.
00047
00048 all :$(progs2)
00049
```

```
00050 clean:
00051         rm -f  $(progs)
00052
00053 install:
00054         cp  $(progs)  $(BINDIR)
00055
00056 install2:
00057         cp  $(script) $(BINDIR)
00058
00059 cleanall:
00060         rm -f  $(progs2) $(main)*.o $(gwsrc)*.o  $(nfpsrc)*.o  *.mod  $(slatsmlib)*.o $(tote)*.o  $(maxloc
      )*.o
00061
00062 doxygen:
00063         cd $(tags);doxygen;cd ./latex;make
00064         echo 'fpgw/latex/refman.pdf generated'
00065
00066 # This is necesaly to compile *.f in right order.
00067 # When you recompile and link, just repeat 'make' (not necessary to repeat 'make init').
00068 # When checkmodule recompile source, you have to repeat 'make'.
00069 init:
00070         rm -f $(main)time_hsfp0.sc.m.F
00071         rm -f $(main)time_hx0fp0.sc.m.F
00072         rm -f $(gwsrc)time_sxcf_fal2.sc.F
00073         rm -f $(gwsrc)time_rppovl.F
00074         rm -f $(gwsrc)time_x0kf_v4h.F
00075         rm -f $(gwsrc)time_ppbafp.fal.F
00076         exec ../../TOOLS/checkmodule ../gwsrc/*.F ../main/*.F ../tote/*.F
00077
00078 checkmod:
00079         init
00080 #../../lm7K/subs/m_hamindex.F
00081
00082
00083 ####from tote ###############################################################################

00084 #LIBLOC  = $(ECAL)/fftw/libfftw.a $(LIBMATH)
00085 ##-L/usr/local/ATLAS/lib/Linux_P4SSE2 -llapack -lcblas -lf77blas -latlas

00086 #LIBSLA  = $(ECAL)/slatsm/slatsm.a
00087 #LIBFP   = $(ECAL)/lm-6.14y/fp/subs.a
00088 #LIBSUBS = $(ECAL)/lm-6.14y/subs/subs.a
00089 #LIBES  = $(LIBSLA) $(LIBLOC)
00090 #lmsrc   = ../../lm-6.14y/
00091 ############################################################################################
00092
00093 COMM=$(nfpsrc)rxx.o \
00094 $(gwsrc)mopen.o
00095
00096 ECOR = \
00097 $(tote)hecor.o \
00098 $(tote)rpaq.o
00099 #eispack.o
00100
00101 NFPLtot  = $(nfpsrc)diagcv2.o
00102
00103 EO= \
00104 $(tote)eout.o \
00105 $(gwsrc)rydberg.o
00106
00107 EO2= \
00108 $(tote)eout2.o \
00109 $(gwsrc)rydberg.o
00110
00111 hecor: $(ECOR) $(NFPLtot) $(GW0)   $(MPI)  $(COMM)
00112         $(LK) $(LKFLAGS1) $(ECOR) $(GW0) $(NFPLtot)  $(MPI)  $(COMM) $(LKFLAGS2) -o $@
00113
00114 eout: $(EO)   $(COMM)
00115         $(LK) $(LKFLAGS1) $(EO) $(COMM) $(LKFLAGS2) -o $@
00116
00117 eout2: $(EO2)  $(COMM)
00118         $(LK) $(LKFLAGS1) $(EO2) $(COMM) $(LKFLAGS2) -o $@
00119
00120 #################
00121 #
00122 # BNDCONN= \
00123 # $(gwsrc)bndconn.o   ### This is not linked but bndconn.o is used in lm/lmfgw.
00124 # It is now included in lm/gw/
00125  DERFC=   $(nfpsrc)derfc.o \
00126           $(nfpsrc)d1mach.o \
00127           $(nfpsrc)i1mach.o
00128
00129 # test_genallcf =  \
00130 # $(main)test_genallcf.o \
00131 # $(gwsrc)genallcf_dump.o \
00132 # $(GW0)
00133
```

```
00134
00135  kino_input_test =  \
00136  $(main)kino_input_test.o
00137
00138  convg =  \
00139  $(main)convgwin.o
00140
00141  GWINIT =  \
00142  $(main)gwinit.m.o \
00143  $(gwsrc)cross.o \
00144  $(gwsrc)genqbz.o \
00145  $(gwsrc)checksymlon.o \
00146  $(gwsrc)bzmesh.o \
00147  $(gwsrc)rangedq.o \
00148  $(gwsrc)iopenxx.o \
00149  $(gwsrc)iprint.o \
00150  $(gwsrc)keyvalue.o \
00151  $(gwsrc)switches.o \
00152  $(gwsrc)iopen.o
00153
00154
00155  QG =  \
00156  $(gwsrc)conv2gwinput.o \
00157  $(main)qg4gw.m.o \
00158  $(gwsrc)getbzdata1.o \
00159  $(gwsrc)mkqg.o \
00160  $(gwsrc)q0irre.o \
00161  $(gwsrc)getgv2.o \
00162  $(gwsrc)tetwt5$(tet5_g) \
00163  $(gwsrc)zsvd.o \
00164  $(GW0)
00165
00166  RDAT_v2 = \
00167  $(gwsrc)keyvalue.o \
00168  $(gwsrc)switches.o \
00169  $(main)rdata4gw_v2.m.o \
00170  $(gwsrc)rwbzdata.o \
00171  $(gwsrc)gintxx.o \
00172  $(gwsrc)cinvrx.o \
00173  $(gwsrc)idxk.o \
00174  $(gwsrc)nword.o \
00175  $(gwsrc)gwinput_v2.o \
00176  $(gwsrc)matm.o \
00177  $(gwsrc)getgv2.o \
00178  $(gwsrc)iopen.o \
00179  $(gwsrc)pplmat.o \
00180  $(gwsrc)bzmesh.o \
00181  $(gwsrc)ext.o \
00182  $(gwsrc)ext2.o \
00183  $(gwsrc)cross.o \
00184  $(gwsrc)rs.o \
00185  $(gwsrc)extension.o \
00186  $(gwsrc)rangedq.o \
00187  $(gwsrc)llnew.o \
00188  $(gwsrc)iqindx.o \
00189  $(gwsrc)polinta.o\
00190  $(gwsrc)m_anf.o
00191
00192
00193  BAS = \
00194  $(main)hbasfp0.m.o \
00195  $(gwsrc)basnfp.o \
00196  $(gwsrc)gintxx.o \
00197  $(gwsrc)rs.o \
00198  $(gwsrc)excore.o
00199
00200
00201  VCC= \
00202  $(main)hvccfp0.m.o \
00203  $(gwsrc)mkjp.o \
00204  $(gwsrc)gintxx.o \
00205  $(gwsrc)extension.o \
00206  $(gwsrc)rangedq.o \
00207  $(gwsrc)keyvalue.o \
00208  $(gwsrc)switches.o \
00209  $(gwsrc)strxq.o \
00210  $(gwsrc)iopen.o \
00211  $(gwsrc)pplmat.o \
00212  $(gwsrc)matm.o \
00213  $(gwsrc)getgv2.o \
00214  $(gwsrc)cross.o \
00215  $(gwsrc)llnew.o \
00216  $(gwsrc)readqg.o \
00217  $(gwsrc)iqindx.o \
00218  $(gwsrc)cputid.o
00219
00220  SXC_SC = \
```

```
00221  $(main)hsfp0.sc.m.o \
00222  $(gwsrc)wse.o \
00223  $(gwsrc)sxcf_fal2$(sxcf_g) \
00224  $(gwsrc)sxcf_fal2.sc$(sxcf_g) \
00225  $(gwsrc)bzints2.o \
00226  $(gwsrc)wintzsg.o \
00227  $(nfpsrc)diagcv2.o \
00228  $(gwsrc)m_zmel.o
00229
00230  SXC = \
00231  $(main)hsfp0.m.o \
00232  $(gwsrc)wse.o \
00233  $(gwsrc)wintzsg.o \
00234  $(gwsrc)sxcf_fal2$(sxcf_g) \
00235  $(gwsrc)bzints2.o \
00236  $(gwsrc)genallcf_dump.o \
00237  $(nfpsrc)diagcv2.o \
00238  $(gwsrc)m_zmel.o
00239
00240  WMAT = \
00241  $(maxloc)hwmat.o \
00242  $(maxloc)maxloc0.o \
00243  $(gwsrc)wse.o \
00244  $(maxloc)wmat.o \
00245  $(gwsrc)genallcf_dump.o
00246
00247  MLOC = \
00248  $(maxloc)hmaxloc.o \
00249  $(maxloc)maxloc0.o \
00250  $(maxloc)maxloc1.o \
00251  $(maxloc)maxloc2.o \
00252  $(maxloc)maxloc3.o \
00253  $(gwsrc)wse.o \
00254  $(gwsrc)genallcf_dump.o
00255
00256  MLOC1D = \
00257  $(maxloc)hmaxloc1D.o \
00258  $(maxloc)maxloc0.o \
00259  $(maxloc)maxloc1.o \
00260  $(maxloc)maxloc2.o \
00261  $(maxloc)maxloc3.o \
00262  $(gwsrc)wse.o \
00263  $(gwsrc)genallcf_dump.o
00264
00265  heftet = \
00266  $(main)heftet.m.o \
00267  $(gwsrc)bzints2.o
00268
00269  hnocc_mlw = \
00270  $(maxloc)hnocc_mlw.o \
00271  $(gwsrc)bzints2.o
00272
00273  hef = \
00274  $(main)hef.m.o \
00275  $(gwsrc)wse.o
00276
00277  CHK = \
00278  $(main)hchknw.m.o \
00279  $(gwsrc)genallcf_dump.o
00280
00281  X0_SC = \
00282  $(main)hx0fp0.sc.m.o \
00283  $(gwsrc)wcf.o \
00284  $(gwsrc)tetwt5$(tet5_g) \
00285  $(gwsrc)m_tetwt.o \
00286  $(gwsrc)x0kf_v4h$(x0kf_g) \
00287  $(nfpsrc)diagcv2.o \
00288  $(gwsrc)m_zmel.o \
00289  $(gwsrc)m_freq.o
00290
00291  X0 = \
00292  $(main)hx0fp0.m.o \
00293  $(gwsrc)wcf.o \
00294  $(gwsrc)tetwt5$(tet5_g) \
00295  $(gwsrc)m_tetwt.o \
00296  $(gwsrc)x0kf_v4h$(x0kf_g) \
00297  $(nfpsrc)diagcv2.o \
00298  $(tote)rpaq.o \
00299  $(gwsrc)cinvrx.o \
00300  $(gwsrc)zsvd.o \
00301  $(gwsrc)m_zmel.o \
00302  $(gwsrc)m_freq.o
00303
00304  X0mlw = \
00305  $(maxloc)hx0fp0.m.o \
00306  $(maxloc)wcf.o \
00307  $(gwsrc)tetwt5$(tet5_g) \
```

```
00308  $(gwsrc)m_tetwt.o \
00309  $(nfpsrc)diagcv2.o \
00310  $(tote)rpaq.o \
00311  $(gwsrc)cinvrx.o\
00312  $(gwsrc)m_freq.o
00313
00314  # UU = \
00315  # $(main)h_uumatrix.m.o \
00316  # $(gwsrc)wcf.o \
00317  # $(gwsrc)tetwt5$(tet5_g) \
00318  # $(gwsrc)gintxx.o \
00319  # $(gwsrc)pplmat.o \
00320  # $(gwsrc)getgv2.o \
00321  # $(gwsrc)x0kf_v4h$(x0kf_g) \
00322  # $(gwsrc)rs.o \
00323  # $(nfpsrc)u_lat_0.o \
00324  # $(nfpsrc)wronkj.o \
00325  # $(nfpsrc)mklegw.o \
00326  # $(nfpsrc)bessl.o \
00327  # $(nfpsrc)cross.o \
00328  # $(nfpsrc)diagcv2.o
00329
00330  # UU2 = \
00331  # $(maxloc)huumat.o \
00332  # $(gwsrc)wcf.o \
00333  # $(gwsrc)tetwt5$(tet5_g) \
00334  # $(gwsrc)gintxx.o \
00335  # $(gwsrc)pplmat.o \
00336  # $(gwsrc)getgv2.o \
00337  # $(gwsrc)rs.o \
00338  # $(nfpsrc)u_lat_0.o \
00339  # $(nfpsrc)wronkj.o \
00340  # $(nfpsrc)mklegw.o \
00341  # $(nfpsrc)bessl.o \
00342  # $(nfpsrc)cross.o
00343
00344  PSIG = \
00345  $(maxloc)hpsig.o \
00346  $(gwsrc)wcf.o \
00347  $(gwsrc)tetwt5$(tet5_g) \
00348  $(gwsrc)m_tetwt.o \
00349  $(gwsrc)gintxx.o \
00350  $(gwsrc)pplmat.o \
00351  $(gwsrc)getgv2.o \
00352  $(gwsrc)rs.o \
00353  $(nfpsrc)u_lat_0.o \
00354  $(nfpsrc)wronkj.o \
00355  $(nfpsrc)mklegw.o \
00356  $(nfpsrc)bessl.o \
00357  $(nfpsrc)cross.o
00358
00359  PHIG = \
00360  $(maxloc)hphig.o \
00361  $(gwsrc)wcf.o \
00362  $(gwsrc)tetwt5$(tet5_g) \
00363  $(gwsrc)m_tetwt.o \
00364  $(gwsrc)gintxx.o \
00365  $(gwsrc)pplmat.o \
00366  $(gwsrc)getgv2.o \
00367  $(gwsrc)rs.o \
00368  $(nfpsrc)u_lat_0.o \
00369  $(nfpsrc)wronkj.o \
00370  $(nfpsrc)mklegw.o \
00371  $(nfpsrc)bessl.o \
00372  $(nfpsrc)cross.o
00373
00374  MPI = $(gwsrc)MPI_fpgw2.o
00375
00376  GW0 = \
00377  $(gwsrc)m_hamindex.o\
00378  $(gwsrc)readpomat.o \
00379  $(gwsrc)keyvalue.o \
00380  $(gwsrc)rppovl.o \
00381  $(gwsrc)nocctotg.o \
00382  $(gwsrc)ppbafp.fal$(para_g) \
00383  $(gwsrc)psi2b_v2$(para_g) \
00384  $(gwsrc)psi2b_v3$(para_g) \
00385  $(gwsrc)wfacx.o \
00386  $(gwsrc)sortea.o \
00387  $(gwsrc)rydberg.o \
00388  $(gwsrc)polinta.o \
00389  $(gwsrc)efsimplef.o \
00390  $(gwsrc)extension.o \
00391  $(gwsrc)rangedq.o \
00392  $(gwsrc)nword.o \
00393  $(gwsrc)scg.o \
00394  $(gwsrc)matm.o \
```

```
00395  $(gwsrc)rdpp.o \
00396  $(gwsrc)mptauof.o \
00397  $(gwsrc)genallcf_mod.o \
00398  $(gwsrc)rgwinf_mod.o \
00399  $(gwsrc)rotdlmm.o \
00400  $(gwsrc)iopen.o \
00401  $(gwsrc)cputid.o \
00402  $(gwsrc)rw.o \
00403  $(gwsrc)ext.o \
00404  $(gwsrc)ext2.o \
00405  $(gwsrc)cross.o \
00406  $(gwsrc)mate.o \
00407  $(gwsrc)mate1.o \
00408  $(gwsrc)icopy.o \
00409  $(gwsrc)bib1.o \
00410  $(gwsrc)index.o \
00411  $(gwsrc)idxk.o \
00412  $(gwsrc)maxnn.o \
00413  $(gwsrc)reindx.o \
00414  $(gwsrc)iprint.o \
00415  $(gwsrc)bz.o \
00416  $(gwsrc)bzmesh.o \
00417  $(gwsrc)genqbz.o \
00418  $(gwsrc)switches.o \
00419  $(gwsrc)rwbzdata.o \
00420  $(gwsrc)llnew.o  \
00421  $(gwsrc)readeigen.o \
00422  $(gwsrc)readqg.o \
00423  $(gwsrc)iqindx.o  \
00424  $(gwsrc)alloclist.o \
00425  $(gwsrc)m_pkm4crpa.o \
00426  $(gwsrc)m_anf.o
00427
00428 # $(gwsrc)linpackdummy.o \
00429
00430  QPE_SC = \
00431  $(main)hqpe.sc.m$(hqpe_g) \
00432  $(gwsrc)qpe1.sc.o \
00433  $(gwsrc)icompvv2.o \
00434  $(gwsrc)iopenxx.o \
00435  $(slatsmlib)dsifa.o \
00436  $(slatsmlib)dsisl.o \
00437  $(slatsmlib)dsidi.o \
00438  $(slatsmlib)amix.o
00439 # ../../slatsm/slatsm.a
00440
00441
00442
00443  QPE = \
00444  $(gwsrc)switches.o \
00445  $(gwsrc)keyvalue.o \
00446  $(main)hqpe.m$(hqpe_g) \
00447  $(gwsrc)qpe1.o \
00448  $(gwsrc)icompvv2.o \
00449  $(gwsrc)iopenxx.o \
00450  $(gwsrc)iopen.o \
00451  $(gwsrc)rw.o \
00452  $(gwsrc)rydberg.o
00453
00454  MERGE = \
00455  $(main)hmergewv.m.o \
00456  $(gwsrc)switches.o \
00457  $(gwsrc)keyvalue.o \
00458  $(gwsrc)iopen.o
00459
00460  PARAINFO = \
00461  $(main)hparainfo.m.o \
00462  $(gwsrc)charext.o
00463
00464
00465  BNDOUT = \
00466  $(main)hbndout.m.o \
00467  $(gwsrc)iqagree.o \
00468  $(gwsrc)iopenxx.o \
00469  $(gwsrc)iopen.o \
00470  $(gwsrc)polinta.o \
00471  $(gwsrc)rydberg.o \
00472  $(gwsrc)extension.o \
00473  $(gwsrc)rangedq.o \
00474  $(gwsrc)switches.o \
00475  $(gwsrc)keyvalue.o
00476
00477
00478  NFPL  = $(nfpsrc)wronkj.o \
00479          $(nfpsrc)sylm.o \
00480          $(nfpsrc)sylmnc.o \
00481          $(nfpsrc)u_lat_0.o \
```

```
00482          $(nfpsrc)mklegw.o \
00483          $(nfpsrc)cross.o \
00484          $(nfpsrc)setpr.o \
00485          $(nfpsrc)bessl.o \
00486          $(nfpsrc)hsmq.o \
00487          $(nfpsrc)lgen.o \
00488          $(nfpsrc)hansr5.o \
00489          $(nfpsrc)hansr4.o \
00490          $(nfpsrc)lattc.o \
00491          $(nfpsrc)ll.o \
00492          $(nfpsrc)dpcopy.o \
00493          $(nfpsrc)dpadd.o \
00494          $(nfpsrc)syscalls.o \
00495          $(nfpsrc)qdist.o \
00496          $(nfpsrc)dlmtor.o \
00497          $(nfpsrc)dpzero.o \
00498          $(nfpsrc)ropyln.o \
00499          $(nfpsrc)ropcsm.o \
00500          $(nfpsrc)dsisl.o \
00501          $(nfpsrc)dsifa.o \
00502          $(nfpsrc)diagcv2.o \
00503          $(gwsrc)scg.o
00504
00505  SIGMCONV = \
00506  $(gwsrc)switches.o \
00507  $(gwsrc)keyvalue.o \
00508  $(gwsrc)iopen.o \
00509  $(main)hsigmconv.m.o
00510
00511 ##########################################
00512
00513 # bndconn.o:    $(BNDCONN)
00514 #
00515 ############### dependency for use ##################
00516
00517
00518
00519  hsigmconv:     $(SIGMCONV)  $(MPI)  $(COMM)
00520          $(LK) $(LKFLAGS1) $(SIGMCONV) $(MPI)  $(COMM) $(LKFLAGS2) -o $@
00521
00522
00523  gwinit:        $(GWINIT)  $(MPI)  $(COMM)
00524          $(LK) $(LKFLAGS1) $(GWINIT) $(MPI) $(COMM)  $(LKFLAGS2) -o $@
00525
00526
00527  qpwf:          $(maxloc)qpwf.o $(GW0) $(MPI)  $(COMM)
00528         $(LK) $(LKFLAGS1) $(maxloc)qpwf.o $(GW0) $(MPI) $(COMM)  $(LKFLAGS2) -o $@
00529
00530  qg4gw:         $(QG)  $(MPI) $(COMM)
00531          $(LK) $(LKFLAGS1) $(QG) $(MPI) $(COMM)    $(LKFLAGS2) -o $@
00532
00533  rdata4gw_v2:   $(RDAT_v2)  $(NFPL) $(MPI)  $(COMM)
00534          $(LK) $(LKFLAGS1) $(RDAT_v2) $(NFPL) $(MPI) $(COMM)  $(LKFLAGS2) -o $@
00535
00536  hbasfp0:       $(BAS)  $(MPI)  $(COMM) $(GW0)
00537          $(LK) $(LKFLAGS1) $(BAS) $(MPI)  $(COMM)  $(GW0) $(LKFLAGS2) -o $@
00538
00539  hvccfp0:       $(MPI) $(VCC)  $(NFPL) $(DERFC) $(MPI)  $(COMM)
00540          $(LK) $(LKFLAGS1) $(VCC) $(NFPL) $(DERFC) $(MPI)  $(COMM) $(LKFLAGS2) -o $@
00541
00542  hx0fp0:        $(MPI) $(X0) $(GW0) $(MPI)  $(COMM)
00543          $(LK) $(LKFLAGS1) $(X0)     $(GW0) $(MPI)  $(COMM) $(LKFLAGS2) -o $@
00544
00545 # # for maxloc
00546 #  hx0fp0_mlw:  $(X0mlw) $(GW0) $(MPI)  $(COMM)
00547 #       $(LK) $(LKFLAGS1) $(X0mlw)     $(GW0)  $(MPI)  $(COMM)  $(LKFLAGS2) -o $@
00548
00549 #  h_uumatrix:  $(UU) $(GW0)   $(MPI)  $(COMM)
00550 #       $(LK) $(LKFLAGS1) $(UU)     $(GW0)  $(MPI)  $(COMM)  $(LKFLAGS2) -o $@
00551
00552 #  huumat:      $(UU2) $(GW0) $(MPI) $(COMM)
00553 #       $(LK) $(LKFLAGS1) $(UU2)  $(GW0)  $(MPI)  $(COMM)  $(LKFLAGS2) -o $@
00554
00555 #  hphig:       $(PHIG) $(GW0) $(MPI)  $(COMM)
00556 #       $(LK) $(LKFLAGS1) $(PHIG)     $(GW0)  $(MPI)  $(COMM)  $(LKFLAGS2) $(LIBSLA) -o $@
00557
00558 #  hpsig: $(PSIG) $(GW0) $(MPI)  $(COMM)
00559 #       $(LK) $(LKFLAGS1) $(PSIG)     $(GW0)  $(MPI)  $(COMM)  $(LKFLAGS2) -o $@
00560
00561  hx0fp0_sc:     $(MPI) $(X0_SC) $(GW0)   $(MPI)  $(COMM)
00562          $(LK) $(LKFLAGS1) $(X0_SC)     $(GW0) $(MPI)  $(COMM)  $(LKFLAGS2) -o $@
00563
00564 #  hwmat:               $(WMAT) $(GW0) $(MPI)  $(COMM)
00565 #       $(LK) $(LKFLAGS1) $(WMAT)    $(GW0)  $(MPI)  $(COMM)  $(LKFLAGS2) -o $@
00566
00567 #  hmaxloc:     $(MLOC)  $(NFPLtot) $(GW0) $(MPI)  $(COMM)
00568 #       $(LK) $(LKFLAGS1) $(MLOC) $(NFPLtot) $(GW0)  $(MPI)  $(COMM)  $(LKFLAGS2) -o $@
```

```
00569
00570 #  hmaxloc1D:   $(MLOC1D)  $(NFPLtot) $(GW0) $(MPI)  $(COMM)
00571 #       $(LK) $(LKFLAGS1) $(MLOC1D) $(NFPLtot) $(GW0)   $(MPI)  $(COMM) $(LKFLAGS2) -o $@
00572
00573  hsfp0:         $(MPI) $(SXC) $(GW0)   $(MPI)  $(COMM)
00574          $(LK) $(LKFLAGS1) $(SXC)    $(GW0) $(MPI)  $(COMM)  $(LKFLAGS2) -o $@
00575
00576  hsfp0_sc:      $(MPI) $(SXC_SC) $(GW0)   $(MPI)  $(COMM)
00577          $(LK) $(LKFLAGS1) $(SXC_SC)    $(GW0)  $(MPI)  $(COMM) $(LKFLAGS2) -o $@
00578
00579 #  hnocc_mlw:   $(hnocc_mlw) $(GW0) $(MPI)  $(COMM)
00580 #       $(LK) $(LKFLAGS1) $(hnocc_mlw) $(GW0)  $(MPI)  $(COMM)  $(LKFLAGS2) -o $@
00581
00582  heftet:        $(heftet) $(GW0)  $(MPI) $(MPI)  $(COMM)
00583          $(LK) $(LKFLAGS1) $(heftet) $(GW0)  $(MPI)  $(COMM) $(LKFLAGS2) -o $@
00584
00585  hef:           $(hef) $(GW0)   $(MPI) $(COMM)
00586          $(LK) $(LKFLAGS1) $(hef)    $(GW0)  $(MPI)  $(COMM) $(LKFLAGS2) -o $@
00587
00588  hchknw:        $(CHK) $(GW0)  $(MPI)
00589          $(LK) $(LKFLAGS1) $(CHK)    $(GW0)   $(MPI)  $(COMM) $(LKFLAGS2) -o $@
00590
00591  hqpe:          $(QPE) $(MPI) $(COMM)
00592          $(LK) $(LKFLAGS1) $(QPE) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00593
00594  hqpe_sc:               $(QPE_SC)  $(MPI) $(COMM) $(GW0)
00595          $(LK) $(LKFLAGS1) $(QPE_SC) $(GW0) $(MPI)  $(COMM) $(LKFLAGS2) -o $@
00596
00597  hmergewv:      $(MERGE) $(MPI)   $(COMM)
00598          $(LK) $(LKFLAGS1) $(MERGE) $(MPI)  $(COMM)  $(LKFLAGS2) -o $@
00599
00600  # hparainfo:   $(PARAINFO) $(GW0) $(MPI)   $(COMM)
00601  #       $(LK) $(LKFLAGS1) $(PARAINFO) $(GW0)  $(MPI)  $(COMM) $(LKFLAGS2) -o $@
00602
00603  hbndout:       $(BNDOUT) $(MPI)   $(COMM)
00604          $(LK) $(LKFLAGS1) $(BNDOUT) $(MPI)  $(COMM)  $(LKFLAGS2) -o $@
00605
00606  convgwin:      $(convg) $(MPI)   $(COMM)
00607          $(LK) $(LKFLAGS1) $(convg) $(MPI)  $(COMM)  $(LKFLAGS2) -o $@
00608
00609  kino_input_test:       $(kino_input_test) $(GW0) $(MPI)  $(COMM)
00610          $(LK) $(LKFLAGS1) $(kino_input_test) $(GW0)  $(MPI)  $(COMM) $(LKFLAGS2) -o $@
00611
00612 ################################ test
00613 #
00614 # test_genallcf:        $(test_genallcf)
00615 #        $(LK) $(LKFLAGS1) $(test_genallcf) $(LKFLAGS2) -o $@
00616
00617
00618  $(tags)TAGS: $(progs)
00619         cd $(tags);etags ./*/*/*.F ./*/*.F
00620
00621
00622 # --- Make rules ---
00623 .SUFFIXES:
00624 .SUFFIXES: .F .o
00625 #.SUFFIXES: .f .o .c1_o .c2_0 .c3_o .c4_o .F
00626
00627 .F.o:
00628         $(FC) $(FFLAGS) $*.F -c -o $*.o
00629 #       etags $*.f -o $(tags)`echo $*.f| sed 's/..\///' | sed 's/\//-/g'`.tags
00630
00631 #.F.o:
00632 #       $(FC) $(FFLAGS) $*.F -c -o $*.o
00633 #       etags $*.f -o $(tags)`echo $*.f| sed 's/..\///' | sed 's/\//-/g'`.tags
00634
00635 #.f.o:
00636 #       $(FC) $(FFLAGS) $*.f -c -o $*.o
00637 #       etags $*.f -o $(tags)`echo $*.f| sed 's/..\///' | sed 's/\//-/g'`.tags
00638
00639 .f.c1_o:
00640         $(FC) $(FFLAGS_c1) $*.f -c -o $*.c1_o
00641         etags $*.f -o $(tags)`echo $*.f| sed 's/..\///' | sed 's/\//-/g'`.tags
00642
00643 .f.c2_o:
00644         $(FC) $(FFLAGS_c2) $*.f -c -o $*.c2_o
00645         etags $*.f -o $(tags)`echo $*.f| sed 's/..\///' | sed 's/\//-/g'`.tags
00646
00647 .f.c3_o:
00648         $(FC) $(FFLAGS_c3) $*.f -c -o $*.c3_o
00649         etags $*.f -o $(tags)`echo $*.f| sed 's/..\///' | sed 's/\//-/g'`.tags
00650
00651 .f.c4_o:
00652         $(FC) $(FFLAGS_c4) $*.f -c -o $*.c4_o
00653         etags $*.f -o $(tags)`echo $*.f| sed 's/..\///' | sed 's/\//-/g'`.tags
00654
00655
```

```
00656 check:
00657         (cd ../TESTinstallGW;./testgw.py --enforce --all)
00658
00659 # test for f90 dependency
00660 #../main/hvccfp0.m.o    :       ../main/hx0fp0.m.o
00661 #
00662 #../main/hvccfp0.m.o    :       ../main/hbasfp0.m.o
00663
00664 include moduledepends.inc
00665
00666
00667 ################################################################################
00668 ##### You can comment out these blocks to commnet out memory and time check (verbose output)
00669 addtime=script/addtime.awk
00670 septhen=script/then_separate.awk
00671 alloclist=script/add_alloclist.awk
00672 $(main)hsfp0.sc.m.o:  $(main)hsfp0.sc.m.F
00673         gawk -f $(addtime) -vSTART=1 $(main)hsfp0.sc.m.F | gawk -f $(septhen) | gawk -f $(alloclist) >
      $(main)time_hsfp0.sc.m.F
00674         $(FC) $(FFLAGS) $(main)time_hsfp0.sc.m.F -c -o $*.o
00675
00676 $(main)hx0fp0.sc.m.o: $(main)hx0fp0.sc.m.F
00677         gawk -f $(addtime) -vSTART=1 $(main)hx0fp0.sc.m.F | gawk -f $(septhen) | gawk -f $(alloclist) >
      $(main)time_hx0fp0.sc.m.F
00678         $(FC) $(FFLAGS) $(main)time_hx0fp0.sc.m.F -c -o $*.o
00679
00680 $(gwsrc)sxcf_fal2.sc$(sxcf_g): $(gwsrc)sxcf_fal2.sc.F
00681         gawk -f $(addtime) -vSTART=100 $(gwsrc)sxcf_fal2.sc.F | gawk -f $(septhen) | gawk -f $(alloclist) >
      $(gwsrc)time_sxcf_fal2.sc.F
00682         $(FC) $(FFLAGS) $(gwsrc)time_sxcf_fal2.sc.F -c -o $*.o
00683
00684 #$(gwsrc)rppovl.o: $(gwsrc)rppovl.F
00685 #       gawk -f $(addtime) -vSTART=200 $(gwsrc)rppovl.F | gawk -f $(septhen) | gawk -f $(alloclist) >
      $(gwsrc)time_rppovl.F
00686 #       $(FC) $(FFLAGS) $(gwsrc)time_rppovl.F -c -o $*.o
00687
00688 $(gwsrc)x0kf_v4h$(x0kf_g): $(gwsrc)x0kf_v4h.F
00689         gawk -f $(addtime) -vSTART=100 $(gwsrc)x0kf_v4h.F | gawk -f $(septhen) | gawk -f $(alloclist) >
      $(gwsrc)time_x0kf_v4h.F
00690         $(FC) $(FFLAGS) $(gwsrc)time_x0kf_v4h.F -c -o $*.o
00691
00692 $(gwsrc)ppbafp.fal$(para_g): $(gwsrc)ppbafp.fal.F
00693         gawk -f $(addtime) -vSTART=300 $(gwsrc)ppbafp.fal.F | gawk -f $(septhen) | gawk -f $(alloclist) >
      $(gwsrc)time_ppbafp.fal.F
00694         $(FC) $(FFLAGS) $(gwsrc)time_ppbafp.fal.F -c -o $*.o
00695 #$(gwsrc)ppbafp.fal$(para_g): $(gwsrc)ppbafp.fal.F
00696 #       gawk -f $(addtime) -vSTART=300 $(gwsrc)ppbafp.fal.F | gawk -f $(septhen) | gawk -f $(alloclist) >
      $(gwsrc)time_ppbafp.fal.F
00697 #       $(FC) $(FFLAGS) $(gwsrc)time_ppbafp.fal.F -c -o $*.o
00698 ################################################################################
00699
00700
00701 $(gwsrc)wintzsg.o :  $(gwsrc)wintzsg.F
00702         $(FC)  $(FFLAGS) $(gwsrc)wintzsg.F  -c -o $*.o
00703
00704
00705 # DO NOT DELETE
```

4.5 gwsrc/genallcf_mod.F File Reference

Data Types

- module m_genallcf_v3

  get basic settings of crystal structure and nlm info

Functions/Subroutines

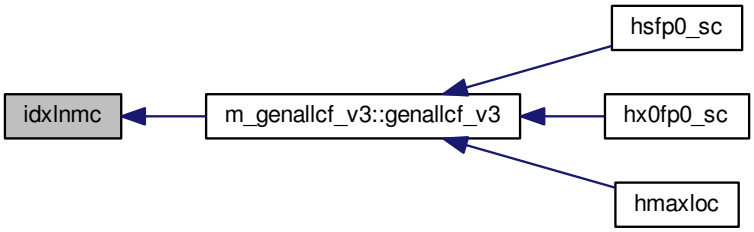
- subroutine idxlnmc (nindxv, nindxc,
- integer function noflmto (nindx, iclass, nl, nclass, natom)
- integer function nalwln (nocc, nunocc, nindx, nl, nn)
- integer function nofln (nindx, nl)
- integer function noflnm (nindx, nl)
- integer function nallow (nocc, nunocc, nindx, nl, nn)
- subroutine incor (ncwf, nindxc, iclass,

4.5.1 Function/Subroutine Documentation

4.5.1.1 subroutine idxlnmc (dimension(0:nl-1,nclass) *nindxv,* dimension(0:nl-1,nclass) *nindxc*)

Definition at line 369 of file genallcf_mod.F.

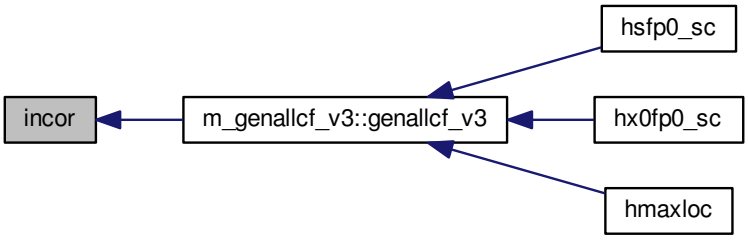
Here is the caller graph for this function:



4.5.1.2 subroutine incor (dimension(0:nl-1,nnc,nclass) *ncwf,* dimension(0:nl-1,nclass) *nindxc, iclass*)

Definition at line 545 of file genallcf_mod.F.

Here is the caller graph for this function:



4.5.1.3 integer function nallow (dimension(0:nl-1,nn) *nocc,* dimension(0:nl-1,nn) *nunocc, nindx, nl, nn*)

Definition at line 503 of file genallcf_mod.F.

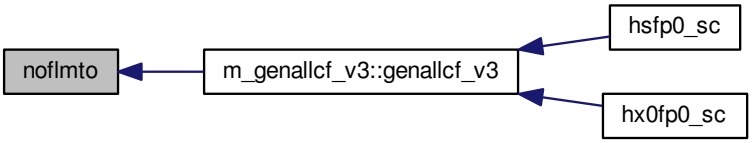
4.5.1.4 integer function nalwln (dimension(0:nl-1,nn) *nocc,* dimension(0:nl-1,nn) *nunocc, nindx, nl, nn*)

Definition at line 452 of file genallcf_mod.F.

4.5.1.5 integer function noflmto (dimension(0:nl-1,nclass) *nindx,* dimension(natom) *iclass, nl, nclass, natom*)

Definition at line 439 of file genallcf_mod.F.

Here is the caller graph for this function:



4.5.1.6 integer function nofln (dimension(0:nl-1) *nindx, nl*)

Definition at line 481 of file genallcf_mod.F.

4.5.1.7 integer function noflnm (dimension(0:nl-1) *nindx, nl*)

Definition at line 492 of file genallcf_mod.F.

4.6 genallcf_mod.F

```
00001 !> get basic settings of crystal structure and nlm info
00002 !!  - genallcf_v3(nwin,efin,incwfx) set data
00003 !!  - This is old routine. Confusing. We need to clean up.
00004       module m_genallcf_v3
00005 !!-----------------------------------------------------------------
00006 !! - structure
00007 !!  -  o                  plat,alat,natom,nclass,pos,
00008 !!  -  o                  ngrp, symgg,
00009 !!  -  o                  invg, ef,
00010 !! - l,n and dimensions
00011 !!   - o                  clabl, nspin,nl,nn,nnv,nnc,
00012 !!   - o                  nindx, nindxv, nindxc, iclass,
00013 !!   - d                  nlmto,nlnx,nlnxv,nlnxc,nlnmx,nlnmxv,nlnmxc,
00014 !!   - o                  z,
00015 !! - l,n,m indices for Phi (atomic basis)
00016 !!   -  o                   il, in, im,  ilnm,  nlnm,
00017 !!   -  o                   ilv,inv,imv, ilnmv, nlnmv,
00018 !!   -  o                   ilc,inc,imc, ilnmc, nlnmc,
00019 !! - core
00020 !!   -  o                   ncwf, ecore, konf, icore, ncore,nctot,
00021 !! - frequency
00022 !!   -                  niw,diw,nw,dw,delta,deltaw,esmr, freq)
00023 !!            symgrp
00024 !!            ,nocc, nunocc, occv, unoccv, occc, unoccc
00025       implicit none
00026       character(120):: symgrp
00027       character(6),allocatable :: clabl(:)
00028       integer,allocatable:: iclass(:),
00029      &    nindxv(:,:),nindxc(:,:),ncwf(:,:,:) ,
00030      o    invg(:), il(:,:), in(:,:), im(:,:),   ilnm(:),  nlnm(:),
00031      o    ilv(:),inv(:),imv(:),  ilnmv(:), nlnmv(:),
00032      o    ilc(:),inc(:),imc(:),  ilnmc(:), nlnmc(:),
00033      o    nindx(:,:),konf(:,:),icore(:,:),ncore(:),
00034      &    occv(:,:,:),unoccv(:,:,:)
00035      &   ,occc(:,:,:),unoccc(:,:,:),
00036      o    nocc(:,:,:),nunocc(:,:,:),   iantiferro(:)
00037       integer::
00038      o  nclass,natom,nspin,nl,nn,nnv,nnc, ngrp,
00039      o  nlmto,nlnx,nlnxv,nlnxc,nlnmx,nlnmxv,nlnmxc, nctot, niw,nw
00040       real(8), allocatable::
00041      o  plat(:,:),pos(:,:),z(:),  ecore(:,:), symgg(:,:,:)   !w(igrp) freq(:),
00042       real(8) :: alat,ef, diw,dw,delta,deltaw,esmr
00043       logical:: done_genallcf_v3=.false.
00044       character(8),allocatable:: spid(:)
00045 c-----------------------------------------------------------------------
```

```
00046       contains
00047
00048       subroutine genallcf_v3(nwin,efin,incwfx)
00049 !!> Readin GWIN_V2 and LMTO(crystal) data and allocate all required.
00050 !!r Return iclass=ibas.
00051 !! nwin,efin,incwfx, are used as switches.
00052 !! input: nwin,efin,incwfx,
00053 !!        GWIN_V2, LMTO
00054 !! output: All the output are given in the declear section above.
00055 !! -----------------------------------------------------------
00056       implicit none
00057       integer(4)::iflmto,ifinin,nwin,incwfx,ifec,i,j,
00058      & lmx, lmx2,nlmto2,nprodxc,nlnaxc,nlnaxv,nprodx,ifi,ig,is
00059      & ,iopen,iclose,nprodxv,nlnax
00060      & ,noflmto,maxnn
00061       integer(4):: infwfx
00062       integer(4):: n1,n2,n3,imagw,lcutmx,n,ic
00063       logical :: nocore
00064       real(8)::efin
00065       real(8),allocatable::tolbas(:)
00066       character(120):: symgrpt
00067       real(8),   allocatable:: ecoret(:,:,:,:)
00068       integer(4),allocatable::ncwf2(:,:,:)
00069       integer:: ia,l,m,ic1,isp,lt,nt,nsp,nr,ncorex,ifix
00070       real(8)::a,b,zz
00071 c      allocate(nclass,natom,nspin,nl,nn,nnv,nnc, ngrp,
00072 c     o  nlmto,nlnx,nlnxv,nlnxc,nlnmx,nlnmxv,nlnmxc, nctot, niw,nw)
00073       if(done_genallcf_v3) call rx('genallcf_v3 is already called')
00074       done_genallcf_v3=.true.
00075
00076 c      allocate(alat,ef, diw,dw,delta,deltaw,esmr,  symgrp)
00077       iflmto     = iopen('LMTO',1,0,0)
00078       if (iflmto < 0) call rx( 'unit file for GWIN_V2 < 0')
00079
00080 c--- readin these by rgwinf_v3
00081 c      character*120 symgrp
00082 c      integer(4)::nclass,natom,nspin,nl,nnv,nnc
00083 c      real(8)::alat
00084 c      integer(4),allocatable::
00085 c     &   iclass(:)
00086 c     &  ,nindxv(:,:),nindxc(:,:)
00087 c     &  ,occv(:,:,:),unoccv(:,:,:)
00088 c     &  ,occc(:,:,:),unoccc(:,:,:)
00089 c     &  ,ncwf(:,:,:)
00090 c      real(8),allocatable:: plat(:,:),pos(:,:),z(:)
00091 c      character*6,allocatable:: clabl(:)
00092 c      write(6,*)' goto rgwin'
00093 c      call rgwinf_v3(iflmto,ifinin,nwin,efin,incwfx) !these are inputs
00094 c      write(6,*)' end of rgwinf_v3'
00095 c-------------------------------------------------------------------------
00096 c--- rgwinf ---
00097       ifi = iflmto
00098       nw  = nwin
00099       ef  = efin
00100       read(ifi,*);  read(ifi,*)
00101       read(ifi,*)symgrpt        !SYMMETRY
00102       j = 0
00103       symgrp='  '//trim(adjustl(symgrpt))
00104       write(6,*)' symgrp=', symgrp
00105       read(ifi,*)
00106       read(ifi,*)
00107       read(ifi,*)
00108       read(ifi,*)alat              !lattice constant
00109       allocate(plat(3,3))          !primitive lattice vectors
00110       read(ifi,*)
00111       read(ifi,*)plat(1:3,1)
00112       read(ifi,*)plat(1:3,2)
00113       read(ifi,*)plat(1:3,3)
00114       read(ifi,*)
00115       read(ifi,*) natom            !Number of atoms
00116 !!
00117       nclass = natom  !We set nclass = natom through the GW calculations
00118       write(6,*)'genalloc: alat natom=',alat,natom
00119       allocate(pos(3,natom))       !positions of atoms
00120       read(ifi,*)
00121       do n = 1,natom
00122         read(ifi,*) pos(1,n),pos(2,n),pos(3,n)
00123       end do
00124       read(ifi,*)
00125       read(ifi,*)
00126       read(ifi,*)
00127       read(ifi,*)nspin           !spin (1=paramagnetic  2=ferromagnetic)
00128       read(ifi,*)
00129       read(ifi,*)nl              !max. no. valence and core l
00130       read(ifi,*)
00131       read(ifi,*)nnv,nnc   !max. no. valence and core n
00132       write(6,*)' nspin nl nnv nnc =',nspin,nl,nnv,nnc
```

```
00133 c--------------------------
00134       if(nnv==1) nnv=2 ! for backword compatibility!takao apr 2002
00135       ! nnv=2 corresponds to phi and phidot
00136       ! nnv=3 corresponds to
00137 c--------------------------
00138       read(ifi,*)
00139       read(ifi,*)  !nrx is not readin
00140       read(ifi,*)
00141       allocate(clabl(nclass),z(nclass)) !class-label, z
00142       do ic = 1,nclass
00143         read(ifi,*) clabl(ic),z(ic) !,nrofi is not readin
00144       end do
00145
00146       allocate(iclass(natom)) !atom and its class.
00147       do n = 1,natom          !!We set nclass = natom through the GW calculations
00148         iclass(n)=n
00149       end do
00150
00151       allocate(nindxv(nl,nclass), nindxc(nl,nclass),
00152      &        occv(nl,nnv,nclass),unoccv(nl,nnv,nclass),
00153      &        occc(nl,nnc,nclass),unoccc(nl,nnc,nclass))
00154       allocate(ncwf2(nl,nnc,nclass),ncwf(nl,nnc,nclass))
00155       allocate(tolbas(0:2*(nl-1)))
00156       ifix=ifi
00157       call rgwinaf(ifi,ifinin,nl,nnv,nnc,nclass, !ifi can be changed.
00158 c> bz
00159      o                  n1,n2,n3,ef,
00160 c> frequencies
00161      o                  niw,diw,nw,dw,delta,deltaw,esmr,imagw,
00162 c> coulomb
00163 c     o                   tolvc,alp,alptx,h,ng,
00164 c> product basis
00165      o                  tolbas,lcutmx,nindxv,nindxc,
00166      o                  occv,unoccv, occc,unoccc,
00167 c> core
00168      o                  ncwf,ncwf2 )
00169 c----
00170       allocate(iantiferro(1:natom),spid(1:natom))
00171       read(ifix,*)
00172       read(ifix,*)iantiferro(1:natom) !may2015
00173       read(ifix,*)
00174       read(ifix,*)spid(1:natom)
00175       inquire(file='NoCore',exist=nocore)
00176       if(nocore) then
00177         occc=0    ! call iclear(nl*nnc*nclass, w(ioccc))
00178         unoccc=0    ! call iclear(nl*nnc*nclass, w(iunoccc))
00179         ncwf  =0    ! call iclear(nl*nnc*nclass, w(incwf))
00180       elseif( incwfx==-1 ) then
00181         write(6,*)' ### incwf=-1 Use ForSxc for core'
00182         ncwf = ncwf2  !call icopy(nl*nnc*nclass,w(incwf2),w(incwf))
00183       elseif( incwfx==-2 ) then
00184         write(6,*)' ### incwf=-2 Use NOT(ForSxc) for core and Pro-basis '
00185         call notbit(nl*nnc*nclass, ncwf2)
00186         ncwf  = ncwf2 ! call icopy (nl*nnc*nclass, w(incwf2),w(incwf))
00187         occc= ncwf  ! call icopy (nl*nnc*nclass, w(incwf),w(ioccc))
00188         unoccc= 0     ! call iclear(nl*nnc*nclass, w(iunoccc))
00189       elseif( incwfx==-3 ) then
00190         call ibiton(nclass,nl,nnc,nindxc, occc, ncwf)
00191         unoccc= 0     ! call iclear(nl*nnc*nclass, w(iunoccc))
00192         write(6,*)' ### incwf=-3  occ=1 unocc=0 incwf=1 for all core '
00193       elseif( incwfx==-4 ) then
00194         write(6,*)' ### incwf=-4  occ=0 and unocc=0 for all core '
00195         occc=0  !call iclear(nl*nnc*nclass, w(ioccc))
00196         unoccc=0 !call iclear(nl*nnc*nclass, w(iunoccc))
00197         ncwf=0  !call iclear(nl*nnc*nclass, w(incwf))
00198       elseif(incwfx==0) then
00199         write(6,*)' ### Use unocc occ ForX0 for core'
00200       else
00201         call rx( ' ### proper incwf is not given for genallcf2:rgwinf ')
00202       endif
00203       deallocate(ncwf2)
00204 c... End of rgwinf section ----------------------------
00205
00206
00207 c> dimensions and constants
00208       lmx        = 2*(nl-1)
00209       lmx2       = (lmx+1)**2
00210       nlmto      = noflmto(nindxv,iclass,nl,nclass,natom)
00211       nlmto2     = nlmto*nlmto
00212       nn         = maxnn(nindxv,nindxc,nl,nclass)
00213
00214 c>> combine nocc,nunocc,nindx
00215       allocate(nindx(nl,nclass))
00216       allocate(nocc(nl,nn,nclass),nunocc(nl,nn,nclass))
00217       call reindx(occv,unoccv,nindxv,
00218      i             occc,unoccc,nindxc,
00219      d             nl,nn,nnv,nnc,nclass,
```

```
00220      o              nocc,nunocc,nindx)
00221       call maxdim(occc,unoccc,nindxc,nl,nnc,nclass,
00222      o              nprodxc,nlnxc,nlnmxc,nlnaxc)
00223       call maxdim(occv,unoccv,nindxv,nl,nnv,nclass,
00224      o              nprodxv,nlnxv,nlnmxv,nlnaxv)
00225       call maxdim(nocc,nunocc,nindx,nl,nn,nclass,
00226      o              nprodx,nlnx,nlnmx,nlnax)
00227
00228 c      nlnx4      = nlnx**4
00229 c      nphi       = nrx*nl*nn*nclass
00230 c      pi         = 4d0*datan(1d0)
00231 c      tpia       = 2d0*pi/alat
00232
00233 c$$$c> frequency mesh
00234 c$$$c      call defdr(ifreq,nw)
00235 c$$$      write(6,*)' nw from rgwinaf=',nw
00236 c$$$      if(nw>0) then
00237 c$$$        allocate(freq(nw))
00238 c$$$        call genfreq(nw,dw,0.d0,
00239 c$$$     o              freq )
00240 c$$$      endif
00241
00242 c> index for allowed core states
00243 c      call defi(iicore,nl*nl*nnc*nclass)
00244 c      call defi(incore,nclass)
00245       allocate(icore(nl**2*nnc,nclass),ncore(nclass))
00246       icore=9999999
00247       ncore=9999999
00248       call incor(ncwf,nindxc,iclass,
00249      d              nl,nnc,nclass,natom,
00250      o              icore,ncore,nctot )
00251 ccccccccccccccccccccccccccccccccccccccccccccccccc
00252 c      write(6,*)' nnc=',nnc,nl,nclass,natom
00253 c      write(6,*)' ncwf ',ncwf
00254 c      write(6,*)' nindxc ',nindxc
00255 c      write(6,*)' iclass ',iclass
00256 c      write(6,*)' --- icore=',icore
00257 c      write(6,*)' --- ncore nctot=',ncore,nctot
00258 ccccccccccccccccccccccccccccccccccccccccccccccccc
00259
00260 c> core energies
00261       ifec       = iopen('ECORE',1,0,0)
00262       allocate(konf(nl,nclass),ecore(nctot,2))
00263       konf=0
00264       allocate(ecoret(0:nl-1,nnc,2,nclass))
00265       ecoret=0d0
00266       do ic = 1,nclass
00267         write(6,*) ' read ECORE : ic=',ic
00268         read (ifec,*)
00269         read (ifec,*)
00270         read (ifec,*)
00271         read (ifec,*) !zz,ic1,nr ,a,b,nsp
00272         read (ifec,*)
00273         read (ifec,*) (konf(l+1,ic),l=0,nl-1)
00274         read (ifec,*)
00275         do  l = 0,nl-1
00276           ncorex = konf(l+1,ic)-l-1
00277           if (ncorex .gt. nnc) call rx( 'ECORE: wrong nnc')
00278           do n = 1,ncorex
00279             read (ifec,*) lt,nt,(ecoret(l,n,isp,ic),isp=1,nspin) !takao
00280             if(nspin==1) ecoret(l,n,2,ic) = ecoret(l,n,1,ic)        !
00281 c           write(6,"(' read ecore=',3i4,2d13.5)")l,n,ic,ecoret(l,n,1:nspin,ic)
00282             if (lt .ne. l) call rx( 'rcore: wrong l')
00283             if (nt .ne. n) call rx( 'rcore: wrong n')
00284           end do
00285         end do
00286       end do
00287       i = 0
00288       do ia = 1,nclass
00289         ic  = iclass(ia)
00290         do l = 0,nl-1
00291         do n = 1,nnc
00292         do m = -l,l
00293           if (ncwf(l+1,n,ic) .eq. 1) then
00294             i = i + 1
00295             if (i > nctot) call rx( 'genalloc_mod: wrong nctot')
00296             ecore(i,1:nspin) = ecoret(l,n,1:nspin,ic)
00297             write(6,"(' ecore=',4i4,2d13.5)")i, l,n,ic,ecore(i,1:nspin)
00298           endif
00299         enddo
00300         enddo
00301         enddo
00302       enddo
00303       deallocate(ecoret)
00304 c> index for core and lmto basis
00305 c      call defi(iil,nlnmx*nclass)
00306 c      call defi(iin,nlnmx*nclass)
```

```
00307 c      call defi(iim,nlnmx*nclass)
00308 c      call defi(iilnm,nn*nl*nl*nclass)
00309 c      call defi(iilv,nlnmxv*nclass)
00310 c      call defi(iinv,nlnmxv*nclass)
00311 c      call defi(iimv,nlnmxv*nclass)
00312 c      call defi(iilnmv,nnv*nl*nl*nclass)
00313 c      call defi(iilc,nlnmxc*nclass)
00314 c      call defi(iinc,nlnmxc*nclass)
00315 c      call defi(iimc,nlnmxc*nclass)
00316 c      call defi(iilnmc,nnc*nl*nl*nclass)
00317       allocate(
00318      & il(nlnmx,nclass),
00319      & in(nlnmx,nclass),
00320      & im(nlnmx,nclass),
00321      & ilnm(nn*nl*nl*nclass),
00322      & ilv(nlnmxv*nclass),
00323      & inv(nlnmxv*nclass),
00324      & imv(nlnmxv*nclass),
00325      & ilnmv(nnv*nl*nl*nclass),
00326      & ilc(nlnmxc*nclass),
00327      & inc(nlnmxc*nclass),
00328      & imc(nlnmxc*nclass),
00329      & ilnmc(nnc*nl*nl*nclass)
00330      & )
00331       call idxlnmc( nindxv,nindxc,
00332      d              nl,nn,nnv,nnc,nlnmx,nlnmxv,nlnmxc,nclass,
00333      o              il,in,im,ilnm,
00334      o              ilv,inv,imv,ilnmv,
00335      o              ilc,inc,imc,ilnmc)
00336       allocate(nlnmv(nclass),nlnmc(nclass),nlnm(nclass))
00337       call nolnma(nindxv,nl,nclass,
00338      o              nlnmv )
00339       call nolnma(nindxc,nl,nclass,
00340      o              nlnmc )
00341       call nolnma(nindx,nl,nclass,
00342      o              nlnm )
00343       i=2 !see previous definition of symgrp
00344       if(symgrp(i+1:i+13)/= 'UseSYMOPSfile') then
00345         call rx( " Not: UseSYMOPSfile in LMTO file")
00346       endif
00347       write(6,*) ' symgrp==UseSYMOPSfile'
00348       ifi = 6661
00349       open (ifi, file='SYMOPS')
00350       read(ifi,*) ngrp
00351       allocate(symgg(3,3,ngrp))
00352       do ig = 1,ngrp
00353         read(ifi,*)
00354         do i=1,3
00355           read(ifi,"(3d24.16)") symgg(i,1:3,ig)
00356         enddo
00357       enddo
00358       close(ifi)
00359       allocate(invg(ngrp))
00360       call invgrp(symgg,ngrp,
00361      o              invg)
00362       is = iclose('LMTO')
00363       is = iclose('ECORE')
00364       call cputid(0)
00365       write(6,*) 'genallcf_v3'
00366       end subroutine genallcf_v3
00367       end module
00368
00369       subroutine idxlnmc(nindxv,nindxc,
00370      d                   nl,nn,nnv,nnc,nlnmx,nlnmxv,nlnmxc,nclass,
00371      o                   il,in,im,ilnm,
00372      o                   ilv,inv,imv,ilnmv,
00373      o                   ilc,inc,imc,ilnmc)
00374 c 92.jan.07
00375 c 92.03.17 include core states
00376 c indexing of core states and lmto basis functions for all classes,
00377 c follows that in tb-lmto program
00378 c il,in,im = l,n,m
00379 c ilnm(n,lm) = index of n,l,m
00380 c lm = l*l + l + m + 1
00381 c note: the indexing starts with core first and then valence on top
00382 c       of core(not the same as index generated from nindx)
00383       implicit real*8(a-h,o-z)
00384       dimension nindxv(0:nl-1,nclass),nindxc(0:nl-1,nclass)
00385       dimension ilnm(nn,nl*nl,nclass),
00386      o          ilnmv(nnv,nl*nl,nclass),
00387      o          ilnmc(nnc,nl*nl,nclass),
00388      o          in(nlnmx,nclass),il(nlnmx,nclass),im(nlnmx,nclass),
00389      o         inv(nlnmxv,nclass),ilv(nlnmxv,nclass),imv(nlnmxv,nclass),
00390      o         inc(nlnmxc,nclass),ilc(nlnmxc,nclass),imc(nlnmxc,nclass)
00391       do     ic = 1,nclass
00392         ind       = 0
00393 c core
```

```
00394         do      l = 0,nl-1
00395           l2        = l*l
00396           do      n = 1,nindxc(l,ic)
00397             do      m = 1,2*l+1
00398               ind       = ind + 1
00399               if (ind .gt. nlnmx) call rx( 'idxlnmc: ind > nlnmx')
00400               lm        = l2 + m
00401               il(ind,ic)= l
00402               in(ind,ic)= n
00403               im(ind,ic)= m - l - 1
00404               ilnm(n,lm,ic) = ind
00405               ilc(ind,ic)= l
00406               inc(ind,ic)= n
00407               imc(ind,ic)= m - l - 1
00408               ilnmc(n,lm,ic)= ind
00409             end do
00410           end do
00411         end do
00412 c valence
00413         indv      = 0
00414         do      l = 0,nl-1
00415           l2        = l*l
00416           ncore     = nindxc(l,ic)
00417           do      n = 1,nindxv(l,ic)
00418             if (ncore+n .gt. nn) call rx( 'idxlnmc: ncore+n > nn')
00419             do      m = 1,2*l+1
00420               ind       = ind + 1
00421               indv      = indv + 1
00422               if (ind .gt. nlnmx) call rx( 'idxlnmc: ind > nlnmx')
00423               lm        = l2 + m
00424               il(ind,ic)= l
00425               in(ind,ic)= ncore + n
00426               im(ind,ic)= m - l - 1
00427               ilnm(ncore+n,lm,ic) = ind
00428               ilv(indv,ic)= l
00429               inv(indv,ic)= n
00430               imv(indv,ic)= m - l - 1
00431               ilnmv(n,lm,ic) = indv
00432             end do
00433           end do
00434         end do
00435       end do
00436       return
00437       end
00438
00439       integer function noflmto(nindx,iclass,nl,nclass,natom)
00440 c total number of lmto basis functions
00441       implicit real*8(a-h,o-z)
00442       dimension nindx(0:nl-1,nclass),iclass(natom)
00443       noflmto   = 0
00444       do 1    i = 1,natom
00445         ic        = iclass(i)
00446       do 1    l = 0,nl-1
00447         noflmto   = noflmto + (2*l+1)*nindx(l,ic)
00448     1 continue
00449       return
00450       end
00451
00452       integer function nalwln (nocc,nunocc,nindx,nl,nn)
00453 c gives the number of allowed product radial phi
00454 c nocc(l,n)   = 0,1 ==> unoccupied, occupied
00455 c nunocc(l,n) = 1,0 ==> unoccupied,occupied
00456 c nalwln    = number of allowed phi(l1,n1) phi(l2,n2)
00457       implicit real*8(a-h,o-z)
00458       parameter(lmax=6,nnx=10)
00459       dimension nocc(0:nl-1,nn),nunocc(0:nl-1,nn),
00460      i          nindx(0:nl-1)
00461       dimension icheck(0:lmax,nnx,0:lmax,nnx)
00462       if (nl-1 .gt. lmax) call rx( 'nalwln: increase lmax')
00463       if (nn .gt. nnx) call rx( 'nalwln: increase nnx')
00464       icheck=0
00465       nalwln     = 0
00466       do 10   l1 = 0,nl-1
00467       do 10   n1 = 1,nindx(l1)
00468         if(nocc(l1,n1) .eq. 0)goto 10
00469         do 20   l2 = 0,nl-1
00470         do 20   n2 = 1,nindx(l2)
00471           if(nunocc(l2,n2) .eq. 0)goto 20
00472           if((l1.ne.l2 .or. n1.ne.n2) .and. icheck(l2,n2,l1,n1).ne.0)
00473      . goto 20
00474           nalwln     = nalwln + 1
00475           icheck(l1,n1,l2,n2) = nalwln
00476    20   continue
00477    10 continue
00478       return
00479       end
00480
```

```
00481       integer function nofln(nindx,nl)
00482 c count the number of l,n
00483       implicit real*8(a-h,o-z)
00484       dimension nindx(0:nl-1)
00485       nofln      = 0
00486       do       l = 0,nl-1
00487         nofln      = nofln + nindx(l)
00488       end do
00489       return
00490       end
00491 c--------------------------------------------------------------------
00492       integer function noflnm(nindx,nl)
00493 c number of l,n,m
00494       implicit real*8(a-h,o-z)
00495       dimension nindx(0:nl-1)
00496       noflnm    = 0
00497       do 1    l = 0,nl-1
00498         noflnm    = noflnm + nindx(l)*(2*l+1)
00499     1 continue
00500       return
00501       end
00502
00503       integer function nallow (nocc,nunocc,nindx,nl,nn)
00504 c gives the number of allowed product basis
00505 c nocc(n,l) = 0,1 ==> unoccupied, occupied
00506 c nallow    = number of allowed product basis
00507       implicit real*8(a-h,o-z)
00508       parameter(lmax=6,nnx=10)
00509       dimension nocc(0:nl-1,nn),nunocc(0:nl-1,nn),
00510      i          nindx(0:nl-1)
00511       dimension icheck(0:lmax,nnx,0:lmax,nnx)
00512       if(nl-1 .gt. lmax) call rx( 'nallow: increase lmax')
00513       if(nn .gt. nnx) call rx( 'nallow: increase nnx')
00514       icheck=0
00515       do      l1 = 0,nl-1
00516         do      n1 = 1,nindx(l1)
00517           do      l2 = 0,nl-1
00518             do      n2 = 1,nindx(l2)
00519               icheck(l1,n1,l2,n2) = nocc(l1,n1)*nunocc(l2,n2)
00520               if (l1 .ne. l2 .or. n1 .ne. n2) then
00521                 if (icheck(l1,n1,l2,n2)*icheck(l2,n2,l1,n1) .ne. 0)
00522      . icheck(l1,n1,l2,n2) = 0
00523               endif
00524             end do
00525           end do
00526         end do
00527       end do
00528       nallow     = 0
00529       do 10   l1 = 0,nl-1
00530       do 10   n1 = 1,nindx(l1)
00531       do 10   m1 = 1,2*l1+1
00532       do 10   l2 = 0,nl-1
00533       do 10   n2 = 1,nindx(l2)
00534       do 10   m2 = 1,2*l2+1
00535 c     if (nocc(l1,n1) .eq. 0)goto 10
00536 c     if (nunocc(l2,n2) .eq. 0)goto 10
00537         if (icheck(l1,n1,l2,n2) .eq. 0) goto 10
00538 c temporary
00539         if (l1 .eq. l2 .and. n1.eq.n2 .and. m1.lt.m2)goto 10
00540         nallow     = nallow + 1
00541    10 continue
00542       return
00543       end
00544
00545       subroutine incor  (ncwf,nindxc,iclass,
00546      d                   nl,nnc,nclass,natom,
00547      o                   icore,ncore,nctot)
00548 c 92.03.18
00549 c sorts out allowed core states and count the number of core states
00550 c ncwf(l,n,cl) = 1 ==> allowed, 0 ==> not allowed
00551 c nindxc(l,cl)  = no. core states/l,class
00552 c nl,nnc = max. no. l,n
00553 c icore(i,cl) = index for allowed core states
00554 c ncore(cl)   = no. allowed core states
00555 c nctot      = total no. allowed core states
00556       implicit real*8 (a-h,o-z)
00557       dimension ncwf(0:nl-1,nnc,nclass),nindxc(0:nl-1,nclass),
00558      i          iclass(natom)
00559       dimension icore(nl*nl*nnc,nclass),ncore(nclass)
00560       ncx        = nl*nl*nnc
00561       do      ic = 1,nclass
00562         i          = 0
00563         j          = 0
00564         do       l = 0,nl-1
00565           do       n = 1,nindxc(l,ic)
00566             do       m = -l,l
00567               j          = j + 1
```

```
00568               if (ncwf(l,n,ic) .eq. 1) then
00569                 i          = i + 1
00570                 if (i .gt. ncx) call rx( 'incore: wrong ncx')
00571                 icore(i,ic)= j
00572               endif
00573             end do
00574           end do
00575         end do
00576         ncore(ic)  = i
00577       end do
00578 c total no. allowed core states
00579       nctot      = 0
00580       do       i = 1,natom
00581         ic         = iclass(i)
00582         nctot      = nctot + ncore(ic)
00583       end do
00584       return
00585       end
```

4.7 gwsrc/m_anf.F File Reference

4.8 m_anf.F

```
00001 !> Antiferro condition module. We have line AFcond at the bottom of 'LMTO' file.
00002 !! Currently(feb2016), only laf is used (thus AF symmetry is not used yet for hx0fp0_sc)
00003 !! To access laf, need to call anfcond() in advance.
00004
00005       module m_anf
00006       implicit none
00007       logical:: laf !! - laf: antiferro switch
00008       integer,allocatable:: ibasf(:) !! - ibasf(ibas) specify AF pair atom.
00009 c      integer:: natom
00010 c     ,ldima(:),iantiferro(:),iclasst(:)
00011 c      real(8),allocatable:: pos(:,:),anfvec(:),qlat(:,:),plat(:,:)
00012       contains
00013
00014       subroutine anfcond()
00015       implicit none
00016       integer,allocatable:: iantiferro(:)
00017       integer:: ifile_handle,ilmto,ildima,ificlass
00018       character(256):: aaa,keyplat
00019       real(8)::vecs(3),vece(3),basdiff(3)
00020       integer:: ibas,lkeyplat,i,ibasx,natom
00021       character(3)::iaaa
00022 !! read LMTO file
00023       write(6,*) 'anfcond:'
00024       ilmto=ifile_handle()
00025       open(ilmto,file='LMTO')
00026       do
00027         read(ilmto,"(a)",end=1011,err=1011) aaa
00028         aaa = adjustl(aaa)
00029 c$$$c        print *,trim(aaa)
00030 c$$$        if(trim(aaa)=='primitive lattice vectors (plat)') then
00031 c$$$          allocate(plat(3,3),qlat(3,3))
00032 c$$$          do i=1,3
00033 c$$$           read(ilmto,*) plat(1:3,i)
00034 c$$$          enddo
00035 c$$$          call dinv33x(plat,qlat)
00036 c$$$        endif
00037         if(trim(aaa)=='number of atoms (natom)') then
00038           read(ilmto,*) natom
00039           read(ilmto,*)
00040           allocate(iantiferro(natom),ibasf(natom))
00041 c$$$          allocate(iantiferro(natom),pos(3,natom))
00042 c$$$          iantiferro=0
00043 c$$$          do ibas = 1,natom
00044 c$$$            read(ilmto,*) pos(1:3,ibas)
00045 c$$$            write(6,*) pos(1:3,ibas)
00046 c$$$          enddo
00047         endif
00048         if(aaa(1:6)=='AFcond') then
00049           read(ilmto,*) iantiferro(1:natom)
00050           ibasf=-999
00051           do ibas=1,natom
00052             do ibasx=ibas+1,natom
00053               if(abs(iantiferro(ibas))/=0 .and. iantiferro(ibas)+iantiferro(ibasx)==0) then
00054                 ibasf(ibas)=ibasx
00055                 exit
00056               endif
00057             enddo
00058             if(ibasf(ibas)/=-999) write(6,"(a,2i5)")' AF pair: ibas ibasf(ibas)=',ibas,ibasf(ibas)
```

```
00059           enddo
00060         endif
00061       enddo
00062  1011 continue
00063       close(ilmto)
00064       if(sum(abs(iantiferro))==0) then
00065         laf=.false. !no AF case
00066         return
00067       endif
00068 !! Antiferro case --------------
00069       laf=.true.
00070       if(laf) write(6,"(a,100i4)") ' Antiferromode=',iantiferro
00071       end subroutine anfcond
00072       end module
00073
00074 !! ---- followings are wrong
00075 c$$$!! Read ldima
00076 c$$$      ildima=ifile_handle()
00077 c$$$      open(ildima,file='ldima')
00078 c$$$      read(ildima,'(a3)') iaaa
00079 c$$$      close(ildima)
00080 c$$$! Is this correct description? takao2015may
00081 c$$$!     ... June2007 for floating orbitals
00082 c$$$!     ldima is generaged by lmfgw; it contains number of MTO including
00083 c$$$!     floating orbital, and positions.
00084 c$$$      if(iaaa=='***') then
00085 c$$$        open(ildima,file='ldima')
00086 c$$$        read(ildima,*) iaaa,natom
00087 c$$$        deallocate(pos)
00088 c$$$        allocate(ldima(natom),pos(1:3,natom) )
00089 c$$$        do ibas = 1,natom
00090 c$$$          read(ildima,*) ldima(ibas),pos(1:3,ibas)
00091 c$$$          write(6,"('ldima pos=',i5,3f10.4)")ldima(ibas),pos(1:3,ibas)
00092 c$$$        enddo
00093 c$$$        close(ildima)
00094 c$$$      else
00095 c$$$        open(ildima,file='ldima')
00096 c$$$        allocate(ldima(natom))
00097 c$$$        do ibas=1,natom
00098 c$$$          read(ildima,*) ldima(ibas)
00099 c$$$          write(6,*) 'ldima=',ldima(ibas)
00100 c$$$        enddo
00101 c$$$        close(ildima)
00102 c$$$      endif
00103 c$$$!! Read CLASS
00104 c$$$      ificlass=ifile_handle()
00105 c$$$      open (ificlass,file='CLASS')
00106 c$$$      allocate(iclasst(natom))
00107 c$$$      write(6,*)'  --- Readingin CLASS info ---'
00108 c$$$      do ibas = 1,natom
00109 c$$$        read(ificlass,*) ibasx, iclasst(ibas)
00110 c$$$      enddo
00111 c$$$      close(ificlass)
00112 c$$$
00113 c$$$!! Get anfvec and ibasf
00114 c$$$      allocate(ibasf(natom))
00115 c$$$c          ifianf = 211
00116 c$$$c          open(ifianf,file='ANFcond')
00117 c$$$c          read(ifianf,*)
00118 c$$$c          read(ifianf,*)
00119 c$$$c          read(ifianf,*) anfvec(1:3)
00120 c$$$!!
00121 c$$$      do ibas=1,natom
00122 c$$$        if(iantiferro(ibas)==-1) then
00123 c$$$          vecs = pos(:,ibas)
00124 c$$$          iclasst(ibas)=  999 !overwrite by 999
00125 c$$$        elseif(iantiferro(ibas)==1) then
00126 c$$$          vece = pos(:,ibas)
00127 c$$$          iclasst(ibas)=  999
00128 c$$$        endif
00129 c$$$      enddo
00130 c$$$!!
00131 c$$$      allocate(anfvec(3))
00132 c$$$      anfvec = vece-vecs
00133 c$$$      do ibas=1,natom
00134 c$$$        do ibasx=1,natom
00135 c$$$          do i=1,3
00136 c$$$            basdiff(i)= sum((pos(:,ibas)+anfvec-pos(:,ibasx))*qlat(:,i))
00137 c$$$          enddo
00138 c$$$c         write(6,"(a,4i4,3f13.6)")' ibas ibasx iclass iclassx basdiff=',ibas,ibasx,iclasst(ibas),
      iclasst(ibasx),basdiff
00139 c$$$          if(sum(abs(basdiff-anint(basdiff)))<1d-6.and.iclasst(ibas)==iclasst(ibasx)) then
00140 c$$$            ibasf(ibas)=ibasx
00141 c$$$            write(6,"(a,2i5)")' ibas ibasf=',ibas,ibasf(ibas)
00142 c$$$            goto 888
00143 c$$$          endif
00144 c$$$        enddo
```

```
00145 c$$$        call rx('m_anf: ibasf did not found')
00146 c$$$ 888    continue
00147 c$$$      enddo
00148 c$$$      write(6,'(" antiferro translation vector=",3f13.6)') anfvec
00149 c$$$      end subroutine anfcond
00150 c$$$      end module
00151
```

4.9 gwsrc/m_freq.F File Reference

Data Types

- module m_freq

  Frequency mesh generator.

4.10 m_freq.F

```
00001 !>Frequency mesh generator
00002 !! - OUTPUT
00003 !!   - fhris :histgram bins to accumlate im part
00004 !!   - freq_r: omega along real axis
00005 !!   - freq_i: omega along imag axis
00006 !!   - wiw: integration weight along im axis
00007 !!   - npm: npm=1 means only positive omega;npm=2 means positive and negative omega.
00008 !! - NOTE: change of frequency mesh defined here may destroy consistency or not. Need check
00009       module m_freq
00010       real(8),allocatable:: frhis(:),freq_r(:),freq_i(:),wiw(:)
00011       integer:: nwhis,npm,nw_i,nw
00012
00013       contains
00014 !> Get data set for m_freq. All arguments are input.
00015 !! - This read GWinput (dw,omg_c) and TimeReversal()
00016 !! - All arguments are input
00017       subroutine getfreq(epsmode,realomega,imagomega,tetra,omg2max,nw_input,niw,ua,mpi__root)
00018       use keyvalue,only: getkeyvalue
00019       implicit none
00020       integer,intent(in):: niw,nw_input
00021       logical,intent(in):: realomega,imagomega,tetra,mpi__root,epsmode
00022       real(8),intent(in):: omg2max,ua
00023
00024       real(8),allocatable:: freqx(:),wx(:),expa(:)
00025       logical:: timereversal,onceww
00026       integer:: nw2,iw,ihis
00027       real(8)::omg_c,dw,omg2
00028       real(8), allocatable :: freqr2(:)
00029       real(8)::  pi       = 4d0*datan(1d0)
00030
00031       logical,save:: done=.false.
00032       if(done) call rx('gerfreq is already done') !sanity check
00033       done =.true.
00034
00035       nw = nw_input
00036 !! We first accumulate Imaginary parts.
00037 !! Then it is K-K transformed to obtain real part.
00038       call getkeyvalue("GWinput","dw",dw )
00039       call getkeyvalue("GWinput","omg_c",omg_c )
00040       write(6,"('dw, omg_c= ',2f13.5)") dw, omg_c
00041 !! histogram bin divisions
00042       nw2=int(omg_c/dw*( sqrt(1.+2*omg2max/omg_c)-1. ) )+1+3 !+3 for margin
00043       allocate(freqr2(nw2))     !+1 b/c (iw-1)
00044       do iw=1,nw2
00045          freqr2(iw)=dw*(iw-1)+dw**2/2./omg_c*(iw-1)**2
00046       enddo                     !linear + quadratic term
00047       if (nw2 < 2 ) call rx( "m_freq: nw2 < 2")
00048       if (dw*(nw-2) > freqr2(nw2-1)) call rx("m_freq: dw*(nw-2) > freqr2(nw2-1)")
00049       nwhis = nw2-1
00050       allocate(frhis(1:nwhis+1))
00051       frhis = freqr2(1:nwhis+1)
00052       write(6,*)' we set frhis nwhis=',nwhis
00053 !! frhis_m
00054       nw=nw2-1       ! nw+1 is how many points of real omega we use
00055                      ! for dressed coulomb line W(iw=0:nw) iw=0 corresponds omg=0
00056                      ! maximum nw=nw2-1 because nwhis=nw2-1
00057       do iw=3,nw2-1
00058                      !nw is chosen from condition that frhis_m(nw-3)<dw*(nw_input-3) <frhis_m(nw-2).
00059                      !Here frhis_m(iw)= (freqr2(iw)+freqr2(iw+1))/2d0
00060                      !nw was constructed such that omg=dw*(nw-2)> all relevant frequensies needed
```

```
00061                     ! for correlation Coulomb Wc(omg),
00062                     ! and one more point omg=dw*(nw-1) needed for extrapolation.
00063                     ! Now, frhis_m(nw-1)> all relevent frequensies for Wc(omg)
00064                     ! and one more point omg=frhis_m(nw) needed for extropolation
00065                     ! used in subroutine alagr3z in  sxcf.f.
00066          omg2 = (freqr2(iw-2)+freqr2(iw-1))/2d0
00067          if (omg2 > dw*(nw_input-3)) then
00068             nw=iw
00069             exit     ! 'nw_input' is only used to get maximum frequency for
00070                      !  dressed coulomb line
00071          endif
00072       enddo
00073
00074       if(epsmode) then
00075         nw  = nwhis-1
00076 c        niw = 0
00077       endif
00078
00079       allocate(freq_r(0:nw))
00080       freq_r(0)=0d0
00081       do iw=1,nw
00082         freq_r(iw)=(frhis(iw)+frhis(iw+1))/2d0
00083       enddo
00084
00085 !! Plot frhis ----------
00086       if(onceww(1)) then
00087         write(6,*)' --- Frequency bins to accumulate Im part  (a.u.) are ---- '
00088         do ihis= 1, min(10,nwhis)
00089           write(6,"(' ihis Init  End=', i5,2f13.6)") ihis,frhis(ihis),frhis(ihis+1)
00090         enddo
00091         write(6,*) 'ihis  ...'
00092         do ihis= max(min(10,nwhis),nwhis-10), nwhis
00093           write(6,"(' ihis Init  End=', i5,2f13.6)") ihis,frhis(ihis),frhis(ihis+1)
00094         enddo
00095       endif
00096 !! Timereversal=F is implimented only for tetra=T and sergeyv=T
00097       npm=1
00098       nw_i=0
00099       if(.not.timereversal()) then
00100          write(6,"('TimeReversal off mode')")
00101          npm=2
00102          nw_i=-nw
00103          if(.not.tetra)      call rx( ' tetra=T for timereversal=off')
00104       endif
00105       write(6,*)'Timereversal=',timereversal()
00106       if(realomega .and. mpi__root) then
00107          open(unit=3111,file='freq_r') !write number of frequency
00108                                        !points nwp and frequensies in 'freq_r' file
00109          write(3111,"(2i8,'  !(a.u.=2Ry)')") nw+1, nw_i
00110          do iw= nw_i,-1
00111             write(3111,"(d23.15,2x,i6)") -freq_r(-iw),iw !This file is reffere by hsfp0 and so.
00112          enddo
00113          do iw= 0,nw
00114             write(3111,"(d23.15,2x,i6)") freq_r(iw),iw !This file is reffere by hsfp0 and so.
00115          enddo
00116          close(3111)
00117       endif
00118 !! set freq_i
00119       if (imagomega) then
00120          write(6,*)' freqimg: niw =',niw
00121          allocate( freq_i(niw) ,freqx(niw),wx(niw),expa(niw) )
00122          call freq01(niw,ua,
00123      o        freqx,freq_i,wx,expa)
00124          ! Generate gaussian frequencies x between (0,1) and w=(1-x)/x
00125          allocate(wiw(niw))
00126          do iw=1,niw
00127            wiw(iw)=wx(iw)/(2d0*pi*freqx(iw)*freqx(iw))
00128          enddo
00129          deallocate(freqx,wx,expa)
00130       endif
00131       end subroutine getfreq
00132       end module m_freq
```

4.11 gwsrc/m_hamindex.F File Reference

Data Types

- module m_hamindex

  This is in lm7K/subs/m_hamindex.F and in fpgw/gwsrc/m_hamindex.F We will need to unify make system and source code in fpgw and lmf. norbtx is given in gwsrc/readeigen.F init_readeigen2.

4.12 m_hamindex.F

```
00001 !> This is in lm7K/subs/m_hamindex.F and in fpgw/gwsrc/m_hamindex.F
00002 !! We will need to unify make system and source code in fpgw and lmf.
00003 !! norbtx is given in gwsrc/readeigen.F init_readeigen2
00004       module m_hamindex
00005       implicit none
00006       integer,parameter:: null=-999999
00007       integer:: ngrp=null, lxx=null, kxx=null,norbmto=null,norbtx=null
00008       integer:: imx=null,nbas,ndimham=null
00009       integer:: nqtt,nqi, nqnum,ngpmx
00010       integer,allocatable:: ltab(:),ktab(:),offl(:),ispec(:), iclasst(:),offlrev(:,:,:),ibastab(:)
00011       integer,allocatable:: iqimap(:),iqmap(:),igmap(:),invgx(:),miat(:,:),ibasindex(:)
     !,ngvecp(:,:,:),ngvecprev(:,:,:,:)
00012       real(8),allocatable:: symops(:,:,:),ag(:,:),tiat(:,:,:),shtvg(:,:), dlmm(:,:,:,:),qq(:,:)
00013       real(8):: plat(3,3),qlat(3,3)
00014       real(8),allocatable:: qtt(:,:),qtti(:,:)
00015       integer,allocatable:: igv2(:,:,:),napwk(:),igv2rev(:,:,:,:)
00016       integer::  napwmx=null,lxxa=null
00017       logical,private:: debug=.false.
00018       contains
00019
00020 !> get index ikt such that for qin(:)=qq(:,ikt)
00021       integer function getikt(qin) !return
00022       integer::i
00023       real(8):: qin(3)
00024 c      if(debug) print *,'nkt=',nkt
00025       do i=1, nqnum !*2 !nkt
00026         if(debug) print *,i,qin, qq(:,i)
00027         if(sum(abs(qin-qq(:,i)))<1d-8) then
00028           getikt=i
00029           return
00030         endif
00031       enddo
00032       print *,' getikt: xxx error nqnum qin=',nqnum,qin
00033       do i=1, nqnum !*2 !nkt
00034         write(*,"('i qq=',i3,3f11.5)")i, qq(:,i)
00035       enddo
00036       call rx( ' getikt can not find ikt for given q')
00037       end function
00038
00039 !> write info for wave rotation.
00040       subroutine writehamindex()
00041       integer(4):: ifi
00042       logical::pmton
00043       logical,save:: done=.false.
00044       if(done) call rx('writehamindex is already done')
00045       done=.true.
00046       ifi=1789
00047       open(ifi,file='HAMindex',form='unformatted')
00048       write(ifi)ngrp,nbas,kxx,lxx,nqtt,nqi,nqnum,imx,ngpmx,norbmto
00049       write(ifi)symops,ag,invgx,miat,tiat,shtvg,qtt,qtti,iqmap,igmap,iqimap
00050       write(ifi)lxxa
00051       write(ifi)dlmm
00052       write(ifi)ibastab,ltab,ktab,offl,offlrev !for rotation of MTO. recovered sep2012 for EIBZ for hsfp0
00053       write(ifi)qq !,ngvecp,ngvecprev
00054       write(ifi)plat,qlat,napwmx
00055       if(napwmx/=0) then !for APW rotation used in rotwvigg
00056          write(ifi) igv2,napwk,igv2rev
00057       endif
00058       close(ifi)
00059       end subroutine writehamindex
00060
00061 !> read info for wave rotation.
00062       subroutine readhamindex()
00063       integer(4):: ifi,nkt
00064       logical::pmton
00065       logical,save:: done=.false.
00066       if(done) call rx('readhamindex is already done')
00067       done=.true.
00068       ifi=1789
00069       open(ifi,file='HAMindex',form='unformatted')
00070       read(ifi)ngrp,nbas,kxx,lxx,nqtt,nqi,nqnum,imx,ngpmx,norbmto
00071       allocate(symops(3,3,ngrp),ag(3,ngrp),qtt(3,nqtt),qtti(3,nqi))
00072       allocate(invgx(ngrp),miat(nbas,ngrp),tiat(3,nbas,ngrp),shtvg(3,ngrp))
00073       allocate(iqmap(nqtt),igmap(nqtt),iqimap(nqtt))
00074       write(6,*) 'ngrp=',ngrp
00075       read(ifi)symops,ag,invgx,miat,tiat,shtvg,qtt,qtti,iqmap,igmap,iqimap
00076       allocate( ltab(norbmto),ktab(norbmto),offl(norbmto),ibastab(norbmto) )
00077       allocate( offlrev(nbas,0:lxx,kxx))
00078       read(ifi) lxxa
00079       allocate( dlmm(-lxxa:lxxa, -lxxa:lxxa, 0:lxxa, ngrp))
00080       read(ifi) dlmm
00081       read(ifi)ibastab,ltab,ktab,offl,offlrev
00082 c      allocate( ngvecprev(-imx:imx,-imx:imx,-imx:imx,nqnum) )
00083 c      allocate( ngvecp(3,ngpmx,nqnum) )
```

```
00084       allocate( qq(3,nqnum)) !this was qq(3,nqnum*2) until Aug2012 when shorbz had been used.
00085       read(ifi)qq !,ngvecp,ngvecprev
00086       read(ifi)plat,qlat,napwmx
00087       if(napwmx/=0)then !for APW rotation used in rotwvigg
00088         nkt=nqtt
00089         allocate( igv2(3,napwmx,nkt) )
00090         allocate( napwk(nkt))
00091         allocate( igv2rev(-imx:imx,-imx:imx,-imx:imx,nkt) )
00092         read(ifi) igv2,napwk,igv2rev
00093       endif
00094       close(ifi)
00095       done=.true.
00096       end subroutine readhamindex
00097       end module
00098
00099
```

4.13 gwsrc/m_tetwt.F File Reference

Data Types

- module m_tetwt

  Get the weights and index for tetrahedron method for the Lindhard function.

4.14 m_tetwt.F

```
00001 !> Get the weights and index for tetrahedron method for the Lindhard function.
00002 !!    - nbnb = total number of weight.
00003 !!    - n1b  = band index for occ.   1\ge n1b \ge nband+nctot.
00004 !!       "Valence index->core index" ordering(Core index follows valence index).
00005 !!    - n2b  = band index for unocc. 1\ge n2b \ge nband
00006 !!    - wwk(ibib,...)  = (complex)weight for the pair for n1b(ibib...),n2b(ibib...).
00007 !!
00008 !! - NOTE: 'call getbzdata1' generates nteti,ntetf,... See mkqg.F about how to call it.
00009 !!
00010       module m_tetwt
00011       real(8),allocatable :: whw(:)
00012       integer,allocatable:: ihw(:,:,:),nhw(:,:,:),jhw(:,:,:),ibjb(:,:,:,:)
00013       integer:: nbnbx,nhwtot
00014       integer,allocatable :: n1b(:,:,:),n2b(:,:,:),nbnb(:,:)
00015 !!
00016       contains !! ------------------------------------------------------------------------
00017 !! routine
00018       subroutine gettetwt(q,iq,is,isf,nwgt) !this routine set output data above.
00019 !! input data; read only
00020       use m_readeigen,only:   readeval !we assume init_readeval is called already
00021       use m_genallcf_v3,only: ecore,nctot,ef    !we assume genallcf_v3 called already.
00022       use m_read_bzdata,only: nqbz,qbas,ginv,nqbzw,nteti,ntetf,idtetf,qbzw,ib1bz,nqibz,qbz
00023         ! we assume read_bzdata is called already
00024       use m_freq,only:                  !we assume getfreq is called already.
00025      &   frhis, nwhis,npm !output of getfreq
00026       use m_zmel,only: nband
00027
00028       implicit none
00029       real(8),intent(in):: q(3)
00030       integer,intent(in):: is,isf,iq,nwgt(*)
00031
00032       real(4),allocatable :: demin(:,:,:,:),demax(:,:,:,:)
00033       logical,allocatable :: iwgt(:,:,:,:)
00034       integer,allocatable:: nbnbtt(:,:),noccxvv(:) !  &         idtetf(:,:),ib1bz(:)
00035       real(8),allocatable:: ekxx1(:,:),ekxx2(:,:) !qbzw(:,: )
00036       logical :: eibzmode,tetra,tmpwwk=.false.,debug,eibz4x0
00037       integer:: kx,ncc,job,jpm,noccxvx(2)=-9999,ik,jhwtot,ib1,ib2,ibib,noccx,noccxv,verbose
00038
00039       tetra=.true.
00040       eibzmode = eibz4x0()
00041       debug=.false.
00042       if(verbose()>=100) debug=.true.
00043
00044       if(.not.allocated(nbnb)) allocate( nbnb(nqbz,npm))
00045       allocate( nbnbtt(nqbz,npm),ekxx1(nband,nqbz),ekxx2(nband,nqbz))
00046
00047 !!===========tetraini block tetra==.true.================================lini
00048 c      if(tetra) then
00049       write(6,"(' tetra mode nqbz nband ispin q=',2i7,i2,3f13.6)") nqbz,nband,is,q
00050 !!     ekxx1 for rk
00051 !!     ekxx2 for q+rk See tetwt4
```

```
00052       do kx = 1, nqbz
00053         call readeval(qbz(:,kx),   is,  ekxx1(1:nband, kx) )
00054         call readeval(q+qbz(:,kx), isf, ekxx2(1:nband, kx) )
00055       enddo
00056 c     takao-feb/2002 i replaced tetwt4(1d30) with tetwt5(job=0) -----
00057 c     ... get pairs(n1b n2b) with non-zero tetrahedron wieghts.
00058 c     the pairs are not dependent on the energy omega
00059 c     in the denominator of the dielectric function.
00060       write(6,"(' -- First tetwt5 is to get size of array --')")
00061       job = 0
00062       if(npm==1) then
00063         ncc=0
00064       else
00065         ncc=nctot
00066       endif
00067       allocate( demin(nband+nctot,nband+ncc,nqbz,npm),
00068      &          demax(nband+nctot,nband+ncc,nqbz,npm) )
00069       allocate( iwgt(nband+nctot,nband+ncc,nqbz,npm) )
00070 !     wgt, demin, demax may require too much memory in epsilon mode.
00071 !     We will have to remove these memory allocations in future.
00072 !     tetwt5x_dtet2 can be very slow because of these poor memory allocation.
00073       if(nctot==0) then
00074         deallocate(ecore)
00075         allocate(ecore(1,2))    !this is dummry
00076       endif
00077       allocate(ibjb(1,1,1,1),ihw(1,1,1),jhw(1,1,1),nhw(1,1,1),whw(1)) !dummy
00078       call tetwt5x_dtet4(npm,ncc,
00079      i q, ekxx1, ekxx2, qbas,ginv,ef,
00080      d ntetf,nqbzw, nband,nqbz,
00081      i nctot,ecore(1,is),idtetf,qbzw,ib1bz,
00082      i job,
00083      o iwgt,nbnb,               !job=0
00084      o demin,demax,             !job=0
00085      i frhis, nwhis,            ! job=1    not-used
00086      i nbnbx,ibjb,nhwtot,       ! job=1    not-used
00087      i ihw,nhw,jhw,             ! job=1    not-used
00088      o whw,                     ! job=1    not-used
00089      i iq,is,isf,nqibz, eibzmode,nwgt)
00090       deallocate(ibjb,ihw,jhw,nhw,whw) !dummy
00091       nbnbx = maxval(nbnb(1:nqbz,1:npm)) !nbnbx = nbnbxx
00092       if(debug) write(6,*)' nbnbx=',nbnbx
00093       allocate(  n1b(nbnbx,nqbz,npm)
00094      &          ,n2b(nbnbx,nqbz,npm))
00095       n1b=0; n2b=0
00096       do jpm=1,npm
00097         call rsvwwk00_4(jpm, iwgt(1,1,1,jpm),nqbz,nband,nctot,ncc, nbnbx,
00098      o   n1b(1,1,jpm), n2b(1,1,jpm), noccxvx(jpm), nbnbtt(1,jpm))
00099       enddo
00100       if(debug) then
00101         do kx  = 1, nqbz
00102           do jpm = 1, npm
00103             write(6,"('jpm kx  minval n1b n2b=',4i5)")jpm,kx,
00104      &       minval(n1b(1:nbnb(kx,jpm),kx,jpm)),
00105      &       minval(n2b(1:nbnb(kx,jpm),kx,jpm))
00106           enddo
00107         enddo
00108       endif
00109       if(sum(abs(nbnb-nbnbtt))/=0)then
00110         do ik=1,nqbz
00111           write(6,*)
00112           write(6,*)"nbnb  =",nbnb(ik,:)
00113           write(6,*)"nbnbtt=",nbnbtt(ik,:)
00114         enddo
00115         call rx( 'hx0fp0:sum(nbnb-nbnbtt)/=0')
00116       endif
00117       noccxv = maxval(noccxvx)
00118       noccx  = nctot + noccxv
00119       write(6,*)' Tetra mode: nctot noccxv= ',nctot,noccxv
00120       deallocate(iwgt)
00121 c      endif
00122 c=========end of tetraini block============================================1end
00123
00124 !! TetrahedronWeight_5 block. tetwt5  ixc==,4,6,11 =======4ini
00125 c     if(ixc==11) then !sf 21May02
00126 c     --- method(tetwt5) for the tetrahedron weight
00127 !     Histogram secstions are specified by frhis(1:nwp)
00128 !     The 1st   bin  is     [frhis(1),  frhis(2)]   ...
00129 !     The last  bin  is     [frhis(nw), frhis(nwp)].
00130 !     nwp=nw+1; frhis(1)=0
00131 !     takao-feb/2002
00132       if(frhis(1)/=0d0) call rx( ' hx0fp0: we assume frhis(1)=0d0')
00133       write(6,*)' -----------------nbnbx nqbz= ',nbnbx,nqbz
00134 !!     ... make index sets
00135       allocate(ihw(nbnbx,nqbz,npm),nhw(nbnbx,nqbz,npm),jhw(nbnbx,nqbz,npm))
00136       ihw=0; nhw=0; jhw=0
00137       jhwtot = 1
00138       do jpm =1,npm
```

```
00139         do ik    = 1,nqbz
00140           do ibib = 1,nbnb(ik,jpm)
00141             call hisrange( frhis, nwhis,
00142      i       demin(n1b(ibib,ik,jpm),n2b(ibib,ik,jpm),ik,jpm),
00143      i       demax(n1b(ibib,ik,jpm),n2b(ibib,ik,jpm),ik,jpm),
00144      o       ihw(ibib,ik,jpm),nhw(ibib,ik,jpm))
00145             jhw(ibib,ik,jpm)= jhwtot
00146             jhwtot = jhwtot + nhw(ibib,ik,jpm)
00147           enddo
00148         enddo
00149       enddo
00150       nhwtot = jhwtot-1
00151       write(6,*)' nhwtot=',nhwtot
00152       deallocate(demin,demax)
00153       allocate( whw(nhwtot),    ! histo-weight
00154      & ibjb(nctot+nband,nband+ncc,nqbz,npm) )
00155       whw=0d0
00156       ibjb = 0
00157       do jpm=1,npm
00158         do ik    = 1,nqbz
00159           do ibib = 1,nbnb(ik,jpm)
00160             ib1  = n1b(ibib,ik,jpm)
00161             ib2  = n2b(ibib,ik,jpm)
00162             ibjb(ib1,ib2,ik,jpm) = ibib
00163           enddo
00164         enddo
00165       enddo
00166 !!     ... Generate the histogram weights whw
00167       job=1
00168       write(6,*) 'goto tetwt5x_dtet4 job=',job
00169       allocate(demin(1,1,1,1),demax(1,1,1,1),iwgt(1,1,1,1)) !dummy
00170       call tetwt5x_dtet4(  npm,ncc,
00171      i q, ekxx1, ekxx2, qbas,ginv,ef,
00172      d ntetf,nqbzw, nband,nqbz,
00173      i nctot,ecore(1,is),idtetf,qbzw,ib1bz,
00174      i job,
00175      o iwgt,nbnb,                ! job=0
00176      o demin,demax,              ! job=0
00177      i frhis,nwhis,              ! job=1
00178      i nbnbx,ibjb,nhwtot,        ! job=1
00179      i ihw,nhw,jhw,              ! job=1
00180      o whw,                      ! job=1
00181      i iq,is,isf,nqibz, eibzmode,nwgt)
00182       deallocate(demin,demax,iwgt) !duumy
00183 !! ======TetrahedronWeight_5 block end =========
00184       end subroutine
00185       end module
```

4.15 gwsrc/m_zmel.F File Reference

Data Types

- module m_zmel

  Get the matrix element zmel = ZO^-1 <MPB psi|psi> , where ZO is ppovlz. To use this module, set data in this module, and call "call get_zmelt" or "call get_zmelt2". Then we have matrix elements zmel (exchange=F for correlation) or zmeltt (exchange=T). In future, they may be unified...

Functions/Subroutines

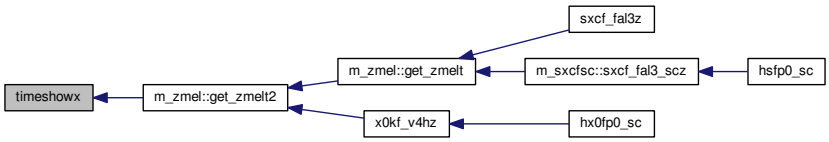
- subroutine timeshowx (info)

4.15.1 Function/Subroutine Documentation

4.15.1.1 subroutine timeshowx (character∗(∗) *info*)

Definition at line 382 of file m_zmel.F.

Here is the caller graph for this function:



4.16 m_zmel.F

```
00001 !> Get the matrix element zmel =  ZO^-1 <MPB psi|psi> , where ZO is ppovlz.
00002 !! To use this module, set data in this module, and call "call get_zmelt" or "call get_zmelt2".
00003 !! Then we have matrix elements zmel (exchange=F for correlation)
00004 !! or zmeltt (exchange=T). In future, they may be unified...
00005       module m_zmel
00006
00007 !! Base data for crystal structure.
00008 !! these are set by 'call genallcf_v3' usually in the main routine.
00009       use m_genallcf_v3,only:
00010      i  nclass,natom,nspin,nl,nn,nnv,nnc, ngrp,
00011      i  nlmto,nlnx,nlnxv,nlnxc,nlnmx,nlnmxv,nlnmxc, niw,nw,
00012      i  alat,ef, diw,dw,delta,deltaw,esmr,symgrp,iclass,nlnmv,
00013 c  clabl,nindxv,nindxc,ncwf,
00014 c     &  il, in, im, ilnm, nlnm, ilv,inv,imv,  ilnmv,
00015 c     &  ilc,inc,imc,  ilnmc,
00016      i   invg, nlnmc, !nindx,konf
00017      i  icore,ncore,occv,unoccv ,
00018      i  occc,unoccc, nocc, nunocc, plat, pos,z,ecore,  symgg,
00019      i  done_genallcf_v3
00020 !! Get eigenfuncitons. cphi is coefficients of MTO+lo part, geig is IPW parts.
00021 !! Before calling them (get coefficients of eigen funcitons),
00022 !! We need to call init_readeigen, init_readeigen2 in main rouitne.
00023       use m_readeigen,only: readcphi,readgeig
00024 !! Basic data set to get zmel*
00025 !! these are set by 'call rdpp' in main routine
00026       use m_rdpp,only:
00027      i  nblocha, lx, nx,  ppbrd , mdimx,nbloch, cgr,
00028      i  done_rdpp
00029 !! BZ data. To set these data 'call read_BZDATA' in main rouitne.
00030       use m_read_bzdata,only:
00031      i  nqbz,nqibz,  qbas,ginv,qbz,qibz,wbz,
00032      i  done_read_bzdata
00033 !! general purpose routine to read values in GWinput file.
00034       use keyvalue,only: getkeyvalue
00035
00036 !! ------------------------------------------
00037       integer,parameter:: NULL=-99999
00038 !! These are set by mptauof in main routine. 'call mptauof'
00039       integer,allocatable :: miat(:,:)
00040       real(8),allocatable :: tiat(:,:,:),shtvg(:,:)
00041 !! We set these values in main routine.
00042       integer:: nband=NULL,ngcmx=NULL,ngpmx=NULL,ntq=NULL !set in main routine
00043       integer,allocatable :: itq(:)                       !set in main routine
00044       real(8),allocatable :: ppbir(:,:,:)                 !set in main routine, call pbafp_v2.
00045       complex(8),allocatable,target :: ppovlz(:,:)        !set in main rouitne
00046 c      integer,allocatable:: imdim(:)                     !set in main routine
00047
00048 !! OUTPUT:  zmel for exchange=F, zmeltt for exchange=T.
00049       complex(8),allocatable :: zmel(:,:,:),zmeltt(:,:,:) !output
00050
00051 !! local save.
00052       real(8),private:: qbasinv(3,3),q_bk(3)=1d10,qk_bk(3)=1d0
00053       logical,private:: init=.true.
00054       complex(8),allocatable,private :: cphiq(:,:), cphim(:,:)
00055       real(8),allocatable,private :: rmelt(:,:,:),cmelt(:,:,:)
00056       integer,private::kxold=-9999
00057
00058       contains
00059 !! ------------------------------------
00060       subroutine get_zmelt(exchange,q,kx, kvec,irot,rkvec,kr,isp,  ngc,ngb,nmmax,nqmax, nctot,ncc)
00061 !! Get <phiq(q,ncc+nqmax,ispq) |phim(q-rkvec,nctot+nmmax,ispm) MPB(rkvec,ngb)> ZO^-1
00062 !!
00063 !! ncc=0
00064 !! kvec is in the IBZ, rk = Rot_irot(kvec), kx,kr are dummy.
00065 !! \parameter all inputs
```

```
00066 !! \parameter output=rmelt,clemt  matrix <MPB psi|psi>
00067       implicit none
00068       logical:: exchange
00069       integer:: kx,kr,isp,ngc,ngb,nmmax,nqmax,irot,ispq,ispm,nmini,nqini, nctot,ncc
00070       real(8) ::  quu(3),q(3), kvec(3),rkvec(3)
00071       ispq = isp
00072       ispm = isp
00073       nmini=1
00074       nqini=1
00075       call get_zmelt2(exchange,
00076      &    kvec,irot,rkvec,ngc,ngb,    !MPB        for  MPB_rkvec
00077      &      nmini,nmmax,ispm,nctot,  !middle-phi for  phi_{q-rkvec}
00078      &    q,nqini,nqmax,ispq,ncc  )  !end-phi    for  phi_q
00079       end subroutine get_zmelt
00080 !! ----------------------------------------
00081 cold  ntqxx--->nqmax
00082 cold  nbmax -->nmmax
00083 !!note: For usual correlation mode, I think nctot=0
00084 !!note: For self-energy mode;   we calculate <iq1|\Sigma |iq2> , where iq1 and iq2 are in nqmax.
00085 !!       nstate = nctot+nmmax
00086 !!       allocate(zmeltt(ngb,    nstate,  nqmax))
00087 !!       zmeltt= < MPB     phi    | phi   > (but true matrix elements are for <phi|phi MPB> (complex
      conjugate).
00088 !!               <rkvec q-rkvec  |  q    >
00089  !                      cphim    | cphiq
00090 !                       ispm     | ispq
00091 !            nctot+  nmini:nmmax | ncc + nqini:nqmax
00092 !                    middle state| end state
00093 !
00094 !!--- For dielectric funciton, we use irot=1 kvec=rkvec=q. We calulate \chi(q).
00095 !!              q      rkvec     | q + rkvec
00096 !                   nkmin:nkmax | nkqmin:nkqmax
00097 !                  (we fix nkmin=1)
00098 !           or
00099 !              nt0=nkmax-nkmin+1 | ntp0=nkqmax-nkqmin+1
00100 !                      1:nt0     | 1:ntp0
00101 !                         occ    | unocc
00102 !                       (cphi_k   | cphi_kq !in x0kf)
00103 !                    middle state| end state
00104 !
00105 !!     rkvec= rk(:,k)-qq  ! <phi(q+rk,nqmax)|phi(rk,nctot+nmmax)  MPB(q,ngb )>
00106 !!     kvec = rk(:,k)-qq  ! k
00107 !!
00108 !! NOTE: dimension
00109 !!  nmtot = nctot+ nmmax-mnini+1
00110 !!  nqtot = ncc  + nqmax-nqini+1
00111 !! <rkvec,1:ngb   q-rkvec, 1:nmtot | q, 1:nqtot>
00112 !! ----------------------------------------
00113       subroutine get_zmelt2(exchange,
00114      &    kvec,irot,rkvec,ngc,ngb,  !MPB for       MPB_rkvec
00115      &    nmini,nmmax,ispm,nctot,   !middle for    phi_{q-rkvec}
00116      &    q,nqini,nqmax,ispq,ncc)   !end state for phi_q
00117 !! \parameter all inputs
00118 !! \parameter output=rmelt,clemt  matrix <MPB psi|psi>
00119       implicit none
00120       logical:: exchange
00121       integer:: invr,nxx,itp,irot,isp,kr,no,nmmax,ngc,ngb,nqmax,nbcut
00122       integer:: iatomp(natom),nmini,nqini,nctot,ncc
00123       real(8) :: symope(3,3),shtv(3),tr(3,natom),qk(3),det
00124      &  , quu(3),q(3), kvec(3),rkvec(3),wtt
00125       complex(8),allocatable :: zzzmel(:,:,:),zw (:,:)
00126       integer:: nmtot,nqtot
00127       real(8),allocatable :: drealzzzmel(:,:,:), dimagzzzmel(:,:,:) ,ppb(:)
00128       logical:: debug=.false.
00129       complex(8),parameter:: img=(0d0,1d0),tpi= 8d0*datan(1d0)
00130       complex(8):: expikt(natom)
00131       integer:: it,ia,kx,verbose,nstate,imdim(natom)
00132       logical:: oncew
00133       real(8),parameter::epsd=1d-6
00134       integer::ispq,ispm,iii,itps
00135 !TIME0_1001
00136       if(verbose()>80) debug=.true.
00137       if(debug) write(*,*) 'get_zmel in m_zmel: start'
00138       call getkeyvalue("GWinput","nbcutlow_sig",nbcut, default=0 )
00139       if(.not.done_genallcf_v3) call rx('m_zmel: not yet call genallcf_v3')
00140       if(.not.done_rdpp)        call rx('m_zmel: not yet call rdpp')
00141       if(.not.done_read_bzdata) call rx('m_zmel: not yet call read_bzdata')
00142
00143       if(init) then
00144          call dinv33(qbas,0,qbasinv,det)
00145          allocate( cphiq(nlmto,nband), cphim(nlmto,nband))
00146          init=.false.
00147       endif
00148
00149       if(sum(abs(q-q_bk))>epsd) then
00150          call readcphi(q, nlmto,ispq, quu, cphim )
00151          cphiq(1:nlmto,1:ntq) = cphim(1:nlmto,itq(1:ntq))
```

```
00152          q_bk=q
00153       endif
00154
00155       allocate( rmelt(ngb, nctot+nmmax, ncc+nqmax), ! nstate= nctot+nband
00156      &  cmelt(ngb, nctot+nmmax, ncc+nqmax))
00157       if(debug) write(*,*) 'get_zmel in m_zmel: 22222222'
00158
00159 !! qk = q-rk. rk is inside 1st BZ, not restricted to the irreducible BZ
00160       qk =  q - rkvec
00161       if(sum(abs(qk-qk_bk))>epsd) then
00162          call readcphi(qk, nlmto,ispm, quu, cphim)
00163          qk_bk=qk
00164       endif
00165 c      call getsrdpp2( nclass,nl,nxx)
00166 !! Rotate atomic positions invrot*R = R' + T
00167       invr   =  invg(irot)       !invrot (irot,invg,ngrp)
00168       tr     = tiat(:,:,invr)
00169       iatomp= miat(:,invr)
00170       symope= symgg(:,:,irot)
00171       shtv   = matmul(symope,shtvg(:,invr))
00172 !! ppb= <Phi(SLn,r) Phi(SL'n',r) B(S,i,Rr)>
00173 !! Note spin-dependence. Look for ixx==8 in hbas.m.F calling basnfp.F, which gives ppbrd.
00174       allocate( ppb(nlnmx*nlnmx*mdimx*nclass))
00175       ppb = ppbir(:,irot,ispq)
00176       if(debug) write(*,*) 'get_zmel in m_zmel: 3333333333'
00177
00178 !TIME1_1001 "init"
00179 !TIME0_1101
00180
00181 !! phase factors expikt(ia) is for exp(ik.T(R))
00182       do ia = 1,natom
00183         imdim(ia)  = sum(nblocha(iclass(1:ia-1)))+1
00184         expikt(ia) = exp(img *tpi* sum(kvec*tr(:,ia)) )
00185       end do
00186       nmtot  = nctot + nmmax -nmini+1      ! = phi_middle
00187       nqtot  = ncc   + nqmax -nqini+1      ! = phi_end
00188       allocate( zzzmel(nbloch,nmtot,nqtot))
00189       zzzmel=0d0
00190 !! MTO Core
00191       if(ncc>0.or.nctot>0) then
00192         call psicb_v3( nctot,ncc,nmmax,nqmax,iclass,expikt,
00193      i              cphim(1,nmini),  !middle phi
00194      i              cphiq(1,nqini),  !end phi
00195      i              ppb,!ppb,
00196      i              nlnmv,nlnmc,nblocha, !mdim,
00197      i              imdim,iatomp,
00198      i              mdimx,nlmto,nbloch,nlnmx,natom,nclass,
00199      i              icore,ncore,nl,nnc,
00200      o              zzzmel)
00201       endif
00202       if(debug) write(6,'("Goto psi2b_v3 nctot ncc nmmax nqmax=",4i4)') nctot,ncc,nmmax,nqmax
00203       if(debug) write(6,'("4444 zzzmelsum ",3i5,3d13.5)') nbloch,nmtot,nqtot,sum(abs(zzzmel)),sum(zzzmel)
00204 !! MTO Valence
00205       if(nmmax*nqmax>0) then      ! val num of nm  ! val num of nq
00206         call psi2b_v3( nctot,ncc, nmmax-nmini+1,   nqmax-nqini+1, iclass,expikt, !phase,
00207      i              cphim(1,nmini),
00208      i              cphiq(1,nqini),
00209      i              ppb,! ppb,
00210      i              nlnmv, nlnmc,nblocha, !mdim,
00211      i              imdim,iatomp,
00212      d              mdimx,nlmto,nbloch,nlnmx, natom,nclass,
00213      o              zzzmel)
00214       endif
00215       if(debug) write(6,'("5555 zzzmelsum ",3i5,3d13.5)') nbloch,nmtot,nqtot,sum(abs(zzzmel)),sum(zzzmel)
00216 !TIME1_1101 "psi2b_v3"
00217
00218 !TIME0_1201
00219 !! IPW
00220       allocate(drealzzzmel(nbloch,nmtot,nqtot),dimagzzzmel(nbloch,nmtot,nqtot))
00221       drealzzzmel=dreal(zzzmel)
00222       dimagzzzmel=dimag(zzzmel)
00223       deallocate(zzzmel)
00224 !     qk =  q - rkvec   !ncc+nqmax? nqtot?
00225       itps = nqini
00226       call drvmelp( q, nqmax-nqini+1, ! q     nt0 (in FBZ)
00227      i  qk,  nmmax-nmini+1,            ! q-rk  ntp0
00228      i  kvec,        ! k in IBZ for mixed product basis. rk = symope(kvec)
00229      i  ispq,ispm,ginv,
00230      i  ngc,ngcmx, ngpmx,nband,itq,
00231      i  symope, shtv, qbas, qbasinv,qibz,qbz,nqbz,nqibz,
00232      i  drealzzzmel, dimagzzzmel, nbloch, nctot,ncc,itps,
00233      o  rmelt,cmelt)
00234       if(debug) write(6,*) ' sxcf_fal1: end of drvmelp2 sum rmelt cmelt',sum(rmelt),sum(cmelt)
00235       deallocate(drealzzzmel,dimagzzzmel)
00236       if(verbose()>50) call timeshowx("5 after drvmelp")
00237       if(nbcut/=0.and.(.not.exchange)) then
00238          do it= nctot+1,nctot+min(nbcut,nmmax)
```

```
00239             rmelt(:, it,:) =0d0
00240             cmelt(:, it,:) =0d0
00241           enddo
00242         endif
00243 !TIME1_1201 "drvmelp"
00244
00245 !! NOTE:=======================================
00246 !! zmelt = rmelt(igb(rkvec), iocc(q), iunocc(q-rkvec)) + i* cmelt
00247 !! iunocc: band index at target  q.
00248 !! iocc:   band index at intermediate vector qk = q - rkvec
00249 !! igb: index of mixed product basis       at rkvec (or written as rk)
00250 !!   igb=1,ngb
00251 !!   ngb=nbloch+ngc  ngb: # of mixed product basis
00252 !!                   nbloch: # of product basis (within MTs)
00253 !!                   ngc: # of IPW for the Screened Coulomb interaction.
00254 !!                   igc is for given
00255 !! See readgeig in drvmelp2.
00256 !! ================================================
00257 c-------------------------------------------------------------------------
00258 c$$$!! smbasis
00259 c$$$!! smbasis ---need to fix this
00260 !!  Read pomatr
00261 c$$$      if(smbasis()) then  !this smbasis if block is from hsfp0.sc.m.F
00262 c$$$         write(6,*)' smooth mixed basis : augmented zmel'
00263 c$$$         ifpomat = iopen('POmat',0,-1,0) !oct2005
00264 c$$$         nkpo = nqibz+nq0i
00265 c$$$         nnmx=0
00266 c$$$         nomx=0
00267 c$$$         do ikpo=1,nkpo
00268 c$$$            read(ifpomat) q_r,nn_,no,iqx !readin reduction matrix pomat
00269 c$$$            if(nn_>nnmx) nnmx=nn_
00270 c$$$            if(no>nomx) nomx=no
00271 c$$$            allocate( pomat(nn_,no) )
00272 c$$$            read(ifpomat) pomat
00273 c$$$            deallocate(pomat)
00274 c$$$         enddo
00275 c$$$         isx = iclose("POmat")
00276 c$$$         ifpomat = iopen('POmat',0,-1,0) !oct2005
00277 c$$$         allocate( pomatr(nnmx,nomx,nkpo),qrr(3,nkpo),nor(nkpo),nnr(nkpo) )
00278 c$$$         do ikpo=1,nkpo
00279 c$$$            read(ifpomat) qrr(:,ikpo),nn_,no,iqx !readin reduction matrix pomat
00280 c$$$            nnr(ikpo)=nn_
00281 c$$$            nor(ikpo)=no
00282 c$$$            read(ifpomat) pomatr(1:nn_,1:no,ikpo)
00283 c$$$         enddo
00284 c$$$         isx = iclose("POmat")
00285 c$$$         write(6,*)"Read end of POmat ---"
00286 c$$$      endif
00287 c-----------------------------------
00288 c$$$               if(smbasis()) then !
00289 c$$$                  ntp0= nqmax
00290 c$$$                  nn= nnr(kx)
00291 c$$$                  no= nor(kx)
00292 c$$$                  allocate( pomat(nn,no) )
00293 c$$$                  pomat= pomatr(1:nn,1:no,kx)
00294 c$$$                  if( sum(abs(kvec-qrr(:,kx)))>1d-10 .and.kx <= nqibz ) then
00295 c$$$                     call rx( 'qibz/= qrr')
00296 c$$$                  endif
00297 c$$$                  if(no /= ngb.and.kx <= nqibz) then
00298 c$$$!!     A bit sloppy check only for kx<nqibz because qibze is not supplied...
00299 c$$$                     write(6,"(' q  ngb  ',3d13.5,3i5)")  kvec,ngb
00300 c$$$                     write(6,"(' q_r  nn no',3d13.5,3i5)") q_r,nn,no
00301 c$$$                     call rx( 'x0kf_v2h: POmat err no/=ngb')
00302 c$$$                  endif
00303 c$$$                  if(timemix) call timeshow("xxx2222 k-cycle")
00304 c$$$                  ngb = nn      ! Renew ngb !!!
00305 c$$$                  allocate ( zmel(nn, nctot+nmmax, ntp0) )
00306 c$$$                  call matm( pomat, dcmplx(rmelt,cmelt), zmel,
00307 c$$$     &                 nn, no, (nctot+nmmax)*ntp0 )
00308 c$$$                  deallocate(rmelt, cmelt)
00309 c$$$                  allocate( rmelt(ngb, nctot+nmmax, ntp0), !ngb is reduced.
00310 c$$$     &                 cmelt(ngb, nctot+nmmax, ntp0) )
00311 c$$$                  rmelt = dreal(zmel)
00312 c$$$                  cmelt = dimag(zmel)
00313 c$$$                  deallocate(zmel,pomat)
00314 c$$$               else
00315 c$$$                  nn=ngb
00316 c$$$                  no=ngb
00317 c$$$               endif
00318
00319 c      if( oncew() ) then
00320 c         write(6,"('ngb nn no=',3i6)") ngb,nn,no
00321 c      endif
00322 c               if(timemix) call timeshow("22222 k-cycle")
00323       if(allocated(zzzmel))deallocate(zzzmel) !rmel,cmel)
00324       if(debug) write(6,*) ' sxcf: goto wtt'
00325       if(debug) write(6,"('sum of rmelt cmelt=',4d23.16)")sum(rmelt),sum(cmelt)
```

```
00326 !! === End of zmelt ; we now have matrix element zmelt= rmelt + img* cmelt ===
00327 !TIME0_1301
00328
00329 !! Multipled by ppovlz and reformat
00330       if(exchange) then
00331          if(debug) write(*,*) 'exchange mode 0000 ngb nmtot nqtot',ngb,nmtot,nqtot
00332          allocate( zmel(ngb, nmtot, nqtot))
00333          zmel = dcmplx(rmelt,cmelt)
00334          if(debug) write(*,*) 'exchange mode 1111'
00335          deallocate(rmelt,cmelt)
00336          if(debug) then
00337             do it = 1,nmtot
00338                write(6,"('wwwwwsc ',i5,2f10.4)") it,sum(abs(zmel(:,it,1)))
00339             enddo
00340             write(*,*) 'eeeeeeeeeeeee end of wwwwsc',nctot,nmmax
00341             write(6,*)'sumcheck ppovlz=',sum(abs(ppovlz(:,:)))
00342          endif
00343 !! OUTPUT zmeltt for exchange
00344          allocate(zmeltt(nmtot,nqtot,ngb))
00345
00346          if(verbose()>39) then
00347             write(*,*)'info: USE GEMM FOR SUM (zmeltt=zmel*ppovlz) in sxcf_fal2.sc.F'
00348             write(*,*)'zgemmsize',nqtot*nmtot,ngb,ngb
00349             write(*,*)'size ,zmel',size(zmel,dim=1),size(zmel,dim=2),size(zmel,dim=3)
00350             write(*,*)'size ,ppovlz',size(ppovlz,dim=1),size(ppovlz,dim=2)
00351             write(*,*)'size ,zmeltt',size(zmeltt,dim=1),size(zmeltt,dim=2),size(zmeltt,dim=3)
00352          endif
00353          call flush(6)
00354          call zgemm('T','N',nqtot*nmtot,ngb,ngb,(1d0,0d0),
00355      .      zmel,ngb,ppovlz,ngb,(0d0,0d0),zmeltt,nqtot*nmtot )
00356          deallocate(zmel)
00357       else
00358 !! Correlation case. Get zmel
00359          if(debug) write(*,*) 'correlation mode 0000'
00360 c         nstate = nctot + nmmax ! = nstate for the case of correlation
00361          allocate(zmeltt(ngb, nmtot, nqtot))
00362          zmeltt= dcmplx(rmelt,-cmelt) !zmeltt= <itp|it,ib>
00363          deallocate(rmelt,cmelt)
00364 !! zmel(igb,it*itp) = C(ppovlz)*N(zmeltt(:,it*itp))
00365 !!  C means Hermitian conjugate, N means normal
00366 !! http://www.netlib.org/lapack/lapack-3.1.1/html/zgemm.f.html
00367 !! OUTPUT
00368          allocate( zmel(ngb, nmtot, nqtot) )
00369
00370          if(debug) write(6,'("4 zzzppp222aaa ",3d13.5)') sum(abs(zmeltt)),sum(zmeltt)
00371          call zgemm('C','N',ngb, nmtot*nqtot,ngb,(1d0,0d0),
00372      .      ppovlz, ngb, zmeltt,ngb, (0d0,0d0),zmel,ngb)
00373          deallocate(zmeltt)
00374          if(debug) write(*,*)'zz000 nmtot,ngb,nstate ',nmtot,ngb,nqtot
00375          if(debug) write(*,*)'zz000 sumchk zmel ',sum(abs(zmel(1:ngb,1:nmtot,1:nqtot)))
00376          if(debug) write(*,*) 'correlation mode end'
00377 !TIME1_1301 "matmul_zmelp_povlz"
00378       endif
00379       end subroutine get_zmelt2
00380       end module m_zmel
00381
00382       subroutine timeshowx(info)
00383       character*(*) :: info
00384       write(*,'(a,$)')info
00385       call cputid(0)
00386       end
00387
```

4.17 gwsrc/mkjp.F File Reference

Functions/Subroutines

- subroutine vcoulq_4 (q, nbloch, ngc,nbas, lx, lxx, nx, nxx,alat, qlat, vol, ngvecc,strx, rojp, rojb, sgbb, sgpb, fouvb,nblochpmx, bas, rmax,eee, aa, bb, nr, nrx, rkpr, rkmr, rofi,
- subroutine mkjp_4 (q, ngc, ngvecc, alat, qlat, lxx, lx, nxx, nx, bas, a, b, rmax, nr, nrx, rprodx, eee, rofi, rkpr, rkmr, rojp, sgpb, fouvb)
- real(8) function fac2m (i)
- subroutine genjh (eee, nr, a, b, lx, nrx, lxx, rofi, rkpr, rkmr)
- subroutine mkjb_4 (lxx, lx, nxx, nx, a, b, nr, nrx, rprodx, rofi, rkpr, rkmr, rojb, sgbb)
- subroutine sigint_4 (rkp, rkm, kmx, a, b, nr, phi1, phi2, rofi, sig)
- subroutine intn_smpxxx (g1, g2, int, a, b, rofi, nr, lr0)

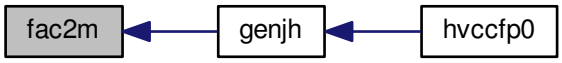
- subroutine sigintan1 (absqg, lx, rofi, nr, a1int)
- subroutine sigintpp (absqg1, absqg2, lx, rmax, sig)

4.17.1 Function/Subroutine Documentation

4.17.1.1 real(8) function fac2m (*i*)

Definition at line 627 of file mkjp.F.

Here is the caller graph for this function:



4.17.1.2 subroutine genjh (real(8) *eee,* integer(4) *nr,* real(8) *a,* real(8) *b,* integer(4) *lx,* integer(4) *nrx,* integer(4) *lxx,* real(8), dimension(nrx) *rofi,* real(8), dimension(nrx,0:lxx) *rkpr,* real(8), dimension(nrx,0:lxx) *rkmr*)

Definition at line 641 of file mkjp.F.

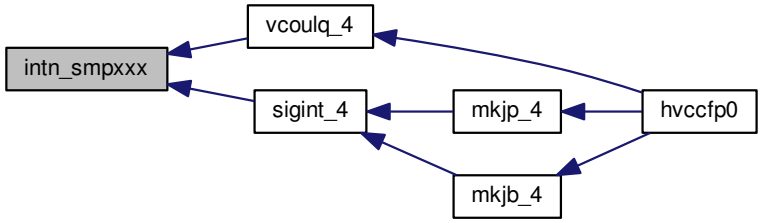
Here is the call graph for this function:

Here is the caller graph for this function:

4.17.1.3 subroutine intn_smpxxx (double precision, dimension(nr) *g1,* double precision, dimension(nr) *g2,* double precision, dimension(nr) *int,* double precision *a,* double precision *b,* double precision, dimension(nr) *rofi,* integer *nr,* integer *lr0*)

Definition at line 780 of file mkjp.F.

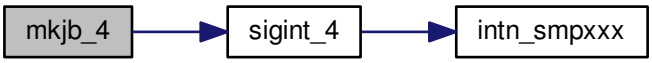
Here is the caller graph for this function:



4.17.1.4 subroutine mkjb_4 (integer(4) *lxx,* integer(4) *lx,* integer(4) *nxx,* integer(4), dimension(0:lxx) *nx,* real(8) *a,* real(8) *b,* integer(4) *nr,* integer(4) *nrx,* real(8), dimension(nrx,nxx,0:lxx) *rprodx,* real(8), dimension(nrx) *rofi,* real(8), dimension(nrx,0:lxx) *rkpr,* real(8), dimension(nrx,0:lxx) *rkmr,* real(8), dimension(nxx, 0:lxx) *rojb,* real(8), dimension(nxx, nxx, 0:lxx) *sgbb*)

Definition at line 679 of file mkjp.F.

Here is the call graph for this function:

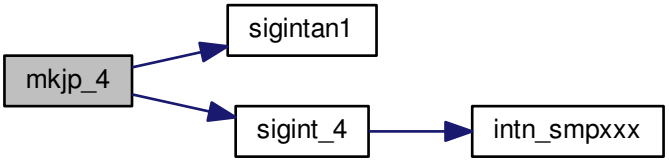


Here is the caller graph for this function:

4.17.1.5 subroutine mkjp_4 (real(8), dimension(3) *q,* integer(4) *ngc,* integer(4), dimension(3,ngc) *ngvecc,* real(8) *alat,* real(8), dimension(3,3) *qlat,* integer(4) *lxx,* integer(4) *lx,* integer(4) *nxx,* integer(4), dimension(0:lxx) *nx,* real(8), dimension(3) *bas,* real(8) *a,* real(8) *b,* real(8) *rmax,* integer(4) *nr,* integer(4) *nrx,* real(8), dimension(nrx,nxx,0:lxx) *rprodx,* real(8) *eee,* real(8), dimension(nrx) *rofi,* real(8), dimension(nrx,0:lxx) *rkpr,* real(8), dimension(nrx,0:lxx) *rkmr,* complex(8), dimension(ngc, (lxx+1)∗∗2) *rojp,* complex(8), dimension(ngc, nxx, (lxx+1)∗∗2) *sgpb,* complex(8), dimension(ngc, nxx, (lxx+1)∗∗2) *fouvb*)

Definition at line 429 of file mkjp.F.

Here is the call graph for this function:

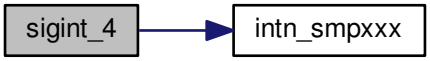


Here is the caller graph for this function:

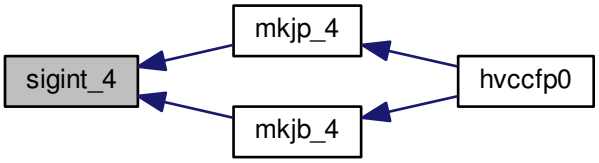
4.17.1.6 subroutine sigint_4 (real(8), dimension(nr) *rkp,* real(8), dimension(nr) *rkm,* integer(4) *kmx,* real(8) *a,* real(8) *b,* integer(4) *nr,* real(8), dimension(nr) *phi1,* real(8), dimension(nr) *phi2,* real(8), dimension(nr) *rofi,* real(8) *sig*)

Definition at line 760 of file mkjp.F.

Here is the call graph for this function:



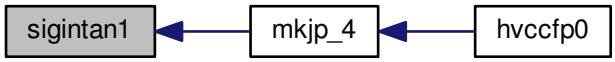
Here is the caller graph for this function:



4.17.1.7 subroutine sigintan1 (real(8) *absqg,* integer(4) *lx,* real(8), dimension(nr) *rofi,* integer(4) *nr,* real(8), dimension(nr,0:lx) *a1int*)

Definition at line 820 of file mkjp.F.

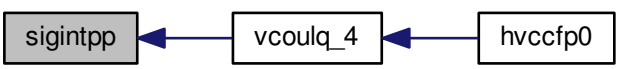
Here is the caller graph for this function:



4.17.1.8 subroutine sigintpp (real(8) *absqg1,* real(8) *absqg2,* integer(4) *lx,* real(8) *rmax,* real(8), dimension(0:lx) *sig*)

Definition at line 862 of file mkjp.F.

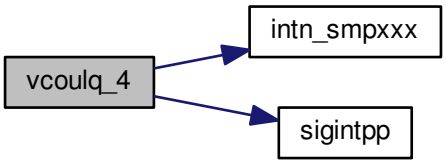
Here is the caller graph for this function:



4.17.1.9 subroutine vcoulq_4 (real(8), dimension(3) *q,* integer(4) *nbloch,* integer(4) *ngc,* integer(4) *nbas,* integer(4), dimension(nbas) *lx,* integer(4) *lxx,* integer(4), dimension(0:lxx,nbas) *nx,* integer(4) *nxx,* real(8) *alat,* real(8), dimension(3,3) *qlat,* real(8) *vol,* integer(4), dimension(3,ngc) *ngvecc,* complex(8), dimension((lxx+1)**2, nbas, (lxx+1)**2,nbas) *strx,* complex(8), dimension(ngc, (lxx+1)**2, nbas) *rojp,* real(8), dimension(nxx, 0:lxx, nbas) *rojb,* real(8), dimension(nxx, nxx, 0:lxx, nbas) *sgbb,* complex(8), dimension(ngc, nxx, (lxx+1)**2, nbas) *sgpb,* complex(8), dimension(ngc, nxx, (lxx+1)**2, nbas) *fouvb,* integer(4) *nblochpmx,* real(8), dimension(3,nbas) *bas,* real(8), dimension(nbas) *rmax,* real(8) *eee,* real(8), dimension(nbas) *aa,* real(8), dimension(nbas) *bb,* integer(4), dimension(nbas) *nr,* integer(4) *nrx,* real(8), dimension(nrx,0:lxx,nbas) *rkpr,* real(8), dimension(nrx,0:lxx,nbas) *rkmr,* real(8), dimension(nrx,nbas) *rofi*)

Definition at line 1 of file mkjp.F.

Here is the call graph for this function:



Here is the caller graph for this function:

4.18 mkjp.F

```
00001       subroutine vcoulq_4(q,nbloch, ngc,
00002      &                  nbas, lx,lxx, nx,nxx,
00003      &                  alat, qlat, vol, ngvecc,
00004      &          strx,rojp,rojb, sgbb,sgpb, fouvb,   !sgpp,fouvp,
00005      &          nblochpmx,bas,rmax,
00006      &    eee, aa,bb,nr,nrx,rkpr,rkmr,rofi,
00007 !         These inputs are to generate sgpp on the fly.
00008      o          vcoul)
00009 Co Coulmb matrix for each q. ------------------------------------------------
00010 Ci strx:  Structure factors
00011 Ci nlx corresponds to (lx+1)**2 . lx corresponds to 2*lmxax.
00012 Ci rho-type integral
00013 Ci  ngvecc     : q+G vector
00014 Ci  rojp rojb  : rho-type integral
00015 ci  sigma-type onsite integral
00016 ci  Fourier
00017 Ci  nx(l,ibas) : max number of radial function index for each l and ibas.
00018 Ci               Note that the definition is a bit different from nx in basnfp.
00019 ci  nxx        : max number of nx among all l and ibas.
00020 ci  lx(nbas)   : max number of l for each ibas.
00021 ci  lxx        :
00022 ci
```

```
00023 ci  vol : cell vol
00024 c
00025 Co Vcoul
00026 cr vcoul is in a.u. You have to multiply e~2=2 if you want to it in Ry,
00027 cr    vcoul = 2d0*vcoul !  in Ry unit.
00028 c-------------------------------------------------------------------------
00029 c  rojp = <j_aL(r) | P(q+G)_aL > where
00030 c         |P(q+G)_aL> : the projection of exp(i (q+G) r) to aL channnel.
00031 c         |j_aL>      : \def r^l/(2l+1)!! Y_L.  The spherical bessel functions near r=0.  Energy-dependence
       is omitted.
00032 c
00033       implicit none
00034       integer(4) :: nbloch, nblochpmx, nbas,
00035      &              lxx,lx(nbas), nxx, nx(0:lxx,nbas)
00036       real(8)    :: egtpi,vol,q(3),fpi
00037
00038 ci structure con
00039       complex(8) :: strx((lxx+1)**2, nbas, (lxx+1)**2,nbas)
00040 ci |q+G|**2
00041       integer(4) :: ngc, ngvecc(3,ngc)
00042       real(8)    :: qlat(3,3),alat,absqg2(ngc),qg(3)
00043
00044 ci rho-type onsite integral
00045       complex(8) ::   rojp(ngc, (lxx+1)**2, nbas)
00046       real(8)    ::   rojb(nxx, 0:lxx, nbas)
00047 ci sigma-type onsite integral
00048       real(8)    :: sgbb(nxx,  nxx,  0:lxx,      nbas)
00049       complex(8) :: sgpb(ngc,  nxx,  (lxx+1)**2, nbas)
00050 c     &              ,sgpp(ngc,  ngc,  (lxx+1)**2, nbas)
00051 ci Fourier
00052       complex(8) ::
00053      &             fouvb(ngc,  nxx, (lxx+1)**2, nbas)
00054 Co
00055      &             ,vcoul(nblochpmx, nblochpmx)
00056 c     &             ,fouvp(ngc,  ngc, (lxx+1)**2, nbas)
00057
00058 cinternals
00059       integer(4) :: ibl1, ibl2,ig1,ig2,ibas,ibas1,ibas2,
00060      &              l,m,n, n1,l1,m1,lm1,n2,l2,m2,lm2,ipl1,ipl2
00061       integer(4) :: ibasbl(nbloch), nbl(nbloch), lbl(nbloch),
00062      &              mbl(nbloch), lmbl(nbloch)
00063       real(8) :: pi, fpivol,tpiba
00064       complex(8) :: rojpstrx((lxx+1)**2,nbas)
00065
00066 c check
00067       complex(8),allocatable :: hh(:,:),oo(:,:),zz(:,:)
00068       real(8),allocatable    :: eb(:)
00069
00070       complex(8),allocatable :: matp(:),matp2(:)
00071       complex(8) :: xxx
00072       integer(4) :: nblochngc,nev,nmx,ix
00073       logical :: ptest=.false. ! See ptest in basnfp.f
00074
00075 c-------------------
00076       real(8),   allocatable :: cy(:),yl(:)
00077       complex(8),allocatable :: pjyl_(:,:),phase(:,:)
00078       complex(8) :: img=(0d0,1d0)
00079       real(8):: bas(3,nbas),r2s,rmax(nbas)
00080       integer(4):: lm
00081 #ifdef COMMONLL
00082       integer(4)::ll(51**2)
00083       common/llblock/ll
00084 #else
00085       integer(4) :: ll
00086       external ll
00087 #endif
00088       real(8)::  fkk(0:lxx),fkj(0:lxx),fjk(0:lxx),fjj(0:lxx),sigx(0:lxx),radsig(0:lxx)
00089       complex(8):: fouvp_ig1_ig2, fouvp_ig2_ig1, sgpp_ig1_ig2
00090
00091       integer(4):: nrx,nr(nbas),ir,ig
00092       real(8):: eee , int1x(nrx),int2x(nrx),phi(0:lxx),psi(0:lxx)
00093      &  ,aa(nbas),bb(nbas),rkpr(nrx,0:lxx,nbas),rkmr(nrx,0:lxx,nbas)
00094      &  ,rofi(nrx,nbas)
00095       real(8), allocatable:: ajr(:,:,:,:), a1(:,:,:)
00096       logical :: debug=.false.
00097 c-----------------------------------------------------------------
00098       write(6,'(" vcoulq_4: nblochpmx  nbloch ngc=",3i6)') nblochpmx,nbloch,ngc
00099 c     print *, ' sum fouvp=',sum(fouvp(:,:,:,1))
00100 c     print *, ' sum fouvb=',sum(fouvb(:,:,:,1))
00101       pi    = 4d0*datan(1d0)
00102       fpi    = 4*pi
00103       fpivol = 4*pi*vol
00104
00105 c---for sgpp fouvp
00106       allocate(  !ajr(1:nr,0:lx,ngc),a1(1:nr,0:lx,ngc),rkpr(nr,0:lx),rkmr(nr,0:lx),
00107      &  pjyl_((lxx+1)**2,ngc),phase(ngc,nbas) )
00108       allocate(cy((lxx+1)**2),yl((lxx+1)**2))
```

```
00109       call sylmnc(cy,lxx)
00110
00111 c=======================================================
00112       vcoul = 0d0
00113 c-gvec
00114       tpiba = 2*pi/alat
00115       do ig1 = 1,ngc
00116         qg(1:3) = tpiba * (q(1:3)+ matmul(qlat, ngvecc(1:3,ig1)))
00117         absqg2(ig1)  = sum(qg(1:3)**2)
00118 c---for spgg fourvp ----------
00119         do ibas=1,nbas
00120           phase(ig1,ibas) = exp( img*sum(qg(1:3)*bas(1:3,ibas))*alat  )
00121         enddo
00122         call sylm(qg/sqrt(absqg2(ig1)),yl,lxx,r2s) !spherical factor Y( q+G )
00123         do lm =1,(lxx+1)**2
00124           l = ll(lm)
00125           pjyl_(lm,ig1) = fpi*img**l *cy(lm)*yl(lm)  * sqrt(absqg2(ig1))**l  !*phase
00126           ! <jlyl | exp i q+G r> projection of exp(i q+G r) to jl yl  on MT
00127         enddo
00128 c--------------
00129       enddo
00130 c
00131
00132
00133 c-- index (mx,nx,lx,ibas) order.
00134       ibl1 = 0
00135       do ibas= 1, nbas
00136         do l   = 0, lx(ibas)
00137 c         write(6,'(" l ibas nx =",3i5)') l,nx(l,ibas),ibas
00138           do n   = 1, nx(l,ibas)
00139             do m   = -l, l
00140               ibl1  = ibl1 + 1
00141               ibasbl(ibl1) = ibas
00142               nbl(ibl1) = n
00143               lbl(ibl1) = l
00144               mbl(ibl1) = m
00145               lmbl(ibl1) = l**2 + l+1 +m
00146 c         write(6,*)ibl1,n,l,m,lmbl(ibl1)
00147             enddo
00148           enddo
00149         enddo
00150       enddo
00151       if(ibl1/= nbloch) then
00152         write(6,*)' ibl1 nbloch',ibl1, nbloch
00153 Cstop2rx 2013.08.09 kino        stop ' vcoulq: error ibl1/= nbloch'
00154         call rx( ' vcoulq: error ibl1/= nbloch')
00155       endif
00156
00157
00158 c-- <B|v|B> block
00159 c      write(6,*)' vcoulq: bvb block xxx rojbsum='
00160 c      write(6,*) sum(rojb(:,:,1))
00161 c      write(6,*) sum(rojb(:,:,2))
00162 c      write(6,*) sum(rojb(:,:,3))
00163 c      write(6,*) sum(rojb(:,:,4))
00164 c      write(6,*)' vcoulq: bvb block xxx sgbbbsum='
00165 c      write(6,*) sum(sgbb(:,:,:,1))
00166 c      write(6,*) sum(sgbb(:,:,:,2))
00167 c      write(6,*) sum(sgbb(:,:,:,3))
00168 c      write(6,*) sum(sgbb(:,:,:,4))
00169       do ibl1= 1, nbloch
00170         ibas1= ibasbl(ibl1)
00171         n1   = nbl(ibl1)
00172         l1   = lbl(ibl1)
00173         m1   = mbl(ibl1)
00174         lm1  = lmbl(ibl1)
00175         do ibl2= 1, ibl1
00176           ibas2= ibasbl(ibl2)
00177           n2   = nbl(ibl2)
00178           l2   = lbl(ibl2)
00179           m2   = mbl(ibl2)
00180           lm2  = lmbl(ibl2)
00181           vcoul(ibl1,ibl2) =
00182      &     rojb(n1, l1, ibas1) *strx(lm1,ibas1,lm2,ibas2)
00183      &    *rojb(n2, l2, ibas2)
00184           if(ibas1==ibas2 .and. lm1==lm2) then
00185             vcoul(ibl1,ibl2) = vcoul(ibl1,ibl2) + sgbb(n1,n2,l1, ibas1)
00186             ! sigma-type contribution. onsite coulomb
00187           endif
00188         enddo
00189       enddo
00190
00191 ccccccccccccccccccccccccc
00192 c      goto 1112
00193 ccccccccccccccccccccccccc
00194
00195 c <P_G|v|B>
```

```
00196       if(debug) write(6,*)' vcoulq_4: pgvb block 1111'
00197       do ibl2= 1, nbloch
00198         ibas2= ibasbl(ibl2)
00199         n2   = nbl(ibl2)
00200         l2   = lbl(ibl2)
00201         m2   = mbl(ibl2)
00202         lm2  = lmbl(ibl2)
00203         do ig1 = 1,ngc
00204           ipl1 = nbloch + ig1
00205           vcoul(ipl1,ibl2) = fouvb(ig1,  n2, lm2, ibas2)
00206
00207           do ibas1= 1, nbas
00208             do lm1  = 1, (lx(ibas1)+1)**2
00209               vcoul(ipl1,ibl2) = vcoul(ipl1,ibl2) -
00210      &      dconjg(rojp(ig1, lm1, ibas1)) *strx(lm1,ibas1,lm2,ibas2)
00211      &      *rojb(n2, l2, ibas2)
00212               if(ibas1==ibas2 .and.lm1==lm2) then
00213                 vcoul(ipl1,ibl2) = vcoul(ipl1,ibl2) -
00214      &        sgpb(ig1, n2, lm2, ibas2)
00215               endif
00216             enddo
00217           enddo
00218         enddo
00219       enddo
00220
00221       if(debug) write(6,*)' vcoulq_4: ajr allocate'
00222 C... prepare funciton ajr and a1.
00223 C... ajr:spherical bessel, a1: integral of (sperical bseel)*(rkp rkm)
00224 c------------------
00225       allocate( ajr(nrx,0:lxx,nbas,ngc), a1(nrx,0:lxx,nbas) )
00226       if(debug) write(6,*)' vcoulq_4: end ajr allocate'
00227       do ig1 = 1,ngc
00228         do ibas= 1,nbas
00229           if(debug) write(6,"('ccc: ',10i15)")ig1,ibas
00230           do ir = 1,nr(ibas)
00231             call bessl(absqg2(ig1)*rofi(ir,ibas)**2,lxx,phi,psi)
00232             do l  = 0,lx(ibas)
00233
00234               if(debug.and.ig==162.and.ibas==8) then
00235                 write(6,"('ccc: ',10i15)")ig1,ibas,ir,l
00236                 write(6,*)"ccc:", phi(l)
00237                 write(6,*)"ccc:", rofi(ir,ibas)
00238               endif
00239
00240               ajr(ir,l,ibas,ig1) = phi(l)* rofi(ir,ibas) **(l +1 )
00241               ! ajr = j_l(sqrt(e) r) * r / (sqrt(e))**l
00242             enddo
00243           enddo
00244         enddo
00245       enddo
00246 c------------------
00247
00248 c <P_G|v|P_G>
00249       if(debug) write(6,*)' vcoulq_4: pgvpg block'
00250       do ig1 = 1,ngc
00251         ipl1 = nbloch + ig1
00252         rojpstrx = 0d0
00253         do ibas1= 1, nbas
00254           do lm1  = 1, (lx(ibas1)+1)**2
00255             do ibas2= 1, nbas
00256               do lm2  = 1, (lx(ibas2)+1)**2
00257                 rojpstrx(lm2, ibas2) = rojpstrx(lm2, ibas2)+
00258      &      dconjg(rojp(ig1, lm1, ibas1)) *strx(lm1,ibas1,lm2,ibas2)
00259               enddo
00260             enddo
00261           enddo
00262         enddo
00263
00264 c----------------------
00265         do ibas=1,nbas
00266           do l = 0,lx(ibas)
00267             call intn_smpxxx( rkpr(1,l,ibas), ajr(1,l,ibas,ig1),int1x
00268      &         ,aa(ibas),bb(ibas),rofi(1,ibas),nr(ibas),0)
00269             call intn_smpxxx( rkmr(1,l,ibas), ajr(1,l,ibas,ig1),int2x
00270      &         ,aa(ibas),bb(ibas),rofi(1,ibas),nr(ibas),0)
00271             a1(1,          l,ibas) = 0d0
00272             a1(2:nr(ibas),l,ibas) =
00273      &             rkmr(2:nr(ibas),l,ibas) *( int1x(1)-int1x(2:nr(ibas)) )
00274      &           + rkpr(2:nr(ibas),l,ibas) *  int2x(2:nr(ibas))
00275           enddo
00276         enddo
00277 c---------------------
00278
00279         do ig2 = 1,ig1
00280           ipl2 = nbloch + ig2
00281           if(ig1==ig2) vcoul(ipl1,ipl2) = fpivol/(absqg2(ig1) -eee) !eee is negative
00282           do ibas2= 1, nbas
```

```
00283 c... for fouvp and sgpp -------
00284             call wronkj( absqg2(ig1), absqg2(ig2), rmax(ibas2),lx(ibas2),
00285      o                  fkk,fkj,fjk,fjj)
00286
00287             if(eee==0d0) then
00288               call sigintpp( absqg2(ig1)**.5, absqg2(ig2)**.5, lx(ibas2), rmax(ibas2),
00289      o            sigx)
00290             else
00291               do l = 0,lx(ibas2)
00292                 call gintxx(a1(1,l,ibas2), ajr(1,l,ibas2,ig2)
00293      &                     ,aa(ibas2),bb(ibas2),nr(ibas2), sigx(l))
00294               enddo
00295             endif
00296             do l = 0,lx(ibas2)
00297               radsig(l) = fpi/(2*l+1) * sigx(l)
00298             enddo
00299
00300 c-----------------------------
00301             do lm2  = 1, (lx(ibas2)+1)**2
00302               l= ll(lm2)
00303 c...fouvp sgpp-----------
00304               fouvp_ig1_ig2 = fpi/(absqg2(ig1)-eee) *dconjg(pjyl_(lm2,ig1)*phase(ig1,ibas2))
00305      &      * (-fjj(l)) * pjyl_(lm2,ig2)*phase(ig2,ibas2)
00306               fouvp_ig2_ig1 = fpi/(absqg2(ig2)-eee) *dconjg(pjyl_(lm2,ig2)*phase(ig2,ibas2))
00307      &      * (-fjj(l)) * pjyl_(lm2,ig1)*phase(ig1,ibas2)
00308               sgpp_ig1_ig2  = dconjg(pjyl_(lm2,ig1)*phase(ig1,ibas2))*radsig(l)
00309      &                      * pjyl_(lm2,ig2)*phase(ig2,ibas2)
00310 c-------------------------
00311               vcoul(ipl1,ipl2) = vcoul(ipl1,ipl2)
00312      &       +  rojpstrx(lm2,ibas2)*rojp(ig2, lm2, ibas2)
00313 c     &       -  dconjg( fouvp(ig2,  ig1, lm2, ibas2)) !BugFix Mar5-01 It was dcmplx.
00314 c     &       -          fouvp(ig1,  ig2, lm2, ibas2)
00315 c     &       +  sgpp(ig1, ig2, lm2, ibas2)
00316      &       -  dconjg( fouvp_ig2_ig1 )
00317      &       -          fouvp_ig1_ig2
00318      &       +  sgpp_ig1_ig2
00319             enddo
00320           enddo
00321         enddo
00322       enddo
00323 ccccccccccccccccccccccccccccc
00324 c 1112 continue
00325 ccccccccccccccccccccccccccccc
00326
00327
00328 c-- Right-Upper part of vcoul.
00329       if(debug) write(6,*)' vcoulq_4: right-upper'
00330       do ipl1=1, nbloch+ngc
00331         do ipl2=1, ipl1-1
00332           vcoul(ipl2,ipl1) = dconjg(vcoul(ipl1,ipl2))
00333         enddo
00334       enddo
00335
00336 ccccccccccccccccccccccccccccc
00337 c test.xxxxxxxxxx
00338 c$$$      do ibl2= 1, nbloch
00339 c$$$        ibas2= ibasbl(ibl2)
00340 c$$$        n2   = nbl (ibl2)
00341 c$$$        l2   = lbl (ibl2)
00342 c$$$        m2   = mbl (ibl2)
00343 c$$$        lm2  = lmbl(ibl2)
00344 c$$$        if(l2==1.and.ibas2>2) then
00345 c$$$          vcoul(nbloch+1:nbloch+ngc, ibl2) = 0d0
00346 c$$$          vcoul(ibl2, nbloch+1:nbloch+ngc) = 0d0
00347 c$$$        endif
00348 c$$$      enddo
00349 ccccccccccccccccccccccccc
00350
00351 c vcoul is in a.u. You have to multiply e~2=2 if you want to it in Ry,
00352 c     vcoul = 2d0*vcoul !  in Ry unit.
00353 c
00354
00355 c check write
00356       do ix = 1,nbloch+ngc,20
00357         write(6,"(' Diagonal Vcoul =',i5,2d18.10)") ix,vcoul(ix,ix)
00358       enddo
00359       if( allocated(yl)   ) deallocate(yl)
00360       if( allocated(cy)   ) deallocate(cy)
00361       if( allocated(phase)) deallocate(phase)
00362       if( allocated(pjyl_)) deallocate(pjyl_)
00363       if(.not.ptest) return
00364
00365
00366
00367 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00368 c! Below ia a plane-wave test.
00369 c--- check! Coulomb by plane wave expansion.
```

```
00370       write(6,*) ' --- plane wave Coulomb matrix check 1---- '
00371       write(197,*) ' --- off diagonal ---- '
00372       nblochngc = nbloch+ngc
00373       allocate(matp(nblochngc),matp2(nblochngc))
00374       do ig1 = 1,ngc
00375         matp = 0d0
00376         do ibl2= 1, nbloch
00377           ibas2= ibasbl(ibl2)
00378           n2   = nbl(ibl2)
00379           l2   = lbl(ibl2)
00380           m2   = mbl(ibl2)
00381           lm2  = lmbl(ibl2)
00382           matp(ibl2) = fouvb(ig1, n2, lm2, ibas2)*absqg2(ig1)/fpi
00383         enddo
00384         matp(nbloch+ig1) = 1d0
00385         ig2=ig1
00386 c      do ig2 = 1,ngc !off diagnal
00387         matp2 = 0d0
00388         do ibl2= 1, nbloch
00389           ibas2= ibasbl(ibl2)
00390           n2   = nbl(ibl2)
00391           l2   = lbl(ibl2)
00392           m2   = mbl(ibl2)
00393           lm2  = lmbl(ibl2)
00394           matp2(ibl2) = fouvb(ig2, n2, lm2, ibas2)*absqg2(ig2)/fpi
00395         enddo
00396         matp2(nbloch+ig2) = 1d0
00397         xxx= sum(
00398      &    matmul(matp(1:nblochngc),vcoul(1:nblochngc,1:nblochngc))
00399      &            *dconjg(matp2(1:nblochngc))  )
00400         if(ig1/=ig2) then  !off diagnal
00401           if(abs(xxx)>1d-1 ) then
00402             write(197,'(2i5, 2d13.6)') ig1,ig2, xxx
00403             write(197,'("    matpp ", 2d13.6)')
00404      &      vcoul(nbloch+ig1,nbloch+ig2)
00405             write(197,*)
00406           endif
00407         else
00408           write(196,'(2i5," exact=",3d13.6,"q ngsum=",3f8.4,i5)')
00409      &       ig1,ig2,fpi*vol/absqg2(ig1)
00410      &    , fpi*vol/absqg2(ig2),absqg2(ig1), q(1:3)
00411      &    , sum(ngvecc(1:3,ig1)**2)
00412           write(196,'("           cal  =", 2d13.6)') xxx
00413           write(196,'("           vcoud=", 2d13.6)')
00414      &       vcoul(nbloch+ig1,nbloch+ig2)
00415           write(196,*)
00416         endif
00417 c      enddo !off diagnal
00418       enddo
00419 c
00420       deallocate(matp,matp2)
00421 c      stop ' *** ptest end *** See fort.196 and 197'
00422 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00423       end
00424
00425
00426
00427
00428 c=======================================================================
00429       subroutine mkjp_4( q,ngc,ngvecc, alat, qlat, lxx,lx,nxx,nx,
00430      i                   bas, a,b,rmax,nr,nrx,rprodx,
00431      i    eee,rofi,rkpr,rkmr,
00432      o        rojp,sgpb,fouvb)
00433 C- Make integrals in each MT. and the Fourier matrix.
00434 Cr the integrals rojp, fouvb,fouvp
00435 Cr are for  J_L(r)= j_l(sqrt(e) r)/sqrt(e)**l Y_L,
00436 Cr which behaves as r^l/(2l+1)!! near r=0.
00437 Cr
00438 Cr oniste integral is based on
00439 Cr 1/|r-r'| = \sum 4 pi /(2k+1) \frac{r_<^k }{ r_>^{k+1} } Y_L(r) Y_L(r')
00440 Cr See PRB34 5512(1986) for sigma type integral
00441 Cr
00442       implicit none
00443       integer(4) :: ngc,ngvecc(3,ngc), lxx, lx, nxx,nx(0:lxx),nr,nrx
00444       real(8)    :: q(3),bas(3), rprodx(nrx,nxx,0:lxx),a,b,rmax,alat,
00445      &              qlat(3,3)
00446 ci rho-type onsite integral
00447       complex(8) :: rojp(ngc, (lxx+1)**2)
00448 ci sigma-type onsite integral
00449       complex(8) :: sgpb(ngc,  nxx,  (lxx+1)**2)
00450 c     &               sgpp(ngc,  ngc,  (lxx+1)**2)
00451       real(8),allocatable::cy(:),yl(:)
00452 ci Fourier
00453       complex(8) ::
00454      &              fouvb(ngc,  nxx, (lxx+1)**2)
00455 c     &               fouvp(ngc,  ngc, (lxx+1)**2)
00456 c internal
```

```
00457       integer(4) :: nlx,ig1,ig2,l,n,ir,n1,n2,lm !, ibas
00458 #ifdef COMMONLL
00459       integer(4)::ll(51**2)
00460       common/llblock/ll
00461 #else
00462       integer(4) :: ll
00463       external ll
00464 #endif
00465       real(8)    :: pi,fpi,tpiba, qg1(3),
00466      &  fkk(0:lx),fkj(0:lx),fjk(0:lx),fjj(0:lx),absqg1,absqg2,
00467      &  fac,radint,radsigo(0:lx),radsig(0:lx),phi(0:lx),psi(0:lx)
00468      &   ,r2s,sig,sig1,sig2,sigx(0:lx),sig0(0:lx) ,qg2(3)
00469       complex(8) :: img =(0d0,1d0),phase
00470       complex(8),allocatable :: pjyl(:,:)
00471       real(8),allocatable ::ajr(:,:,:),a1(:,:,:), !rkpr(:,:),rkmr(:,:),
00472      &  qg(:,:),absqg(:)
00473
00474
00475       real(8):: rofi(nrx),rkpr(nrx,0:lxx),rkmr(nrx,0:lxx),eee
00476       logical :: debug=.false.
00477 c rkpr(nr,0:lx),rkmr(nr,0:lx),
00478 c--------------------------------------------------
00479       if(debug) print *,' mkjp_4:'
00480       nlx = (lx+1)**2
00481       allocate(ajr(1:nr,0:lx,ngc),a1(1:nr,0:lx,ngc),
00482      &  qg(3,ngc),absqg(ngc),
00483      &  pjyl((lx+1)**2,ngc) )
00484
00485       pi    = 4d0*datan(1d0)
00486       fpi   = 4*pi
00487       tpiba = 2*pi/alat
00488       allocate(cy((lx+1)**2),yl((lx+1)**2))
00489       call sylmnc(cy,lx)
00490 !      print *,' mkjp_4: end of sylmnc'
00491 C... q+G and <J_L | exp(i q+G r)>  J_L= j_l/sqrt(e)**l Y_L
00492       do ig1 = 1,ngc
00493         qg(1:3,ig1) = tpiba * (q(1:3)+ matmul(qlat, ngvecc(1:3,ig1)))
00494         qg1(1:3) = qg(1:3,ig1)
00495         absqg(ig1)  = sqrt(sum(qg1(1:3)**2))
00496         absqg1   = absqg(ig1)
00497         phase = exp( img*sum(qg1(1:3)*bas(1:3))*alat  )
00498         call sylm(qg1/absqg1,yl,lx,r2s) !spherical factor Y( q+G )
00499         do lm =1,nlx
00500           l = ll(lm)
00501           pjyl(lm,ig1) = fpi*img**l *cy(lm)*yl(lm) *phase  *absqg1**l
00502           ! <jlyl | exp i q+G r> projection of exp(i q+G r) to jl yl  on MT
00503         enddo
00504       enddo
00505
00506 cc rofi and aj = r**l / (2l+1)!! \times r. Sperical Bessel at e=0.
00507 c      rofi(1) = 0d0
00508 c      do ir   = 1, nr
00509 c        rofi(ir) = b*( exp(a*(ir-1)) - 1d0)
00510 c      enddo
00511 c      do l = 0,lx
00512 c        rkpr(1:nr,l) = rofi(1:nr)**(l      +1 )
00513 c        rkmr(2:nr,l) = rofi(2:nr)**(-l-1   +1 )
00514 c        rkmr(1,l)    = rkmr(2,l)
00515 c      enddo
00516
00517 c rojp
00518       if(debug) print *,' mkjp_4: rojp'
00519       do ig1 = 1,ngc
00520         call wronkj( absqg(ig1)**2, eee, rmax,lx,
00521      o                fkk,fkj,fjk,fjj)
00522         do lm = 1,nlx
00523           l = ll(lm)
00524           rojp(ig1,lm) = (-fjj(l))* pjyl(lm,ig1)
00525         enddo
00526       enddo
00527
00528 c ajr
00529       do ig1 = 1,ngc
00530         do ir  = 1,nr
00531           call bessl(absqg(ig1)**2*rofi(ir)**2,lx,phi,psi)
00532           do l   = 0,lx
00533             ajr(ir,l,ig1) = phi(l)* rofi(ir) **(l +1 )
00534             ! ajr = j_l(sqrt(e) r) * r / (sqrt(e))**l
00535           enddo
00536 ccccccccccccccccccccccccc
00537 c        write(116,'(i3,10d13.6)') ir, rofi(ir), ajr(ir,0:lx,ig1)
00538 ccccccccccccccccccccccccc
00539         enddo
00540 ccccccccccccccccccccccccc
00541 c        write(6,*) ig1,sum(ajr(1:nr,0:lx,ig1))
00542 ccccccccccccccccccccccccc
00543       enddo
```

```
00544
00545 c-------------------------
00546       if(eee==0d0) then
00547 c        print *,' mkjp_4: use sigintAn1 eee=0(r0c=infty) mode'
00548         do ig1 = 1,ngc
00549           call sigintan1( absqg(ig1), lx, rofi, nr
00550      o                  ,a1(1:nr, 0:lx,ig1) )
00551         enddo
00552 c      else
00553 c We need to impliment a version of sigintAn1 to treat eee/=0 case...
00554       endif
00555
00556 c-------------------------
00557 c sgpb
00558       do ig1 = 1,ngc
00559         do lm  = 1,nlx
00560           l = ll(lm)
00561           do n =1,nx(l)                       ! r jl        , r B(r)
00562             if(eee==0d0) then
00563               call gintxx(a1(1,l,ig1),rprodx(1,n,l),a,b,nr, sig )
00564 ccccccccccccccccc
00565 c        write(6,"( ' sgpb= ',3i5,2d14.6)") ig1,n,lm, sgpb(ig1,n,lm)
00566 ccccccccccccccccc
00567             else !for a while, we use this version of sgpb
00568               call sigint_4(rkpr(1,l),rkmr(1,l), lx,a,b,nr, ajr(1,l,ig1),rprodx(1,
    n,l)
00569      &            , rofi, sig)
00570             endif
00571             sgpb(ig1,n,lm) = dconjg(pjyl(lm,ig1))* sig/(2*l+1)*fpi
00572 ccccccccccccccccc
00573 c        write(6,"( ' sgpb= ',3i5,2d14.6)") ig1,n,lm, sgpb(ig1,n,lm)
00574 c        write(6,*)
00575 ccccccccccccccccc
00576           enddo
00577         enddo
00578       enddo
00579 ccccccccccccccccccccccc
00580 c      stop 'test end======================='
00581 ccccccccccccccccccccccc
00582
00583 c---------------------------------------
00584 c sgpp block------->removed
00585 c---------------------------------------
00586
00587 c Fourier
00588 c fouvb
00589       if(debug) print *,' mkjp_4: Four'
00590       fouvb=0d0
00591       do ig1 = 1,ngc
00592         do lm  = 1,nlx
00593           l = ll(lm)
00594           do n =1,nx(l)
00595 ccccccccccccccccccccccccccccccccccccccccccccccc
00596 c          print *,' ig1 lm l n=',ig1,lm,l,n
00597 ccccccccccccccccccccccccccccccccccccccccccccccc
00598             call gintxx(ajr(1,l,ig1), rprodx(1,n,l), a,b,nr,
00599      o                  radint )
00600 ccccccccccccccccccccccccccccccccccccccccccccccc
00601 c          print *,' radint=',radint
00602 ccccccccccccccccccccccccccccccccccccccccccccccc
00603             fouvb(ig1, n, lm) =
00604      &     fpi/(absqg(ig1)**2-eee) *dconjg(pjyl(lm,ig1))*radint !eee is supposed to be negative
00605
00606           enddo
00607         enddo
00608       enddo
00609 ccccccccccccccccccc
00610 c        write(6,*)' fourvb sum=',sum (fouvb)
00611 ccccccccccccccccccc
00612
00613 c-----------------------------
00614 c fouvp block --->removed
00615 c-----------------------------
00616
00617       deallocate(ajr,a1,   qg,absqg,   pjyl)
00618       if (allocated( cy )) deallocate(cy)
00619       if (allocated( yl )) deallocate(yl)
00620       end
00621
00622
00623
00624
00625
00626
00627       real(8) function fac2m(i)
00628 cC A table of (2l-1)!!
00629 c     data fac2l /1,1,3,15,105,945,10395,135135,2027025,34459425/
```

```
00630       logical,save::  init=.true.
00631       real(8),save:: fac2mm(0:100)
00632       if(init) then
00633         fac2mm(0)=1d0
00634         do l=1,100
00635           fac2mm(l)=fac2mm(l-1)*(2*l-1)
00636         enddo
00637       endif
00638       fac2m=fac2mm(i)
00639       end
00640 c=======================================================================
00641       subroutine genjh(eee,nr,a,b,lx, nrx,lxx,
00642      o      rofi,rkpr,rkmr)
00643 C-- Generate radial mesh rofi, spherical bessel, and hankel functions
00644 Cr  rkpr, rkmr are real fucntions --
00645 ci eee=E= -kappa**2 <0
00646 cr      rkpr = (2l+1)!! * j_l(i sqrt(abs(E)) r) * r / (i sqrt(abs(E)))**l
00647 cr      rkmr = (2l-1)!! * h_l(i sqrt(abs(E)) r) * r * i*(i sqrt(abs(E)))**(l+1)
00648 cr rkpr reduced to be r**l*r      at E \to 0
00649 cr rkmr reduced to be r**(-l-1)*r at E \to 0
00650 C-------------------------------------------------------------
00651       implicit none
00652       integer(4):: nr,lx, nrx,lxx,ir,l
00653       real(8):: a,b,eee,psi(0:lx),phi(0:lx)
00654       real(8):: rofi(nrx),rkpr(nrx,0:lxx),rkmr(nrx,0:lxx),fac2m
00655       rofi(1)    = 0d0
00656       do ir      = 1, nr
00657         rofi(ir) = b*( exp(a*(ir-1)) - 1d0)
00658       enddo
00659       if(eee==0d0) then
00660         do l = 0,lx
00661           rkpr(1:nr,l) = rofi(1:nr)**(l +1)
00662           rkmr(2:nr,l) = rofi(2:nr)**(-l-1 +1)
00663           rkmr(1,l)    = rkmr(2,l)
00664         enddo
00665       else
00666         do ir  = 1, nr
00667           call bessl(eee*rofi(ir)**2,lx,phi(0:lx),psi(0:lx))
00668           do l = 0,lx    !fac2m(l)= (2l-1)!!
00669 c           print *,' phi=',l,phi(l),phi(l)*fac2m(l+1)
00670 c           print *,' psi=',l,psi(l),psi(l)/fac2m(l)
00671             rkpr(ir,l) = phi(l)* rofi(ir)**(l +1) *fac2m(l+1)
00672             if(ir/=1) rkmr(ir,l) = psi(l)* rofi(ir) **(-l ) /fac2m(l)
00673           enddo
00674         enddo
00675         rkmr(1,0:lx) = rkmr(2,0:lx)
00676       endif
00677       end
00678 c=======================================================================
00679       subroutine mkjb_4( lxx,lx,nxx,nx,
00680      i                   a,b,nr,nrx,rprodx,
00681      i      rofi,rkpr,rkmr,
00682      o        rojb,sgbb)
00683 C--make integrals in each MT. and the Fourier matrix.
00684       implicit none
00685       integer(4) :: lxx, lx, nxx, nx(0:lxx),nr,nrx
00686       real(8)    :: q(3), rprodx(nrx,nxx,0:lxx),a,b
00687 ci rho-type onsite integral
00688       real(8)    :: rojb(nxx, 0:lxx)
00689 ci sigma-type onsite integral
00690       real(8)    :: sgbb(nxx, nxx, 0:lxx)
00691 c internal
00692       integer(4) :: l,n,ir,n1,n2,l1
00693       real(8)    ::
00694      &  fac, xxx,fpi,pi,sig
00695       real(8) :: rofi(nrx),rkpr(nrx,0:lxx),rkmr(nrx,0:lxx)
00696       pi    = 4d0*datan(1d0)
00697       fpi   = 4*pi
00698 c      real(8),allocatable :: rkpr(:,:),rkmr(:,:)
00699 c
00700 c      allocate(rkpr(nr,0:lx),rkmr(nr,0:lx))
00701 c-------------------------------------------------
00702 c rofi and aj = r**l / (2l+1)!! \times r. Sperical Bessel at e=0.
00703 ccccccccccccccccccccccccccccccc
00704 c      do l = 0,lx
00705 c      do n = 1,nx(l)
00706 c      do n1 = 1,nx(l)
00707 c        call gintxx(rprodx(1:nr,n,l), rprodx(1:nr,n1,l), a,b,nr,
00708 c     o                 xxx )
00709 c      write(6,*)' check rprodx =',l,n,n-n1,xxx
00710 c      enddo
00711 c      enddo
00712 c      enddo
00713 c      stop 'xxx'
00714 ccccccccccccccccccccccccccccc
00715
00716 c      rofi(1)    = 0d0
```

```
00717 c      do ir      = 1, nr
00718 c        rofi(ir) = b*( exp(a*(ir-1)) - 1d0)
00719 c      enddo
00720 c      do l = 0,lx
00721 c        rkpr(1:nr,l) = rofi(1:nr)**(l +1)
00722 c        rkmr(2:nr,l) = rofi(2:nr)**(-l-1) *rofi(2:nr)
00723 c        rkmr(1,l)    = rkmr(2,l)
00724 c      enddo
00725
00726 C... initialize
00727       rojb=0d0
00728       sgbb=0d0
00729 c rojb
00730       fac = 1d0
00731       do l = 0,lx
00732         fac = fac/(2*l+1)
00733         do n = 1,nx(l)
00734           call gintxx(rkpr(1,l), rprodx(1,n,l), a,b,nr,
00735      o                rojb(n,l) )
00736         enddo
00737         rojb(1:nx(l),l) = fac*rojb(1:nx(l),l)
00738       enddo
00739 c sgbb
00740       do l  = 0,lx
00741         do n1 = 1,nx(l)
00742           do n2 = 1,nx(l)
00743             call sigint_4(rkpr(1,l),rkmr(1,l),lx,a,b,nr,rprodx(1,n1,l),rprodx(1,n2,l)
00744      &            , rofi,sig )
00745             sgbb(n1, n2, l)=sig/(2*l+1)*fpi
00746           enddo
00747         enddo
00748       enddo
00749 c      write(6,*) ' rojbsum=', sum(rojb(:,:)),   sum(abs(rojb(:,:)))
00750 c      write(6,*) ' sgbbsum=', sum(sgbb(:,:,:)), sum(abs(sgbb(:,:,:)))
00751 ccccccccccccccccccccccccccccccccccc
00752 c      write(6,*)' sigint 1 1 0=',sgbb(1, 1, 0) !/(16d0*datan(1d0))
00753 c      sgbb(1, 1, 0) =0d0
00754 ccccccccccccccccccccccccccccccccccccc
00755 c      deallocate(rkpr,rkmr)
00756       end
00757
00758
00759 c-------------------------------------------------------------
00760       subroutine sigint_4(rkp,rkm,kmx,a,b,nr,phi1,phi2,rofi, sig)
00761       implicit none
00762       integer(4) :: nr,kmx,k,ir
00763       real(8):: a,b, a1(nr),a2(nr),b1(nr),rkp(nr),rkm(nr),
00764      &    int1x(nr),int2x(nr), phi1(nr), phi2(nr),rofi(nr),sig
00765       real(8),parameter:: fpi = 4d0*3.14159265358979323846d0
00766 c
00767       a1(1) = 0d0;  a1(2:nr) = rkp(2:nr)
00768       a2(1) = 0d0;  a2(2:nr) = rkm(2:nr)
00769       b1(1:nr) = phi1(1:nr)
00770       call intn_smpxxx(a1,b1,int1x,a,b,rofi,nr,0)
00771       call intn_smpxxx(a2,b1,int2x,a,b,rofi,nr,0)
00772 c
00773       a1(1) = 0d0; a1(2:nr) =
00774      &  rkm(2:nr) *( int1x(1)-int1x(2:nr) )+ rkp(2:nr) * int2x(2:nr)
00775       b1(1:nr) = phi2(1:nr)
00776       call gintxx(a1,b1,a,b,nr, sig )
00777       end
00778
00779 c-------------------------------------------------------------
00780       subroutine intn_smpxxx(g1,g2,int,a,b,rofi,nr,lr0)
00781 c-- intergral of two wave function. used in ppdf
00782 c
00783 c int(r) = \int_(r)^(rmax) u1(r') u2(r') dr'
00784 c
00785 c lr0 dummy index, now not used.
00786 c simpson rule ,and with higher rule for odd devision.
00787 c -------------------------------------------------------------
00788       IMPLICIT none
00789       integer nr,ir,lr0
00790       double precision g1(nr),g2(nr),int(nr),a,b,rofi(nr),w1,w2,w3
00791      &    ,ooth,foth
00792       data ooth,foth/0.33333333333333333,1.3333333333333333333/
00793       data w1,w2,w3/0.41666666666666666,0.6666666666666666666,
00794      &             -0.083333333333333333/
00795       if(mod(nr,2).eq.0)
00796 Cstop2rx 2013.08.09 kino     &  stop ' INTN: nr should be odd for simpson integration rule'
00797      &  call rx( ' INTN: nr should be odd for simpson integration rule')
00798 c
00799       int(1)=0.0d0
00800       DO  10  ir = 3,nr,2
00801         int(ir)=int(ir-2)
00802      &            + ooth*g1(ir-2)*g2(ir-2)*( a*(b+rofi(ir-2)) )
00803      &            + foth*g1(ir-1)*g2(ir-1)*( a*(b+rofi(ir-1)) )
```

```
00804      &             + ooth*g1(ir)*g2(ir)*( a*(b+rofi(ir)) )
00805     10 CONTINUE
00806
00807 c At the value for odd points, use the same interpolation above
00808       do 20 ir = 2,nr-1,2
00809         int(ir)=int(ir-1)
00810      &             + w1*g1(ir-1)*g2(ir-1)*( a*(b+rofi(ir-1)) )
00811      &             + w2*g1(ir)  *g2(ir)*  ( a*(b+rofi(ir)  ) )
00812      &             + w3*g1(ir+1)*g2(ir+1)*( a*(b+rofi(ir+1)) )
00813     20 continue
00814       do ir=1,nr
00815         int(ir)=int(nr)-int(ir)
00816       enddo
00817       END
00818
00819 c--------------------------------------------------------------------
00820       subroutine sigintan1( absqg, lx, rofi, nr,
00821      o             a1int)
00822 c a1int(r') = r' * \int_0^a r^2 {r_{<}}^l / (r_{>})^{l+1} *
00823 c                 j_l(absqg r)/absqg**l
00824       implicit none
00825       integer(4) :: nr,l,ir,lx
00826       real(8):: a1int(nr,0:lx), rofi(nr),absqg
00827       real(8)::
00828      &   ak(0:lx) ,aj(0:lx), dk(0:lx), dj(0:lx),
00829      &   aknr(0:lx),ajnr(0:lx),dknr(0:lx),djnr(0:lx),
00830      &   phi(0:lx),psi(0:lx)
00831 c---
00832 c      print *,' sigintAn1: absqg=',absqg
00833       if(absqg<1d-10) then
00834 c      if(absqg<1d-6) then !23jan2004 1d-10 ok?
00835 Cstop2rx 2013.08.09 kino        stop "sigintAn1: absqg=0 is not supported yet. Improve here."
00836         call rx( "sigintAn1: absqg=0 is not supported yet. Improve here.")
00837 c This part for absqg=0 has not been checked yet!
00838 c       call bessl(0d0,lx,phi,psi)
00839 c        do ir = 1,nr
00840 c        do l  = 0,lx
00841 c          a1int(ir,l) = .5d0* rofi(nr)**2     * rofi(ir)**l     * phi(l)
00842 c     &                +(1d0/(2d0*l+3d0)-.5d0) * rofi(ir)**(l+2) * phi(l)
00843 c       enddo
00844 c        enddo
00845       else
00846         call  radkj(absqg**2, rofi(nr),lx,aknr,ajnr,dknr,djnr,0)
00847         a1int(1,:) = 0d0
00848         do ir = 2,nr
00849           call radkj(absqg**2, rofi(ir),lx,ak,aj,dk,dj,0)
00850           do l  = 0,lx
00851             a1int(ir,l) = ( (2*l+1)* aj(l)
00852      &     -((l+1)* ajnr(l)+ rofi(nr)*djnr(l) )*(rofi(ir)/rofi(nr))**l)
00853      &     /absqg**2
00854      &    *rofi(ir)
00855           enddo
00856         enddo
00857       endif
00858 c      print *,' sigintAn1: end'
00859       end
00860
00861 c--------------------------------------------------
00862       subroutine sigintpp( absqg1, absqg2, lx, rmax,
00863      o             sig)
00864 c sig(l)   =  \int_0^a r^2 {r_{<}}^l / (r_{>})^{l+1} *
00865 c               j_l(absqg1 r)/absqg1**l
00866 c               j_l(absqg2 r)/absqg2**l
00867 c e1\ne0 e2\ne0
00868       implicit none
00869       integer(4) :: l,lx
00870       real(8)::  rmax,sig(0:lx), absqg1,absqg2, e1,e2,
00871      &   ak1(0:lx) ,aj1(0:lx), dk1(0:lx), dj1(0:lx),
00872      &   ak2(0:lx) ,aj2(0:lx), dk2(0:lx), dj2(0:lx),
00873      &  fkk(0:lx),fkj(0:lx),fjk(0:lx),fjj(0:lx)
00874 c---
00875       e1 = absqg1**2
00876       e2 = absqg2**2
00877 c
00878 c      print *," sigintpp",e1,e2
00879 c
00880       call wronkj( e1,e2, rmax,lx,   fkk,fkj,fjk,fjj )
00881       call  radkj( e1,    rmax,lx,   ak1,aj1,dk1,dj1,0)
00882       call  radkj( e2,    rmax,lx,   ak2,aj2,dk2,dj2,0)
00883 c
00884       do l = 0,lx
00885         sig(l)= ( -l*(l+1)*rmax*aj1(l)*aj2(l)
00886      &            + rmax**3 * dj1(l)*dj2(l)
00887      &            + 0.5d0*rmax**2* (aj1(l)*dj2(l)+aj2(l)*dj1(l))
00888      &            - fjj(l)*(2*l+1)*(e1+e2)/2d0
00889      &           ) /(e1*e2)
00890       enddo
```

```
00891       end
00892
```

4.19 gwsrc/mkqg.F File Reference

Data Types

- module m_q0p

  Q0P (offset Gamma points) generator.

Functions/Subroutines

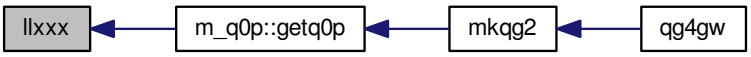
- subroutine mkqg2 (alat, plat, symops, ngrp, n1q, n2q, n3q, iq0pin, QpGcut_psi, QpGcut_Cou, ifiqg, ifiqgc)
- double precision function tripl (a, b, c)
- integer(4) function llxxx (ilm)

4.19.1 Function/Subroutine Documentation

4.19.1.1 integer(4) function llxxx (*ilm*)

Definition at line 983 of file mkqg.F.

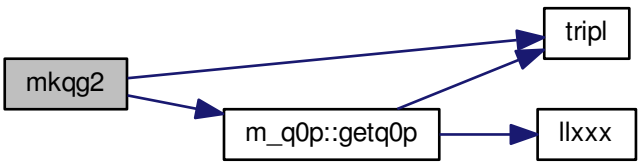
Here is the caller graph for this function:



4.19.1.2 subroutine mkqg2 (real(8) *alat,* real(8), dimension(3,3) *plat,* real(8), dimension(3,3,ngrp) *symops,* integer(4) *ngrp,* integer(4) *n1q,* integer(4) *n2q,* integer(4) *n3q,* integer(4) *iq0pin,* real(8) *QpGcut_psi,* real(8) *QpGcut_Cou,* integer(4) *ifiqg,* integer(4) *ifiqgc*)

Definition at line 358 of file mkqg.F.

Here is the call graph for this function:

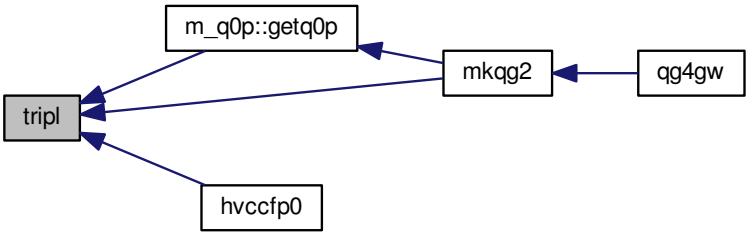


Here is the caller graph for this function:

4.19.1.3 double precision function tripl (dimension(0:2) *a*, dimension(0:2) *b*, dimension(0:2) *c*)

Definition at line 966 of file mkqg.F.

Here is the caller graph for this function:



4.20 mkqg.F

```
00001 !> Q0P (offset Gamma points) generator
00002       module m_q0p
00003 !! All of them are outputs when getq0p is called.
00004       real(8),allocatable:: q0i(:,:),wt(:)  ! Q0P and its weight.
00005       integer:: nq0i                        ! Number of Q0P
00006       integer,private:: nq0x,nmm !not output
00007 
00008       contains
00009 !>- Q0P data set is given for 'getq0p'
00010       subroutine getq0p(newoffsetG,alat,plat,qlat,n1q,n2q,n3q,alp,alpv,
00011      & ngcxx,ngcx,nqbz,nqibz,nstbz,qbz,qibz,symops,ngrp,ngvect)
00012 !! this is called in subroutine mkqg2
00013 !!  output
00014 !!    q0i (offset Gamma point)
00015 !!    wt  (weight of q0i)
00016 !!    EPSwklm (file)
00017 !! All arguments are input.
00018 !! In addition, we write a file EPSwklm, which is key for new offset Gamma method.
00019 !! deltaq_scale() given by Q0Pchoice in GWinput change the of offset Gamma method.
00020       use keyvalue,only: getkeyvalue
00021       implicit none
00022       logical,intent(in):: newoffsetg
00023       integer,intent(in):: n1q,n2q,n3q,nstbz(*),nqbz,nqibz,ngcxx,ngcx(nqbz),ngrp
00024       real(8),intent(in):: alat,qlat(3,3),alp,alpv(3),plat(3,3)
00025      & ,qbz(3,nqbz),qibz(3,nqibz),symops(3,3,ngrp)
00026       integer,intent(in) ::ngvect(3,ngcxx,nqbz)
00027 
00028       integer::nq00ix,nx0,nq00i, xyz2lm(3),nnn
00029       real(8):: xn !,www,wgtq0
00030       logical:: noq0p,timereversal
```

```
00031       real(8),allocatable:: q0x(:,:),wt0(:)
00032       real(8):: deltaq,deltaq_scale,delta8,delta5,emat(3,3)
00033       real(8):: pi=4d0*atan(1d0)
00034       real(8),allocatable:: wti(:),qi(:,:),epinv(:,:,:),cg(:,:,:),matxxl(:,:,:),
00035      &     dmlx(:,:),cy(:),yl(:),epinvq0i(:,:),wklm(:) !,norq0x(:) !,wqfac(:)
00036       integer:: bzcase=1,i,iq0i,ifidmlx,iopen,lmxax,lx,lxklm,j,iclose,llxxx
00037       real(8):: rrr(3),r2s,qxx(3),voltot,tripl
00038       integer,allocatable::irrx(:)
00039       voltot = abs(alat**3*tripl(plat,plat(1,2),plat(1,3)))
00040 !! number of spherical points.
00041 c  nq00ix=12 !spherical points
00042 c  nq00ix=3   !spherical points
00043       nq00ix=6
00044       nx0 = 1
00045       if(nx0==2) xn=3d0 ! ratio parameter for Q2 and Q1,
00046 ! only effective for nx0=2 case
00047 c      nq0x=nq00ix*nx0
00048       nq0x=nq00ix
00049
00050 c     nq0x=4*nx0
00051 c     if(q0pchoice()/1000==1) then
00052 c     nn1= (q0pchoice()-1000)/10
00053 c     nn2= mod(q0pchoice()-1000,10)
00054 c     nq0x= 4*nn1*nn2
00055 c     print *,' mkqg: q0pchoice nq0x=',q0pchoice(),nq0x
00056 c     endif
00057 c$$$         if(newaniso) then    !feb2012
00058 c$$$      nq0x=nq00ix
00059 c$$$         elseif( q0pchoice()<0) then
00060 c$$$c     nq0x = 8*abs (q0pchoice())
00061 c$$$            nq0x = max( (2*abs(q0pchoice()))**3, 8*abs(q0pchoice()))
00062 c$$$         endif
00063 c      www = wgtq0p()
00064
00065       call getkeyvalue("GWinput","TestNoQ0P",noq0p,default=.false.)
00066       if(noq0p) then
00067         nq00i=0
00068         print *,' TestNoQ0P=.true. '
00069         nq0i=0
00070       else
00071         nmm=1
00072         if(.not.timereversal()) nmm=2
00073         allocate( q0x(3,nq0x), wt0(nq0x), irrx(nq0x), wt(nq0x), q0i(3,nq0x*nmm))
00074         if(newoffsetg) then
00075           deltaq=deltaq_scale()*alat/(2*pi) !dq is 0.01 a.u.
00076           if(nq00ix==3) then
00077             nq00i=3
00078 c     q0x(:,1)=(/-deltaq, deltaq,  deltaq/)
00079 c     q0x(:,2)=(/deltaq, -deltaq,  deltaq/)
00080 c     q0x(:,3)=(/deltaq, deltaq,  -deltaq/)
00081             q0x(:,1)= qlat(:,1)/n1q/2d0*deltaq_scale()
00082             q0x(:,2)= qlat(:,2)/n2q/2d0*deltaq_scale()
00083             q0x(:,3)= qlat(:,3)/n3q/2d0*deltaq_scale()
00084           elseif(nq00ix==6) then
00085 !! six independent direction is required to calculate full dielectric matrix (symmetric -->six components).
00086             nq00i=6
00087             q0x(:,1)= qlat(:,1)/n1q/2d0*deltaq_scale()
00088             q0x(:,2)= qlat(:,2)/n2q/2d0*deltaq_scale()
00089             q0x(:,3)= qlat(:,3)/n3q/2d0*deltaq_scale()
00090 c                   norq0x(1)=sqrt(sum(q0x(:,1)**2))
00091 c                   norq0x(2)=sqrt(sum(q0x(:,2)**2))
00092 c                   norq0x(3)=sqrt(sum(q0x(:,3)**2))
00093 c before 21dec2012
00094 c     q0x(:,4)= (q0x(:,1)-q0x(:,2))/2d0
00095 c     q0x(:,5)= (q0x(:,2)-q0x(:,3))/2d0
00096 c     q0x(:,6)= (q0x(:,3)-q0x(:,1))/2d0
00097 c     norq0x(4)=sqrt(sum(q0x(:,4)**2))
00098 c     norq0x(5)=sqrt(sum(q0x(:,5)**2))
00099 c     norq0x(6)=sqrt(sum(q0x(:,6)**2))
00100 c     q0x(:,4)= (q0x(:,1)-q0x(:,2))/norq0x(4)*(norq0x(1)+norq0x(2))/2d0
00101 c     q0x(:,5)= (q0x(:,2)-q0x(:,3))/norq0x(5)*(norq0x(2)+norq0x(3))/2d0
00102 c     q0x(:,6)= (q0x(:,3)-q0x(:,1))/norq0x(6)*(norq0x(3)+norq0x(1))/2d0
00103 !! shorter ones. no normalization. dec2012
00104             if(sum((q0x(:,1)-q0x(:,2))**2)<sum((q0x(:,1)+q0x(:,2))**2)) then
00105               q0x(:,4)= (q0x(:,1)-q0x(:,2))/2d0
00106             else
00107               q0x(:,4)= (q0x(:,1)+q0x(:,2))/2d0
00108             endif
00109             if(sum((q0x(:,2)-q0x(:,3))**2)<sum((q0x(:,2)+q0x(:,3))**2)) then
00110               q0x(:,5)= (q0x(:,2)-q0x(:,3))/2d0
00111             else
00112               q0x(:,5)= (q0x(:,2)+q0x(:,3))/2d0
00113             endif
00114             if(sum((q0x(:,3)-q0x(:,1))**2)<sum((q0x(:,3)+q0x(:,1))**2)) then
00115               q0x(:,6)= (q0x(:,3)-q0x(:,1))/2d0
00116             else
00117               q0x(:,6)= (q0x(:,3)+q0x(:,1))/2d0
```

```
00118             endif
00119 c     q0x(:,1)=(/-deltaq, deltaq,  deltaq/)
00120 c     q0x(:,2)=(/deltaq, -deltaq,  deltaq/)
00121 c     q0x(:,3)=(/deltaq, deltaq,  -deltaq/)
00122 c     q0x(:,4)=(/deltaq, -deltaq,  -deltaq/)
00123 c     q0x(:,5)=(/-deltaq, deltaq,  -deltaq/)
00124 c     q0x(:,6)=(/-deltaq, -deltaq,  deltaq/)
00125 c     nq00i=6
00126 c     q0x(:,1)=(/deltaq, 0d0,    0d0/)
00127 c     q0x(:,2)=(/0d0, deltaq,    0d0/)
00128 c     q0x(:,3)=(/0d0,    0d0, deltaq/)
00129 c     q0x(:,4)=(/0d0,   deltaq, deltaq/)
00130 c     q0x(:,5)=(/deltaq, 0d0,   deltaq/)
00131 c     q0x(:,6)=(/deltaq,deltaq, 0d0/)
00132           elseif(nq00ix==12) then
00133 !! spherical design des.3.12.5
00134 !! des.3.12.5
00135             nq00i=12
00136             delta8=0.850650808352d0*deltaq
00137             delta5=0.525731112119d0*deltaq
00138             q0x(:,1)=(/delta8, 0d0, -delta5/)
00139             q0x(:,2)=(/delta5, -delta8, 0d0/)
00140             q0x(:,3)=(/0d0,-delta5, delta8/)
00141
00142             q0x(:,4)=(/delta8, 0d0, delta5/)
00143             q0x(:,5)=(/-delta5,-delta8,0d0/)
00144             q0x(:,6)=(/0d0,delta5,-delta8/)
00145
00146             q0x(:,7)=(/-delta8,0d0,-delta5/)
00147             q0x(:,8)=(/-delta5,delta8,0d0/)
00148             q0x(:,9)=(/0d0,delta5,delta8/)
00149
00150             q0x(:,10)=(/-delta8,0d0,delta5/)
00151             q0x(:,11)=(/delta5,delta8,0d0/)
00152             q0x(:,12)=(/0d0,-delta5,-delta8/)
00153           else
00154             call rx( 'mkqg: not implemented nq00i')
00155           endif
00156           do i=1,nq00i
00157             write(*,'(" initial q0x=",i3,3f9.3)')i,q0x(:,i)
00158           enddo
00159 !! invariante dielectoric tensor.
00160           allocate(epinv(3,3,nq0x))
00161           call diele_invariant(q0x,nq0x,symops,ngrp,  epinv,q0i,nq0i, wt)
00162           print *,' nq0x,nmm nq0i=',nq0x,nmm,nq0i
00163 !! == To convert invariant tensor on YL representation (Y00 and Y2m) ==
00164           lmxax=1
00165           allocate( cg((lmxax+1)**2,(lmxax+1)**2,(2*lmxax+1)**2) )
00166           allocate( matxxl(3,3,(2*lmxax+1)**2) )
00167           call rotcg(lmxax,(/1d0,0d0,0d0,0d0,1d0,0d0,0d0,0d0,1d0/),1,cg)
00168           xyz2lm( 2)=-1         !y
00169           xyz2lm( 3)= 0         !z
00170           xyz2lm( 1)= 1         !x
00171 !! matxxl(i,j,L) = \int d\Omega x_i x_j  Y_L(\Omega), where x_i are nomlized.
00172           do i=1,3
00173             do j=1,3
00174               matxxl(i,j,:) = cg(xyz2lm(i)+3,xyz2lm(j)+3,:)*4d0*pi/3d0
00175 !sqrt(4*pi/3) comes from normalization of Y_l=1.
00176             enddo
00177           enddo
00178 !! epinv is expanded as
00179 !!   <ehat| epinv|ehat> = \sum_lm dmlx(iq0i,lm) *Y_lm(ehat)
00180           allocate(dmlx(nq0i,9))
00181           do iq0i=1,nq0i
00182             do lx=1,9
00183               dmlx(iq0i,lx)=sum(epinv(:,:,iq0i)*matxxl(:,:,lx))
00184             enddo
00185           enddo
00186 c$$$  !! check xxxxxxxxxxxxxxxxxxx
00187 c$$$  do lx=5,9
00188 c$$$  do i=2,4
00189 c$$$  do j=2,4
00190 c$$$  write(*,"(' l1 l2 l= cg=',3i3,f9.5)")i-1,j-1,lx-7,cg(i,j,lx)
00191 c$$$  enddo
00192 c$$$  enddo
00193 c$$$  write(*,*)
00194 c$$$  enddo
00195 c$$$  do lx=5,9
00196 c$$$  do i=1,3
00197 c$$$  do j=1,3
00198 c$$$  write(*,"(' matxxl l1 l2 l= cg=',3i3,f9.5)")i,j,lx,matxxl(i,j,lx)
00199 c$$$  enddo
00200 c$$$  enddo
00201 c$$$  write(*,*)
00202 c$$$  enddo
00203 c$$$  do lx=1,1
00204 c$$$  do i=1,3
```

```
00205 c$$$  do j=1,3
00206 c$$$  write(*,"(' matxxl l1 l2 l= cg=',3i3,f9.5)")i,j,lx,matxxl(i,j,lx)
00207 c$$$  enddo
00208 c$$$  enddo
00209 c$$$  write(*,*)
00210 c$$$  enddo
00211 c$$$  do iq0i=1,nq0i
00212 c$$$  do lx=1,9
00213 c$$$  write(*,"(' iq0i lx dmlx=',2i3,f9.3)")iq0i,lx,dmlx(iq0i,lx)
00214 c$$$  enddo
00215 c$$$  enddo
00216
00217 !! Test for one r vector as for <ehat|epinv|ehat> = \sum_lm dmlx(iq0i,lm) *Y_lm(ehat)
00218 !! === generate YL for a test vector rrr (rrr is ehat above).====
00219           lx=2
00220           allocate(cy((lx+1)**2),yl((lx+1)**2))
00221           call sylmnc(cy,lx)
00222           rrr=(/.5d0,-.1d0,-0.7d0/) !test data
00223           rrr=rrr/sqrt(sum(rrr**2))
00224           call sylm(rrr,yl,lx,r2s) !spherical factor Y( q+G )
00225 !! ===== check (rrr*emat*rrr = sum(dmlx* YL)
00226 c     do lm=1,9; write(*,"('r lm=',3f8.3,i4,' ylm=',f8.3)") rrr,lm,cy(lm)*yl(lm) ;   enddo
00227           write(*,*)
00228           write(*,"(  ' test: r=',3f10.5)") rrr
00229 !!
00230           do iq0i=1,nq0i
00231             write(*,"(' test: ylm   expansion=',i3,f10.5)")
00232      &       iq0i,sum(dmlx(iq0i,:)*cy(:)*yl(:))
00233             emat=epinv(:,:,iq0i)
00234             write(*,"(' test: epinv expansion=',i3,f10.5)")
00235      &       iq0i,sum(rrr*matmul(emat,rrr))
00236           enddo
00237           write(*,*)
00238           allocate( epinvq0i(nq0i,nq0i))
00239           do i=1,nq0i
00240           do j=1,nq0i         !epinvq0i= <q0i/|q0i|| epinv(:,:,iq0j)|q0i/|q0i|>
00241            epinvq0i(i,j)=sum(q0i(:,i)*matmul(epinv(:,:,j),q0i(:,i)))/sum(q0i(:,i)**2)
00242           enddo
00243           enddo
00244           deallocate(cy,yl)
00245           lxklm=6         !this is used for inversion procedure in hx0fp0.sc.m.f
00246           nnn=n1q*n2q*n3q
00247           allocate(wklm((lxklm+1)**2)) !wklm-->Klm in Comp.Phys. Comm 176(1007)1-13
00248           call getwklm(alat,voltot,plat,qlat,alp,qbz,nqbz,ngcx,ngcxx,ngvect, lxklm,n1q,n2q,n3q,
00249      o     wklm)                !,wqfac)
00250 !! Takao think anisotropic treatment in Computer Phys. Comm 176(1007)1-13
00251 !! (in our version with auxially function) can be numerically proremetic.
00252 !! We only keep wklm only up to l=2. (I observed high wklm(lm) components
00253 !! are largely negative --->it may cause numerical error).
00254 !!
00255 !! From the begining, we can only excpect "virtual convergence on Nz" for
00256 !! NxNyNz for Si100 slab model in the paper.
00257 !! (I still not understand why it does not show divergent behevior in the anisotropic case).
00258 !!
00259 c     print *,' set wklm=0 for l>2. But lxklm(for inversion of epsioln)=',lxklm
00260           do i=1,(lxklm+1)**2
00261             if(abs(wklm(i))>1d-6 ) write(6,'(" mkqg: l lm Wklm=",2i3,f9.4)')
      llxxx(i),i,wklm(i)
00262           enddo
00263 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00264 !     ! spherical design des.3.12.5 check. Because of angular momentum synsesize,
00265 !     ! des.3.12.5 gives correct normalization of product up to l=2 (lm<=9)
00266 c$$$  deallocate(cy,yl,q0x)
00267 c$$$  nq00i=12
00268 c$$$  allocate(cy((lxklm+1)**2),yll((lxklm+1)**2,nq00i))
00269 c$$$  tpiba  = 2d0*pi/alat
00270 c$$$  call sylmnc(cy,lxklm)
00271 c$$$  allocate( q0x(3,nq00i) )
00272 c$$$  delta8=0.850650808352d0 !*deltaq
00273 c$$$  delta5=0.525731112119d0 !*deltaq
00274 c$$$  q0x(:,1)=(/delta8, 0d0, -delta5/)
00275 c$$$  q0x(:,2)=(/delta5, -delta8, 0d0/)
00276 c$$$  q0x(:,3)=(/0d0,-delta5, delta8/)
00277 c$$$  q0x(:,4)=(/delta8, 0d0, delta5/)
00278 c$$$  q0x(:,5)=(/-delta5,-delta8,0d0/)
00279 c$$$  q0x(:,6)=(/0d0,delta5,-delta8/)
00280 c$$$  q0x(:,7)=(/-delta8,0d0,-delta5/)
00281 c$$$  q0x(:,8)=(/-delta5,delta8,0d0/)
00282 c$$$  q0x(:,9)=(/0d0,delta5,delta8/)
00283 c$$$  q0x(:,10)=(/-delta8,0d0,delta5/)
00284 c$$$  q0x(:,11)=(/delta5,delta8,0d0/)
00285 c$$$  q0x(:,12)=(/0d0,-delta5,-delta8/)
00286 c$$$  do iq=1,nq00i
00287 c$$$  qg(1:3) = q0x(:,iq)
00288 c$$$  call sylm(qg/sum(qg**2),yll(:,iq),lxklm,r2s) !spherical factor Y( q+G )
00289 c$$$  c        print *,' qg for yll=',iq, qg
00290 c$$$  c        print *,' yll=',cy(1:20)*yll(1:20,iq)
```

```
00291 c$$$  enddo
00292
00293 !! normalization check
00294 c     do lm1=1,(3+1)**2
00295 c     do lm2=lm1,(3+1)**2
00296 c     aaa=sum(cy(lm1)*cy(lm2)*yll(lm1,:)*yll(lm2,:))/12d0*4d0*pi
00297 c     if(abs(aaa)>1d-6) write(6,"('ylm*ylm=',2i3,d13.5)")lm1,lm2,aaa
00298 c     enddo
00299 c     enddo
00300 c     do lm1=1,(5+1)**2
00301 c     aaa=sum(cy(lm1)*yll(lm1,:))
00302 c     if(abs(aaa)>1d-6) write(6,"('ylm*ylm=',i3,d13.5)")lm1,aaa
00303 c     enddo
00304 c     stop ' xxxxxxxxx spherical normalization xxxxxxxx'
00305 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00306
00307 c$$$  !! test function generation
00308 c$$$  deallocate(cy,yl)
00309 c$$$  allocate(cy((lxklm+1)**2),yl((lxklm+1)**2))
00310 c$$$  tpiba  = 2d0*pi/alat
00311 c$$$  call sylmnc(cy,lxklm)
00312 c$$$  do iq=1,nqbz
00313 c$$$  funa(:,iq)=0d0
00314 c$$$  do ig=1,ngcx(iq)
00315 c$$$  qg(1:3) = tpiba * (qbz(1:3,iq)+ matmul(qlat, ngvect(1:3,ig,iq)))
00316 c$$$  qg2     = sum(qg(1:3)**2)
00317 c$$$  alpqg2= alp* qg2
00318 c$$$  call sylm(qg/sqrt(qg2),yl,lxklm,r2s) !spherical factor Y( q+G )
00319 c$$$  funa(:,iq) = funa(:,iq) + exp(-alpqg2)*(1d0+1/(qg2+5d0))/qg2*cy(:)*yl(:) !cy*yl =Y_L(qg/|qg|)
00320 c$$$  enddo
00321 c$$$  enddo
00322 c$$$  c what is wtrue???
00323 c$$$  ccccccccccccccccccccccc
00324 c$$$  do lm=1,(lxklm+1)**2
00325 c$$$  wsumau(lm) = sum(funa(lm,2:nqbz))/dble(nqbz)
00326 c$$$  c        write(6,"('  wsum fnua=',i3,8f10.5)") lm,wsumau(lm)
00327 c$$$  if(lm==1) then
00328 c$$$  write(*,"('lm l wklm wtrue wsum wsummesh',2i3,4f12.8)")
00329 c$$$  &     lm,llxxx(lm),wklm(lm), wtrue00,wklm(lm)+wsumau(lm), wsumau(lm)!,wklm(lm)+wsumau(lm)-wtrue00
00330 c$$$  else
00331 c$$$  write(*,"('lm l wklm wtrue wsum wsummesh',2i3,4f12.8)")
00332 c$$$  &     lm,llxxx(lm),wklm(lm), 0d0,  wklm(lm)+wsumau(lm), wsumau(lm)
00333 c$$$  endif
00334 c$$$  enddo
00335 c$$$  stop 'xxxxxxxxxxxxxxxxxxxxxxx'
00336 c$$$  ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00337           ifidmlx = iopen('EPSwklm',0,-1,0)
00338           write(ifidmlx) nq0i,lxklm
00339           write(ifidmlx) dmlx, epinv(:,:,1:nq0i),epinvq0i
00340           write(ifidmlx) wklm
00341           ifidmlx = iclose('EPSwklm')
00342         else
00343           call setq0_2(bzcase, alat, voltot,plat, qlat,alpv, qbz, nstbz, nqbz,
00344      i     ngcx, ngcxx, ngvect,   nq0x,nx0,xn,n1q,n2q,n3q,
00345      o     q0x,wt0,nq00i)
00346 !     ! ... inequivalent q0x points ...
00347           nq0i=0
00348           call q0irre(qibz,0,q0x,wt0,nq00i,symops,ngrp,   q0i,nq0i,wt,plat,.false.,0,irrx)
00349         endif
00350         deallocate(irrx)
00351       endif
00352       write(6,"('i wt q0i=',i3,f16.7,2x,3d23.15)")(i,wt(i),q0i(1:3,i),i=1,nq0i)
00353       end subroutine getq0p
00354       end module m_q0p
00355
00356
00357 !! ------------------------------------------------------------------------
00358       subroutine mkqg2(alat,plat,symops,ngrp,n1q,n2q,n3q,iq0pin,
00359      &     qpgcut_psi, qpgcut_cou, ifiqg, ifiqgc)
00360       use m_get_bzdata1,only:  getbzdata1, !call getbzdata1
00361      & nqbz, nqibz, nqbzw,ntetf,nteti,nqbzm,nqibz_r,
00362      & qbz,wbz,qibz,wibz,
00363      & qbzw, qbasmc, qibz_r,
00364      & idtetf, ib1bz, idteti,
00365      & irk, nstar, nstbz,
00366      & qbzm, qbzwm
00367       use keyvalue,only: getkeyvalue
00368       use m_q0p,only: getq0p,
00369      & q0i,wt,nq0i
00370 !! == Make required q and G in the expantion of GW. ==
00371 !!     |q+G| < QpGcut_psi for eigenfunction psi.
00372 !!     |q+G| < QpGcut_Cou for coulomb interaction
00373 !!
00374 !! OUTPUT
00375 !!     file handle= ifiqg,  which contains q and G points for eigenfunction psi. --> QGpsi
00376 !!     file handle= ifiqgc, which contains q and G points for Coulomb            --> QGcou
00377 !!
```

```
00378 !!     QGpsi(ifiqg), QGcou(ifiqgc), Q0P are written.
00379 !!     See the end of console output.
00380 !! -----------------------------------------------
00381       implicit none
00382       integer(4) ::n1q,n2q,n3q,ifiqg,ifiqgc,nnn,ngcxx,
00383      &     ngrp,i,j,iq,iq00,ngp,ngpmx,ngc,ngcmx,nqnum,iq0pin,
00384      &     nline,nlinemax,ifsyml,iqq,is,nk,ix,nqnumx,i1,ifkpt
00385       real(8)  :: plat(3,3),qlat(3,3),q(3),dummy,qp(3),
00386      &     qpgcut_psi, qpgcut_cou,qpgcut,alpv(3),q0smean,sumt,alp,
00387      &     volum,voltot,pi,q0(3),qlat0(3,3), alat,tripl,
00388      &     symops(3,3,ngrp),xx,qqx(3),alpm
00389       integer(4),allocatable:: ngvecp(:,:), ngvecc(:,:),
00390      &     ngpn(:),ngcn(:),ngvect(:,:,:),ngcx(:), nqq(:)
00391       real(8),allocatable :: q0x(:,:),
00392      &     qq(:,:),qq1(:,:),qq2(:,:),qqm(:,:)
00393       real(8) :: qbas(3,3),vol,ginv(3,3),aaa,det,dq(3) !,www
00394       integer(4) :: mxkp,ifiqibz,iqibz,ifigwin,mtet(3),nm1,nm2,nm3
00395       logical ::tetrai,tetraf,tetra_hsfp0
00396       integer(4):: ifbz
00397       integer(4):: bzcase=1
00398 c     logical :: readgwinput
00399       integer(4):: nqnumm,ifiqmtet,verbose,q0pchoice,nn1,nn2,ifiqbz,iqbz !,auxfunq0p
00400       real(8)::aaij,bbij, qdum(6)
00401       logical:: qbzreg
00402
00403       logical :: qreduce ,qreduce0 ,ibzqq
00404       real(8),allocatable:: qsave(:,:),   qmin(:,:),qmax(:,:)
00405       integer:: imx,ifinin,il,nq0i0,ni,nq0i00,imx0
00406       integer,allocatable :: ndiv(:),ngvecprev(:,:,:),ngveccrev(:,:,:)
00407
00408       real(8):: ddq(3)
00409       logical :: offmesh=.false. ,offmeshg=.false.
00410       logical :: regmesh=.false. ,regmeshg=.false. ,  timereversal
00411
00412       logical ::anyq                , caca,debug=.true. !,newaniso
00413       real(8),allocatable:: qany(:,:)
00414       integer(4):: nany,ifqpnt,ret,imxc,nnn3(3),imx0c,imx11(1,1)
00415       real(8):: deltaq,delta5,delta8,deltaq_scale!=1d0/3.0**.5d0
00416
00417       integer:: nqi,ifix,ig,iq0i,lm
00418       real(8),allocatable:: wti(:),qi(:,:)
00419       integer:: ifidmlx,iclose,iopen !,ifiwqfac
00420
00421       integer:: llxxx,lm1,lm2
00422       real(8),allocatable:: funa(:,:),wsumau(:),yll(:,:)
00423       real(8)::volinv,wtrue00,qg(3),alpqg2,qg2,tpiba
00424       character*99:: q0pf        !nov2012
00425       integer:: dummyia(1,1),iimx,irradd,nmax
00426       real(8):: epstol=1d-8,tolq=1d-6,qx(3),qxx(3)
00427       logical :: newoffsetg !july2014
00428       real(8),allocatable:: wt0(:)
00429       integer,allocatable::irr(:)
00430       real(8):: dq_(3),dq__(3)
00431       integer:: nq0itrue
00432 c-----------------------------------------------
00433       print *,' mkqg2: '
00434       qreduce0 = qreduce()
00435       newoffsetg=.true. !newaniso()
00436       if(iq0pin == 101) then
00437          iq0pin=1
00438          newoffsetg=.false. !for old oldset Gamma case
00439       endif
00440
00441 !! band case --- iq0pin == 3 ==>read syml file
00442 !!     nqq(is),qq1(1:3,is),qq2(1:3,is),is =1,nline
00443       if(iq0pin == 3) then
00444          qreduce0=.false.
00445          nlinemax = 50
00446          allocate(nqq(nlinemax),qq1(1:3,nlinemax),qq2(1:3,nlinemax))
00447          ifsyml = 3001
00448          open(ifsyml,file='SYML')
00449          nline = 0
00450          do
00451             nline = nline + 1
00452             read(ifsyml,*,err=601,end=601)
00453      &       nqq(nline),qq1(1:3,nline),qq2(1:3,nline)
00454          enddo
00455  601     continue
00456          close(ifsyml)
00457          nline = nline - 1
00458          write(6,"(/' Symmetry lines:'/' points',12x,'start',22x,'end')")
00459          do is=1,nline
00460             write(6,"(i6,2x,3f8.4,2x,3f8.4)")
00461      &      nqq(is),(qq1(i,is),i=1,3),(qq2(i,is),i=1,3)
00462          enddo
00463          nqnumx = sum(nqq(1:nline))
00464          allocate( qq(1:3,nqnumx),irr(nqnumx) )
```

```
00465          iqq = 0
00466          do is = 1,nline
00467             nk = nqq(is)
00468             do iq=1,nk
00469                xx = 0d0
00470                if(nk>1) xx=(iq-1d0)/(nk-1d0)
00471                qqx = xx*qq2(1:3,is)+(1d0-xx)*qq1(1:3,is)
00472                iqq = iqq + 1
00473                qq(1:3,iqq) = qqx
00474                write (6,"('  q=',3f7.3)") qq(1:3,iqq)
00475             enddo
00476          enddo
00477          nqnum = iqq
00478          write (6,"(' Total number of q-points:',i5/)") nqnum
00479          call dinv33x(plat,qlat) !it was dinv33(plat,1,qlat) by Ferdi
00480          goto 2001
00481       endif
00482
00483 !! --- Ordinary case --- iq0pin == 1 or 2
00484       voltot = abs(alat**3*tripl(plat,plat(1,2),plat(1,3)))
00485       call dinv33x(plat,qlat)  !it was dinv33(plat,1,qlat) by Ferdi
00486       call getkeyvalue("GWinput","delta",aaa)
00487       if(aaa<0d0) then
00488          print * ,'READ GWIN_V2 --->: tetrahedron method for x0'
00489          tetraf=.true.
00490       else
00491          print * ,'READ GWIN_V2 --->: not use tetrahedron method for x0'
00492          tetraf=.false.
00493       endif
00494       tetrai = .true.           !used in heftet tetra_hsfp0()
00495 !! ---  See indxk in index.f \in genbz2 \in genallc_v2
00496       call dinv33(qlat,0,ginv,det)
00497       write(6,*)'=== plat ==='
00498       write(6,"(3d23.15)") plat
00499       write(6,*)'=== qlat ==='
00500       write(6,"(3d23.15)") qlat
00501       write(6,*)'=== ginv==='
00502       write(6,"(3f9.4)") ginv
00503       do i=1,3
00504          do j=1,3
00505             aaij=sum(qlat(:,i)*plat(:,j))
00506             bbij=sum(qlat(:,i)*ginv(j,:))
00507             if(verbose()>=40) print *,' i j aaij bbij', i,j,aaij,bbij
00508             if(i==j) then
00509                if(abs(aaij-1d0) >1d-10) call rx( 'bug 1 qg4gw')
00510                if(abs(bbij-1d0) >1d-10) call rx( 'bug 2 qg4gw')
00511             else
00512                if(abs(aaij) >1d-10) call rx( 'bug 3 qg4gw')
00513                if(abs(bbij) >1d-10) call rx( 'bug 4 qg4gw')
00514             endif
00515          enddo
00516       enddo
00517       mtet=(/1,1,1/)
00518       call getkeyvalue("GWinput","multitet",mtet,3,default=(/1,1,1/))
00519       if(sum(abs(mtet))<3) then
00520          print *, ' we use regular meshing for tetrahedron scheme '
00521       endif
00522 !! getbzdata1 allocate data in m_get_bzdata1, use at the begining in this routin.
00523       print *
00524       print *,'goto getbzdata1...'
00525       call cputid(0)
00526       call getbzdata1(bzcase,plat,qlat,ginv,n1q,n2q,n3q
00527      &     ,symops,ngrp,tetrai,tetraf,mtet) !all are inputs.
00528       dq_ = -matmul(qlat(1:3,1:3),(/.5d0/n1q,.5d0/n2q,.5d0/n3q/) )
00529       write(6,'(" dq_=",3f9.4)')dq_
00530 !! Write BZDATA
00531       ifbz = 6661
00532       open (ifbz, file='BZDATA')
00533       write(ifbz,"(10i10)")  nqbz,nqibz, nqbzw, ntetf, nteti,ngrp,nqibz_r
00534       write(ifbz,"(10i10)")  n1q,n2q,n3q
00535       print *,' writing BZDATA...'
00536       call cputid(0)
00537       call rwbzdata(ifbz,-1,
00538      &     ngrp,qlat, ginv, qbasmc,
00539      i     qbz, wbz, nstbz,  nqbz,
00540      i     qibz,wibz, nstar,irk,  nqibz,
00541      i     idtetf, ntetf, qbzw,ib1bz, nqbzw,
00542      i     idteti, nteti,dq_, qibz_r,nqibz_r)
00543       close(ifbz)
00544 !! Write QIBZ
00545       write(6,*)' qibz are written in QIBZ file...'
00546       ifiqibz = 6661
00547       open (ifiqibz, file='QIBZ') !write q-points in IBZ.
00548       write(ifiqibz,"(i10)") nqibz
00549       do iqibz = 1,nqibz
00550          write(ifiqibz,"(3d24.16,3x,d24.16)") qibz(1:3,iqibz),wibz(iqibz)
00551       enddo
```

```
00552       close(ifiqibz)
00553 !! Write QBZ
00554       ifiqbz = 6661
00555       open (ifiqbz, file='QBZ') !write q-points in IBZ.
00556       write(ifiqbz,"(i10)") nqbz
00557       do iqbz = 1,nqbz
00558          write(ifiqbz,"(3d24.16,3x,d24.16)") qbz(1:3,iqbz)
00559       enddo
00560       close(ifiqbz)
00561 !!  Write to file KPNTin1BZ
00562       ifkpt = 335
00563       open(ifkpt,file='KPTin1BZ.mkqg.chk')
00564       write(ifkpt,*)"  qbz --> shoten(qbz)"
00565       do      i1 = 1,nqbz
00566          call shorbz(qbz(1,i1),qp,qlat,plat)
00567          write (ifkpt,"(1x,i7,4f10.5,'   ',3f10.5)")
00568      &        i1,qbz(1,i1),qbz(2,i1),qbz(3,i1),wbz(i1),qp
00569       end do
00570       close (ifkpt)
00571       write(6,*) ' --- TOTAL num of q =',nqbz
00572       write(6,*)
00573       write(6,"( '  ngrp  = ',i3)")ngrp
00574       write(6,'("  qibz=",i6,3f12.5)')(i,qibz(1:3,i),i=1,min(10,nqibz))
00575       write(6,*)" ... QIBZ is written in QIBZ file ..."
00576 !!
00577       call getkeyvalue("GWinput","alpha_OffG",alp,default=-1d60)
00578       alpv(:)=alp
00579       if(alp==-1d60) then
00580        call getkeyvalue("GWinput","alpha_OffG_vec",alpv,3,default=(/-1d50,0d0,0d0/))
00581        if(alpv(1)==-1d50) then
00582          call rx( ' mkqg: No alpha_offG nor alpha_offG_vec given in GWinput')
00583        endif
00584       endif
00585       print *
00586       print *,' alpv=',alpv
00587       print *
00588       alpm = minval(alpv)
00589       if(alpm<=0d0) call rx( 'alpha_offG or alpha_offG_vec <=0')
00590 !! === Large if start. ===
00591       if(iq0pin==1) then   ! --- get q0x (offsetted q=0 point) ----------------------
00592 c  QpGcut = 15d0/alpm  !a.u. !exp( -  alp * QpGcut) !alp * QpGcut = 10
00593 c  QpGcut = sqrt(25d0/alpm) !a.u. !exp( -  alp * QpGcut**2) !alp * QpGcut**2 = 22
00594 c  QpGcut = sqrt(100d0/alp)
00595 c  QpGcut = sqrt(150d0/alp)
00596 c  QpGcut = sqrt(300d0/alp)
00597          qpgcut = sqrt(25d0/alpm) !a.u. !exp( -alp*QpGcut**2) !alp * QpGcut**2 = 22
00598          allocate( ngcx(nqbz) )
00599          ngcx=1
00600          do iq = 1, nqbz
00601             q   = qbz(1:3,iq)
00602             call getgv2(alat,plat,qlat,q, qpgcut, 1, ngcx(iq),  dummyia)
00603          enddo
00604          ngcxx = maxval(ngcx)
00605          allocate( ngvect(3,ngcxx,nqbz) )
00606          print *,' goto getgv2: ngcxx=',ngcxx
00607          do iq = 1, nqbz
00608             q  = qbz(1:3,iq)
00609             call getgv2( alat,plat,qlat, q, qpgcut, 2,
00610      &           ngcx(iq), ngvect(1:3,1:ngcx(iq),iq) )
00611          enddo
00612 !! all inputs
00613          call getq0p(newoffsetg,alat,plat,qlat,n1q,n2q,n3q,alp,alpv, !apr2015
00614      i    ngcxx,ngcx,nqbz,nqibz,nstbz,qbz,qibz,symops,ngrp,ngvect)
00615          open (1101,file='Q0P')
00616          write(1101,"(2i5,' !nq0i iq0pin' )") nq0i,iq0pin
00617          write(1101,"(d24.16,3x, 3d24.16)" ) (wt(i),q0i(1:3,i),i=1,nq0i)
00618          close(1101)
00619       elseif(iq0pin==2) then
00620          call getkeyvalue("GWinput","QforEPSIBZ",ibzqq,default=.false.)
00621          if(ibzqq) then
00622             write(6,*)'=== Find QforEPSIBZ=on === '
00623             nq0i= nqibz
00624             allocate( q0i(3,nq0i) )
00625             q0i = qibz
00626          else
00627             write(6,*)'==== Readin <QforEPS>or<QforEPS> in GWinput === '
00628             call getkeyvalue("GWinput","<QforEPS>", unit=ifinin,status=nq0i00,errstop='off')
00629             nq0i00 =max(nq0i00,0)
00630             if(nq0i00>0) close(ifinin)
00631             print *,' end of reaing QforEPS nq0i00',nq0i00,ifinin
00632
00633             call getkeyvalue("GWinput","<QforEPSL>",unit=ifinin,status=nq0i0,errstop='off')
00634             nq0i0  =max(nq0i0,0)
00635             print *,' end of reaing QforEPSL nq0i0',nq0i0,ifinin
00636             if(nq0i0>0) then
00637                allocate( ndiv(nq0i0) )
00638                do i=1,nq0i0
```

```
00639                  read(ifinin,*) qdum(1:6), ndiv(i)
00640               enddo
00641               nq0i = nq0i00 + sum(ndiv)
00642               close(ifinin)
00643            else
00644               nq0i = nq0i00
00645            endif
00646            if(nq0i <=0) call rx( 'There are neither <QforEPS> nor <QforEPS>.')
00647            allocate( q0i(3,nq0i) )
00648            print *,' nq0i=',nq0i
00649            if(nq0i00>0) then
00650               call getkeyvalue("GWinput","<QforEPS>",unit=ifinin,status=nq0i00)
00651               do i=1,nq0i00
00652                  read (ifinin,*) q0i(1:3,i)
00653                  write (6,"('<QforEPS> ' 3f12.8)") q0i(:,i)
00654               enddo
00655               close(ifinin)    !25jan2006
00656            endif
00657            if(nq0i0>0) then
00658               call getkeyvalue("GWinput","<QforEPSL>",unit=ifinin,status=nq0i0)
00659               allocate( qmin(3,nq0i0), qmax(3,nq0i0) )
00660               do i=1, nq0i0
00661               read(ifinin,*)qmin(:,i), qmax(:,i), ndiv(i)
00662               write(6,"('<QforEPSL>',3f12.8,2x,3f12.8,i5)")qmin(:,i),qmax(:,i),ndiv(i)
00663               enddo
00664               close(ifinin)
00665               ni = nq0i00
00666               do il=1, nq0i0
00667                  do i=1, ndiv(il)
00668                     q0i(:,i+ni)= qmin(:,il)+ (qmax(:,il)-qmin(:,il))/ndiv(il) * i
00669                  enddo
00670                  ni= ni + ndiv(il)
00671               enddo
00672               deallocate(qmin,qmax,ndiv)
00673            endif
00674         endif
00675         allocate( wt(nq0i) )
00676         wt = 0d0
00677         open (1101,file='Q0P')
00678         write(1101,"(2i5,a)") nq0i,iq0pin, " !nq0i iq0pin ---"//
00679      &        "This is readin Q0P from GWinput <QforEPS> ---"
00680         write(1101,"(d24.16,3x, 3d24.16)") (wt(i),q0i(1:3,i),i=1,nq0i)
00681         close(1101)
00682      endif
00683      print *,' end fo writing Q0P'
00684      call cputid(0)
00685
00686 !! Timereversal may require q0i. Anyway, qreduce0 will reduce the number of q points by symops.
00687      if(.not.timereversal().and.iq0pin==1) then
00688         write(6,*)" timereversal==off : add -Q0P points"
00689         do iq=1,nq0i
00690            q0i(:,iq+nq0i)= -q0i(:,iq)
00691         enddo
00692         nq0i=nq0i*2
00693      endif
00694
00695 !! === AnyQ mechanism. === q0i is extended. nq0i/=nq0itrue
00696      call getkeyvalue("GWinput","AnyQ",anyq,default=.false.)
00697      if(anyq.and.iq0pin==1) then
00698         print *,'AnyQ (read <QPNT> section =T'
00699 !!     read q-points and states
00700         call getkeyvalue("GWinput","<QPNT>",unit=ifqpnt,status=ret)
00701         call readx(ifqpnt,10)
00702         call readx(ifqpnt,100)
00703         call readx(ifqpnt,100)
00704         read (ifqpnt,*) nany
00705         print *,'  nany=',nany
00706         allocate(qany(3,nany))
00707         do ix=1,nany
00708            read (ifqpnt,*) i, qany(:,ix)
00709            write(6,'(i3,3f13.6)') ix,qany(:,ix)
00710         enddo
00711         nany =ix-1
00712         write(6,*)" Anyq mode: nany=",nany
00713         allocate(qsave(3,nq0i+nany))
00714         qsave(:,    1 :nq0i)     = q0i(:,1:nq0i)
00715         qsave(:,nq0i+1:nq0i+nany)= qany(:,1:nany)
00716         nq0itrue=nq0i !nov2015
00717         nq0i = nq0i+nany
00718         deallocate(q0i)
00719         allocate(q0i(3,nq0i))
00720         q0i=qsave
00721         deallocate(qsave)
00722         close(ifqpnt)
00723      else
00724         nq0itrue=nq0i !nov2015
00725      endif
```

```
00726
00727 !! Four kinds of mesh points. Q0P means offset Gamma (slightly different from Gamma).
00728 !! Which we need?
00729 !! 1. regular
00730 !! 2. offregular (not including Gamma)
00731 !! 3. regular    + Q0P
00732 !! 4. offregular + Q0P
00733       if(iq0pin==2) then        !this is just for dielectric case
00734          regmesh = qbzreg()
00735       else
00736          regmesh = .true.
00737       endif
00738       regmeshg = qbzreg()       !Gamma mesh based on regular mesh
00739       offmesh =  .not.qbzreg()  !we fix bzcase=1 now. apr2015.
00740       offmeshg = .not.qbzreg()  !Gamma mesh based on off-regular mesh
00741       print *,' regmesh offmeshg=', regmesh,regmeshg !regular,     regular+shifted
00742       print *,' offmesh offmeshg=', offmesh,offmeshg !offregmesh, offregular+shifted
00743 !!
00744 c      if(regmesh) nqnum = nqnum + nqbz
00745 c      if(offmesh) nqnum = nqnum + nqbz
00746
00747
00748 !!  We check wether all q0i \in qbz or not. <--- Takao think this block is not necessary now.
00749       nqnum = nqbz
00750       allocate( qq(1:3,nqnum),irr(nqnum) )
00751       qq(1:3,1:nqbz) = qbz(1:3,1:nqbz)
00752       do iq0i=1,nq0i
00753          do iq=1,nqbz
00754             if(sum(abs(q0i(:,iq0i)-qq(:,iq)))<tolq) goto 2112
00755             call rangedq( matmul(ginv,q0i(:,iq0i)-qq(:,iq)), qx)
00756             if(sum(abs(qx))< tolq) goto 2112
00757          enddo
00758          goto 2111
00759  2112    continue
00760          qq(:,iq) = q0i(:,iq0i) !replaced with equivalent q0i.
00761       enddo
00762       print *,' --- We find all q0i in qbz. Skip qreduce.'
00763       goto 2001
00764  2111 continue
00765
00766
00767 !!  --- Start accumulate all required q points
00768       deallocate(qq,irr)
00769       nqnum = nqbz  + nqbz*nq0i
00770       nqnum = nqnum + 1         !add Gamma
00771       nqnum = nqnum + nq0i      !add Gamma + q0i
00772       allocate( qq(1:3,nqnum),irr(nqnum) )
00773       ix = 0
00774       if(regmesh) then
00775          qq(1:3,1:nqbz) = qbz(1:3,1:nqbz)
00776          ix = ix+ nqbz
00777       endif
00778 !!  Off Regular mesh.
00779       if(offmesh) then
00780          do iq = 1, nqbz
00781             ix = ix+1
00782             qq(1:3,ix) = qbz(1:3,iq) - dq_
00783          enddo
00784       endif
00785       nnn   =  ix               !n1q*n2q*n3q!      if(offmesh) nnn = 2*n1q*n2q*n3q
00786       print *,' nnn=',nnn       !This is the number to calcualte Vxc
00787 !!  Shifted mesh
00788       dq__=0d0
00789       if(regmeshg) then
00790          do iq00 = 1, nq0i
00791             do iq   = 1, nqbz
00792                ix = ix+1
00793                qq(1:3,ix) = qbz(1:3,iq) +  q0i(1:3,iq00)
00794             enddo
00795          enddo
00796       endif
00797       if(offmeshg) then
00798          dq__=dq_
00799          do iq00 = 1, nq0i
00800             do iq   = 1, nqbz
00801                ix = ix+1
00802                qq(1:3,ix) = qbz(1:3,iq) - dq_ + q0i(1:3,iq00)
00803             enddo
00804          enddo
00805       endif
00806 !! Add offset Gamma and Gamma point (these can be removed by qreduce and q0irre)
00807       do iq00 = 1, nq0i
00808          ix = ix+1
00809          qq(1:3,ix) = q0i(1:3,iq00)
00810       enddo
00811       ix=ix+1
00812       qq(1:3,ix)=0d0
```

```
00813
00814
00815 !! (not so much used) Get qqm; q point for eigenvalues.
00816 !! Saved to Qmtet. Not so much used now... We need check when we use this...
00817       if(sum(abs(mtet))/=3) then
00818          nqnumm= nqbzm * (nq0i+1)
00819          allocate( qqm(1:3,nqnumm) )
00820          ix=0
00821          do iq00 = 1, 1 + nq0i
00822             do iq   = 1, nqbzm
00823                ix = ix+1
00824                if(iq00==1) then
00825                   qqm(1:3,ix) = qbzm(1:3,iq)
00826                else
00827                   qqm(1:3,ix) = q0i(1:3,iq00-1) + qbzm(1:3,iq)
00828                endif
00829             enddo
00830          enddo
00831          ifiqmtet=501
00832          open(ifiqmtet, file='Qmtet')
00833          write(ifiqmtet,"(i10)") nqnumm
00834          do iq=1,nqnumm
00835             write(ifiqmtet,"(3d24.16)") qqm(1:3,iq)
00836          enddo
00837          close(ifiqmtet)
00838          deallocate(qqm)
00839       endif
00840
00841
00842 !! Remove equivalent q point for translational symmetry
00843       if( qreduce0 ) then
00844          print *,'goto qqsave nq0i nqnum',nq0i,nqnum
00845          call cputid(0)
00846          nmax=nq0i+nqnum
00847          allocate(qsave(3,nmax)) !,qsavel(nmax))
00848          imx=0
00849          if(iq0pin /=1) then
00850             do iq=1,nq0i
00851                call qqsave(q0i(1:3,iq),nmax,ginv,qsave,imx)
00852             enddo
00853          endif
00854          do iq=1,nqnum
00855             call qqsave(qq(1:3,iq),nmax,ginv,qsave,imx)
00856          enddo
00857          nqnum = imx
00858          qq(:,1:imx)=qsave(:,1:imx)
00859          deallocate(qsave)
00860       endif
00861
00862 !! ------------------------------------------
00863  2001 continue
00864 !! ------------------------------------------
00865 !! Here we get all requied q points. We do reduce them by space group symmetry.
00866       if(allocated(wt0)) deallocate(wt0)
00867       allocate(wt0(nqnum+nq0i),qi(3,nqnum+nq0i),wti(nqnum+nq0i))
00868       wt0=1d0
00869 !! Set irreducible k-point flag. irr=1 for (irredusible point) flag, otherwise =0.
00870 !! irr(iq)=1 for irreducile qq(:,iq), iq=1,nqnum
00871       call q0irre(qibz,nqibz,qq,wt0,nqnum,symops,ngrp, qi,nqi,wti,plat,.true.,0,irr)
00872 !! nqnum is the finally obtained number of q points.
00873       allocate(ngpn(nqnum), ngcn(nqnum))
00874       if(debug) write(6,*) ' --- q vector in 1st BZ + Q0P shift. ngp ---'
00875       imx=0
00876       imxc=0
00877       do iq = 1, nqnum
00878          q = qq(1:3,iq)
00879          qxx=q
00880          if(iq0pin==1) then !use qxx on regular mesh points if q is on regular+Q0P(true).
00881             do iqbz=1,nqbz
00882             do i=1,nq0itrue ! nq0itrue/=nq0i for anyq=F nov2015
00883                if(sum(abs(qbz(1:3,iqbz)-dq__+ q0i(:,i)-qxx))<1d-6) then
00884                   qxx=qbz(1:3,iqbz)
00885                   exit
00886                endif
00887             enddo
00888             enddo
00889          endif
00890          ngpn(iq)=1
00891 !! get G vector for |q+G| < QpGcut_psi
00892          call getgv2(alat,plat,qlat, qxx, qpgcut_psi,1,ngpn(iq),imx11) !imx11 !nov2015
00893          imx0=imx11(1,1)
00894          if(imx0>imx) imx=imx0
00895          ngcn(iq)=1
00896 !! get G vector for |q+G| < QpGcut_cou
00897          call getgv2(alat,plat,qlat, qxx, qpgcut_cou,1,ngcn(iq),imx11) !imx11 to avoid warning.
00898          imx0c=imx11(1,1)
00899          if(imx0c>imxc) imxc=imx0c
```

```
00900          if(verbose()>150)write(6,'(3f12.5,3x,2i4)') q ,ngpn(iq) !,ngcn(iq,iq00)
00901          if(verbose()>150)write(6,'(3f12.5,3x,2i4)') q ,ngcn(iq) !,ngcn(iq,iq00)
00902       enddo
00903 !! ------- Write q+G vectors -----------
00904       ngpmx = maxval(ngpn)
00905       ngcmx = maxval(ngcn)
00906       write(ifiqg ) nqnum,ngpmx,qpgcut_psi,nqbz,nqi,imx,nqibz
00907       write(ifiqgc) nqnum,ngcmx,qpgcut_cou,nqbz,nqi,imxc
00908 !! :nqi:   The number of irreducible points (including irr. of offset points). irr=1.
00909 !! ::       We calcualte eigenfunction and Vxc for these points.
00910 !! :nqnum: total number of q points.
00911 !! :imx:   to allocate ngvecprev as follows.
00912       print *,' number of irrecucible points nqi=',nqi
00913       print *,' imx nqnum=',imx,nqnum
00914       write(6,*) ' --- Max number of G for psi =',ngpmx
00915       write(6,*) ' --- Max number of G for Cou =',ngcmx
00916       allocate( ngvecprev(-imx:imx,-imx:imx,-imx:imx) )       !inverse mapping table for ngvecp (psi)
00917       allocate( ngveccrev(-imxc:imxc,-imxc:imxc,-imxc:imxc) ) !inverse mapping table for ngvecc (cou)
00918       ngvecprev=9999
00919       ngveccrev=9999
00920       do iq = 1, nqnum
00921          q = qq(1:3,iq)
00922          qxx=q
00923          q0pf=''
00924          do iqbz=1,nqbz  !use qxx on regular mesh points if q is on regular+Q0P(true).
00925          do i=1,nq0itrue !nq0itrue/=nq0i for anyq=F nov2015
00926             if(sum(abs(qbz(1:3,iqbz)-dq__+ q0i(:,i)-qxx))<1d-6) then
00927                if(sum(abs(q0i(:,i)-qxx))<1d-6) then
00928                  q0pf=' <--Q0P  '    ! offset Gamma points
00929                else
00930                  q0pf=' <--Q0P+R'    ! offset Gamma points-shifted nov2015
00931                endif
00932                if(iq0pin==1) then
00933                   qxx=qbz(1:3,iqbz)
00934                endif
00935                exit
00936             endif
00937          enddo
00938          enddo
00939          ngp = ngpn(iq)
00940          ngc = ngcn(iq)
00941          write(6,"(' iq=',i8,' q=',3f7.3,' ngp ngc= ',2i6,' irr.=',i2,a)") !irr=1 is irreducible k points.
00942      &        iq, q, ngp, ngc, irr(iq),trim(q0pf)
00943          allocate( ngvecp(3,max(ngp,1)), ngvecc(3,max(ngc,1)) )
00944          call getgv2(alat,plat,qlat, qxx, qpgcut_psi, 2, ngp,  ngvecp) ! for eigenfunctions (psi)
00945          call getgv2(alat,plat,qlat, qxx, qpgcut_cou, 2, ngc,  ngvecc) ! for Coulomb        (cou)
00946          write (ifiqg) q, ngp, irr(iq)
00947          do ig = 1,ngp
00948             nnn3 = ngvecp(1:3, ig)
00949             ngvecprev( nnn3(1), nnn3(2),nnn3(3)) = ig
00950          enddo
00951          write (ifiqg)  ngvecp,ngvecprev !ngvecprev is added on mar2012takao
00952          do ig = 1,ngc
00953             nnn3 = ngvecc(1:3, ig)
00954             ngveccrev( nnn3(1), nnn3(2),nnn3(3)) = ig
00955          enddo
00956          write (ifiqgc) q, ngc
00957          write (ifiqgc) ngvecc,ngveccrev
00958          deallocate(ngvecp,ngvecc)
00959       enddo
00960       deallocate(ngpn,ngcn,ngvecprev,ngveccrev)
00961       if(iq0pin==1) deallocate(ngvect)
00962       if(debug) print *,'--- end of mkqg ---'
00963       end
00964
00965 !!###########################################################################
00966       double precision function tripl(a,b,c)
00967 !! == tripl (determinant of 3x3 matrix) ==
00968       implicit real*8 (a-h,p-z), integer(o)
00969       dimension a(0:2),b(0:2),c(0:2)
00970 c     tripl=a(1)*b(2)*c(3)+a(2)*b(3)*c(1)+a(3)*b(1)*c(2)
00971 c     .     -a(3)*b(2)*c(1)-a(2)*b(1)*c(3)-a(1)*b(3)*c(2)
00972 c     ... g77 needs this rewriting
00973       tmp = 0.d0
00974       do i = 0,2
00975          j = mod(i + 1, 3)
00976          k = mod(i + 2, 3)
00977          tmp = tmp + a(i) * (b(j)*c(k) - b(k)*c(j))
00978       enddo
00979       tripl = tmp
00980       end
00981
00982 !!###########################################################################
00983       integer(4) function llxxx(ilm)
00984       integer(4),parameter :: lmx=50
00985       integer(4),save:: lla((lmx+1)**2)
00986       logical:: init=.true.
```

```
00987       if(ilm>(lmx+1)**2) call rx( 'll: ilm too large')
00988       if(init) then
00989          do l=0,lmx
00990             lini= l**2 + 1
00991             lend=(l+1)**2
00992             lla(lini:lend)=l
00993          enddo
00994       endif
00995       llxxx = lla(ilm)
00996       return
00997       end
```

4.21 gwsrc/readqg.F File Reference

Data Types

- module m_readqg

  Return QGcou and QGpsi ===.

Functions/Subroutines

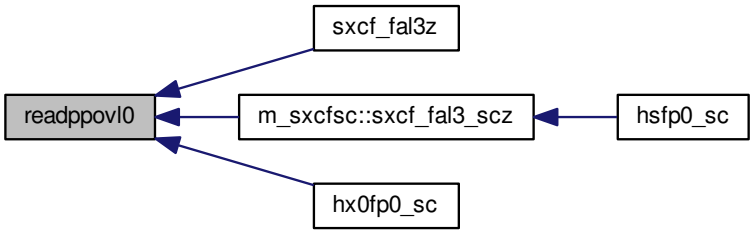
- subroutine readppovl0 (q, ngc, ppovl)

4.21.1 Function/Subroutine Documentation

4.21.1.1 subroutine readppovl0 (real(8), dimension(3), intent(in) *q,* integer(4), intent(in) *ngc,* complex(8), dimension(ngc,ngc), intent(out) *ppovl*)

Definition at line 1 of file readqg.F.

Here is the caller graph for this function:



4.22 readqg.F

```
00001       subroutine readppovl0(q,ngc,ppovl)
00002       implicit none
00003       real(8), intent(in) :: q(3)
00004       integer(4), intent(in) :: ngc
00005       complex(8), intent(out) :: ppovl(ngc,ngc)
00006
00007       integer(4):: ngc_r,ippovl0
00008       real(8):: qx(3)
00009       ippovl0=2301
00010       open(ippovl0,file='PPOVL0',form='unformatted')
00011       do
00012         read(ippovl0) qx,ngc_r
```

```
00013         if(sum(abs(qx-q))<1d-6) then
00014           if(ngc_r/=ngc) call rx( 'readin ppovl: ngc_r/=ngc')
00015           read(ippovl0) ppovl
00016           exit
00017         endif
00018       enddo
00019       close(ippovl0)
00020       end
00021
00022 !> Return QGcou and QGpsi ===
00023       module m_readqg
00024       implicit none
00025       real(8),allocatable,private,target:: qc(:,:),qp(:,:)
00026       logical,private:: init(2)=.true.
00027       real(8),private:: QpGcut_cou, QpGcut_psi
00028       integer(4),private,target::   nqnumc,nqnump,ngcmx,ngpmx
00029       integer(4),allocatable,private:: ngvecp(:,:,:),ngp(:),ngvecc(:,:,:),ngc(:)
00030       integer,pointer,private::nqtt
00031       real(8),pointer,private::qtt(:,:)
00032       real(8),private:: epsd=1d-7
00033       integer,private,pointer:: nkey(:),kk1(:),kk2(:),kk3(:),iqkkk(:,:,:)
00034       integer,target,private :: nkeyp(3),nkeyc(3)
00035       integer,target,allocatable,private:: keyp(:,:),kk1p(:),kk2p(:),kk3p(:),iqkkkp(:,:,:)
00036       integer,target,allocatable,private:: keyc(:,:),kk1c(:),kk2c(:),kk3c(:),iqkkkc(:,:,:)
00037       real(8),private:: ginv_(3,3)
00038       contains
00039 c-------------------------------
00040       subroutine readngmx(key,ngmx)
00041 c- get ngcmx or mgpmx
00042       implicit none
00043       integer(4):: ngmx,ifiqg=4052
00044       character*(*) key
00045       if    (key=='QGpsi') then
00046         open(ifiqg, file='QGpsi',form='unformatted')
00047         read(ifiqg) nqnump, ngpmx, qpgcut_psi
00048         ngmx=ngpmx
00049       elseif(key=='QGcou') then
00050         open(ifiqg, file='QGcou',form='unformatted')
00051         read(ifiqg) nqnumc, ngcmx, qpgcut_cou
00052         ngmx=ngcmx
00053       else
00054         call rx( "readngmx: key is not QGpsi QGcou")
00055       endif
00056       close(ifiqg)
00057       end subroutine
00058
00059 !> Get ngv and ngvec(3,ngv) for given qin(3)
00060 !! key=='QGcou' or 'QGpsi'
00061       subroutine readqg(key,qin,ginv,  qu,ngv,ngvec)
00062       implicit none
00063       character*(*), intent(in) :: key
00064       real(8), intent(in) :: qin(3),ginv(3,3)
00065       real(8), intent(out) :: qu(3)
00066       integer(4), intent(out) :: ngv, ngvec(3,*)
00067
00068       integer(4):: ifi, iq,verbose
00069       if    (key=='QGpsi') then
00070         ifi=1
00071         if(verbose()>=80) write (6,"(' readqg psi: qin=',3f8.3,i5)") qin
00072       elseif(key=='QGcou') then
00073         ifi=2
00074         if(verbose()>=80) write (6,"(' readqg cou: qin=',3f8.3,i5)") qin
00075       else
00076         call rx( "readqg: wrongkey")
00077       endif
00078       if(init(ifi)) then
00079         call init_readqg(ifi,ginv)
00080         init(ifi)=.false.
00081       endif
00082       if(verbose()>=40) write(6,*)'end of init_readqg'
00083       call iqindx2qg(qin,ifi, iq,qu)
00084       if(ifi==1) then
00085         ngv  = ngp(iq)
00086         ngvec(1:3,1:ngv) = ngvecp(1:3,1:ngv,iq)
00087         return
00088       elseif(ifi==2) then
00089         ngv  = ngc(iq)
00090         ngvec(1:3,1:ngv) = ngvecc(1:3,1:ngv,iq)
00091         return
00092       endif
00093       call rx( "readqg: can not find QGpsi or QPcou for given q")
00094       end subroutine readqg
00095
00096 !> Get ngv
00097 !! key=='QGcou' or 'QGpsi'
00098       subroutine readqg0(key,qin,ginv,  qu,ngv)
00099       implicit none
```

```
00100       character*(*), intent(in) :: key
00101       integer(4), intent(out) :: ngv
00102       real(8), intent(in):: qin(3),ginv(3,3)
00103       real(8), intent(out):: qu(3)
00104 
00105       integer(4):: ifi, iq,verbose
00106       if    (key=='QGpsi') then
00107         ifi=1
00108         if(verbose()>=80) write (6,"('readqg0 psi: qin=',3f8.3,i5)") qin
00109       elseif(key=='QGcou') then
00110         ifi=2
00111         if(verbose()>=80) write (6,"('readqg0 cou: qin=',3f8.3,i5)") qin
00112       else
00113         call rx( "readqg: wrongkey")
00114       endif
00115       if(init(ifi)) then
00116         call init_readqg(ifi,ginv)
00117         init(ifi)=.false.
00118       endif
00119       call iqindx2qg(qin,ifi, iq,qu)
00120       if(ifi==1) then
00121         ngv  = ngp(iq)
00122         if(verbose()>=80) write(6,*)'ngp=',ngv
00123       elseif(ifi==2) then
00124         ngv  = ngc(iq)
00125         if(verbose()>=80) write(6,*)'ngc=',ngv
00126       endif
00127       return
00128       call rx( "readqg0: can not find QGpsi or QPcou for given q")
00129       end subroutine
00130 
00131 !> initialization. readin QGpsi or QGcou.
00132       subroutine init_readqg(ifi,ginv)
00133       implicit none
00134       integer(4), intent(in) :: ifi
00135       real(8), intent(in) :: ginv(3,3)
00136 
00137       integer(4):: ifiqg,iq,verbose
00138       real(8)::qq(3)
00139       real(8),allocatable:: qxx(:,:)
00140       integer:: isig,i,ix,kkk,kkk3(3),ik1,ik2,ik3,ik
00141       integer,allocatable:: ieord(:),key(:,:)
00142       ginv_=ginv
00143       write(6,*)' init_readqg ifi=',ifi
00144       ifiqg=4052
00145       if(ifi==1) then
00146         open(ifiqg, file='QGpsi',form='unformatted')
00147         read(ifiqg) nqnump, ngpmx, qpgcut_psi
00148         if(verbose()>49)
00149      &     write (6,"('init_readqg ngnumc ngcmx QpGcut_psi=',2i5,f8.3)")
00150      &     nqnump, ngpmx, qpgcut_psi
00151         allocate(ngvecp(3,ngpmx,nqnump),qp(3,nqnump),ngp(nqnump))
00152         do iq=1, nqnump
00153           read (ifiqg) qp(1:3,iq), ngp(iq)
00154           read (ifiqg) ngvecp(1:3,1:ngp(iq),iq)
00155           if(verbose()>40)
00156      &     write (6,"('init_readqg psi qp ngp =',3f8.3,i5)") qp(1:3,iq),ngp(iq)
00157         enddo
00158       elseif(ifi==2) then
00159         open(ifiqg, file='QGcou',form='unformatted')
00160         read(ifiqg) nqnumc, ngcmx, qpgcut_cou
00161 c         write (6,"('init_readqg ngnumc ngcmx QpGcut_cou=',2i5,f8.3)")
00162 c     &     nqnumc, ngcmx, QpGcut_cou
00163         allocate(ngvecc(3,ngcmx,nqnumc),qc(3,nqnumc),ngc(nqnumc))
00164         do iq=1, nqnumc
00165           read(ifiqg) qc(1:3,iq), ngc(iq)
00166           if(verbose()>40)  write (6,"('init_readqg cou  qc ngc =',3f8.3,i5)") qc(1:3,iq), ngc(iq)
00167           read (ifiqg) ngvecc(1:3,1:ngc(iq),iq)
00168         enddo
00169       endif
00170       close(ifiqg)
00171 
00172 !! === mapping of qtt ===
00173 !! nkey, kk1,kk2,kk3, iqkkk are to get iqindx.
00174 !!  q --> call rangedq(matmul(ginv,q), qx) ---> n= (qx+0.5*epsd)/epsd
00175 !!       --->  ik1,ik2,ik3= tabkk(kkk,iqk,nkey) ---> iqkkk(ik1,ik2,ik3)
00176       if(ifi==1) then
00177          nqtt => nqnump
00178          qtt  => qp
00179          nkey => nkeyp
00180       elseif(ifi==2) then
00181          nqtt => nqnumc
00182          qtt  => qc
00183          nkey => nkeyc
00184       endif
00185 !! followings are the same as codes in readeigen.F
00186       allocate(ieord(nqtt))
```

```
00187       allocate(key(3,0:nqtt),qxx(3,nqtt))
00188       key(:,0)=0 !dummy
00189       key=-99999
00190       do iq=1,nqtt
00191          call rangedq(matmul(ginv_,qtt(:,iq)), qxx(:,iq))
00192       enddo
00193 !! get key and nkey for each ix.
00194       do ix =1,3
00195          call sortea(qxx(ix,:),ieord,nqtt,isig)
00196          ik=0
00197          do i=1,nqtt
00198             kkk=(qxx(ix,ieord(i))+0.5*epsd)/epsd  !kkk is digitized by 1/epsd
00199             if(i==1.or.key(ix,ik)<kkk) then
00200                ik=ik+1
00201                key(ix,ik) = kkk
00202 c               write(6,*)ix, ik,i, key(ix,ik), qxx(ix,ieord(i))
00203             elseif (key(ix,ik)>kkk) then
00204                write(6,*)ix, ik,i, key(ix,ik), qxx(ix,ieord(i))
00205 Cstop2rx 2013.08.09 kino               stop 'iqindx: bug not sorted well'
00206                call rx( 'iqindx: bug not sorted well')
00207             endif
00208          enddo
00209          nkey(ix)=ik
00210       enddo
00211       deallocate(ieord)
00212 !!  key is reallocated. inverse mattping, iqkkk
00213       if(ifi==1) then
00214         allocate( kk1p(nkey(1)),kk2p(nkey(2)),kk3p(nkey(3)) )
00215         allocate( iqkkkp(nkey(1),nkey(2),nkey(3)) )
00216          iqkkk => iqkkkp
00217          kk1 =>kk1p
00218          kk2 =>kk2p
00219          kk3 =>kk3p
00220       elseif(ifi==2) then
00221         allocate( kk1c(nkey(1)),kk2c(nkey(2)),kk3c(nkey(3)) )
00222         allocate( iqkkkc(nkey(1),nkey(2),nkey(3)) )
00223          iqkkk => iqkkkc
00224          kk1 =>kk1c
00225          kk2 =>kk2c
00226          kk3 =>kk3c
00227       endif
00228
00229       kk1(:) = key(1,1:nkey(1))
00230       kk2(:) = key(2,1:nkey(2))
00231       kk3(:) = key(3,1:nkey(3))
00232       deallocate(key)
00233 c      write(6,*)' ifi init_qqq nqtt=',ifi,nqtt
00234 c      write(6,*)'kkk3=',kkk3
00235 c      write(6,*)'nkey=',nkey
00236 c      write(6,*)'kk1=',kk1
00237       do i=1,nqtt
00238          kkk3= (qxx(:,i)+0.5*epsd)/epsd !kkk is digitized by 1/epsd
00239          call tabkk(kkk3(1), kk1,nkey(1), ik1)
00240          call tabkk(kkk3(2), kk2,nkey(2), ik2)
00241          call tabkk(kkk3(3), kk3,nkey(3), ik3)
00242          iqkkk(ik1,ik2,ik3)=i
00243 c      write(6,*)' ik1,ik2,ik3 i=',ik1,ik2,ik3,i
00244       enddo
00245       deallocate(qxx)
00246       end subroutine init_readqg
00247 !! ---
00248       subroutine tabkk(kkin, kktable,n, nout)
00249       integer:: nout,n, kkin, kktable(n),i,mm,i1,i2
00250       i1=1
00251       i2=n
00252       if(kkin==kktable(1)) then
00253          nout=1
00254          return
00255       elseif(kkin==kktable(n)) then
00256          nout=n
00257          return
00258       endif
00259       do i=1,n
00260          mm=(i1+i2)/2
00261          if(kkin==kktable(mm)) then
00262             nout=mm
00263             return
00264          elseif(kkin>kktable(mm)) then
00265             i1=mm
00266          else
00267             i2=mm
00268          endif
00269       enddo
00270       write(6,*) i1,i2,kkin
00271       write(6,*) kktable(i1),kktable(i2)
00272       call rx( 'takk: error')
00273       end subroutine
```

```
00274
00275 c$$$c--- release to save memory area.
00276 c$$$      subroutine releaseqg_notusednow(key)
00277 c$$$      implicit none
00278 c$$$      character*(*) key
00279 c$$$      integer(4):: ifi
00280 c$$$      if    (key=='QGpsi') then
00281 c$$$        ifi=1
00282 c$$$        deallocate(qp,ngvecp)
00283 c$$$      elseif(key=='QGcou') then
00284 c$$$        ifi=2
00285 c$$$        deallocate(qc,ngvecc)
00286 c$$$      else
00287 c$$$        stop "releaseqg: in readQGcou"
00288 c$$$      endif
00289 c$$$      init(ifi)=.false.
00290 c$$$      end subroutine
00291 !!-------------------------------------------------------
00292
00293 !> Find index as q=qq(:,iq) with modulo of premitive vector.
00294 !! ginv is the inverse of plat (premitive translation vector).
00295 !! Use kk1,kk2,kk3,nkey(1:3),iqkkk to get iqindx.
00296       subroutine iqindx2qg(q,ifi, iqindx,qu)
00297       implicit none
00298       integer, intent(in):: ifi
00299       integer, intent(out):: iqindx
00300       real(8), intent(in) :: q(3)
00301       real(8), intent(out) :: qu(3)
00302
00303       integer:: i_out, iq,iqx ,kkk3(3),ik1,ik2,ik3
00304       real(8):: qx(3),qzz(3)
00305       logical::debug=.false.
00306       if(ifi==1) then
00307 c         nqtt => nqnump
00308          qtt  => qp
00309          nkey => nkeyp
00310          iqkkk => iqkkkp
00311          kk1 =>kk1p
00312          kk2 =>kk2p
00313          kk3 =>kk3p
00314       elseif(ifi==2) then
00315 c         nqtt => nqnumc
00316          qtt  => qc
00317          nkey => nkeyc
00318          iqkkk => iqkkkc
00319          kk1 =>kk1c
00320          kk2 =>kk2c
00321          kk3 =>kk3c
00322       endif
00323       if(debug) write(*,"(' iqindx2_: q=',3f20.15)") q
00324       call rangedq(matmul(ginv_,q), qzz)
00325       kkk3 = (qzz+0.5*epsd)/epsd
00326 c      write(6,*)'kkk3=',kkk3
00327 c      write(6,*)'kk1,nkey1',kk1,nkey(1)
00328 c      write(6,*)'kk2,nkey2',kk2,nkey(2)
00329 c      write(6,*)'kk3,nkey3',kk3,nkey(3)
00330       call tabkk(kkk3(1), kk1,nkey(1), ik1)
00331       call tabkk(kkk3(2), kk2,nkey(2), ik2)
00332       call tabkk(kkk3(3), kk3,nkey(3), ik3)
00333 c      write(6,*)' ik1ik2ik3=',ik1,ik2,ik3
00334       iqindx = iqkkk(ik1,ik2,ik3)
00335 c      write(6,*)'iqindx=',iqindx
00336       qu = qtt(:,iqindx)
00337       end subroutine
00338
00339 !> mini-sort routine.
00340       subroutine sortea(ea,ieaord,n,isig)
00341       real(8), intent(in) :: ea(n)
00342       integer(4), intent(inout) :: ieaord(n)
00343       integer, intent(in) :: n
00344       integer, intent(out) :: isig
00345       integer :: ix,i
00346       isig = 1
00347       do i = 1,n
00348         ieaord(i) = i
00349       enddo
00350       do ix= 2,n
00351         do i=ix,2,-1
00352           if( ea(ieaord(i-1)) >ea(ieaord(i) ) ) then
00353             call iswap(ieaord(i-1),ieaord(i))
00354             isig= -isig
00355             cycle
00356           endif
00357           exit
00358         enddo
00359       enddo
00360       end subroutine
```

```
00361       subroutine iswap(i,j)
00362       implicit none
00363       integer,intent(inout) :: i, j
00364       integer:: iwork
00365       iwork = j
00366       j = i
00367       i = iwork
00368       end subroutine
00369       end module m_readqg
```

4.23 gwsrc/sxcf_fal2.F File Reference

Functions/Subroutines

- subroutine sxcf_fal3z (kount, ixc, deltaw, shtw, qip, itq, ntq, ef, ef2, esmr, esmr2,nsp, isp,qbas, ginv,qibz, qbz, wk, nstbz, wik,nstar, irkip,freq_r, freqx, wx,dw, ecore,nlmto, nqibz, nqbz, nctot,

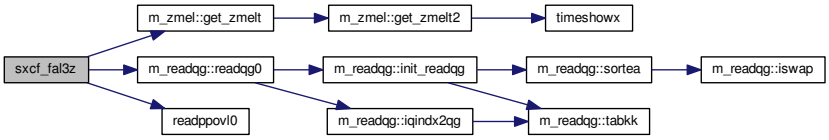
4.23.1 Function/Subroutine Documentation

4.23.1.1 subroutine sxcf_fal3z (intent(in) *kount,* integer, intent(in) *ixc,* real(8), intent(in) *deltaw,* real(8), intent(in) *shtw,* real(8), dimension(3,∗), intent(in) *qip,* intent(in) *itq,* integer, intent(in) *ntq,* real(8), intent(in) *ef,* real(8), intent(in) *ef2,* real(8), intent(in) *esmr,* real(8), intent(in) *esmr2,* integer, intent(in) *nsp,* integer, intent(in) *isp,* real(8), dimension(3∗3), intent(in) *qbas,* real(8), dimension(3∗3), intent(in) *ginv,* real(8), dimension(3,nqibz), intent(in) *qibz,* real(8), dimension(3,nqbz), intent(in) *qbz,* real(8), dimension(nqbz), intent(in) *wk,* integer(4), dimension(nqbz), intent(in) *nstbz,* real(8), dimension(nqibz), intent(in) *wik,* intent(in) *nstar,* intent(in) *irkip,* real(8), dimension(nw_i:nw) *freq_r,* real(8), dimension(niw) *freqx,* real(8), dimension(niw) *wx,* real(8) *dw,* real(8), dimension(nctot) *ecore,* integer *nlmto,* integer *nqibz,* integer *nqbz,* integer *nctot*)

z1p(j,t,t') = S[i=1,nbloch] <psi(q,t') | psi(q-rk,t) B(rk,i)> v(k)(i,j) NOTE: zmel(igb, nctot+nbmax, ntp0) —> <phi phi |igb>

Definition at line 1 of file sxcf_fal2.F.

Here is the call graph for this function:



4.24 sxcf_fal2.F

```
00001       subroutine sxcf_fal3z(kount,ixc,deltaw,shtw,qip,itq, ntq,ef,ef2,esmr,esmr2,
00002      i nsp,isp,                  !tiat,miat,
00003      i qbas,ginv,
00004      i qibz,qbz,wk,nstbz,wik,
00005      i nstar,irkip,
00006      i freq_r,freqx,wx,
00007      i dw,ecore,
00008      d nlmto,nqibz,nqbz,nctot,
00009 c     d nl,nnc,nclass,natom,mdimx,
00010      d nbloch,ngrp, nw_i,nw ,niw,niwx,nq, !nlnmx,
00011      & nblochpmx ,ngpmx,ngcmx,
00012      & wgt0,nq0i,q0i,symgg,alat, nband, ifvcfpout, !shtvg,
00013      & exchange,tote,screen,cohtest, ifexsp,
00014      i iwini,iwend,
00015      i nbmx,ebmx,
00016      i wklm,lxklm,
00017      i dwplot,
```

```
00018      o zsec,coh,exx)
00019       use m_readqg
00020       use m_readeigen,only: readeval
00021       use keyvalue,only: getkeyvalue
00022       use m_zmel,only: get_zmelt,
00023      o ppovlz, zmel,zmeltt
00024       implicit none
00025 !! TimeReversal off. when nw_i is not zero.
00026 !! Calcualte diagonal part only version of simga_ii(e_i)= <i|Re[S](e)|i>
00027 !! Similar with sxcf_fal2.sc.F
00028 Co zsec: S_ij= <i|Re[S](e)|i> where e=e_i and e_i \pm deltaw
00029 Co
00030 Cr  exchange=T : Calculate the exchange self-energy
00031 Cr           =F : Calculate correlated part of the self-energy
00032 Cr
00033 Cr
00034 Cr---- 2001 Sep. esec=omega(itp,iw). Genral iw mode for exchange =F
00035 Cr 2000 takao kotani. This sxcf is starting from sec.f F.Aryasetiawan.
00036 C-------------------------------------------------------------
00037
00038
00039 c---- original document for sce.f (correlation case) by ferdi.Aryasetiawan.
00040 c 92.02.24
00041 c 93.10.18 from sec.f modified to take into account equivalent atoms
00042 c calculates the correlated part of the self-energy SE
00043 c SEc(q,t,t') = <psi(q,t) |SEc| psi(q,t'>
00044 c SEc(r,r';w) = (i/2pi) < [w'=-inf,inf] G(r,r';w+w') Wc(r,r';w') >
00045
00046 c the zeroth order Green function
00047 c G(r,r';w)    = S[occ]   psi(kn,r) psi(kn,r')^* /(w-e(kn)-i*delta)
00048 c             + S[unocc] psi(kn,r) psi(kn,r')^* /(w-e(kn)+i*delta)
00049
00050 c the screened coulomb potential
00051 c Wc(r,r';w)  = W(r,r';w) - v(|r-r'|)
00052 c             = < [r1,r2] v(|r-r1|) X(r1,r2;w) v(|r2-r'|) >
00053 c W(r,r';w)   = < [r''] ei(r,r'';w) v(|r''-r'| >
00054 c ei          = e^(-1), inverse dielectric matrix
00055 c             = 1 + vX
00056 c e           = 1 - vX0 in RPA
00057
00058 c expand Wc(r,r';w) in optimal product basis B
00059 c Wc(r,r';w)  = S[k=FBZ] S[i,j=1,nbloch]
00060 c               B(k,i,r) Wc(k,w)(i,j) B(k,j,r')^*
00061 c Wc(k,w)(i,j) are  the matrix elements of Wc in B
00062
00063 c SEc(q,t,t') = S[k=FBZ] S[n=occ]   S[i,j=1,nbloch]
00064 c        <psi(q,t) |psi(q-k,n) B(k,i)> <B(k,j) psi(q-k,n) |psi(q,t')>
00065 c        (i/2pi) <[w'=-inf,inf] Wc(k,w')(i,j)/(w'+w-e(q-k,n)-i*delta)>
00066 c
00067 c             + S[k=FBZ] S[n=unocc] S[i,j=1,nbloch]
00068 c        <psi(q,t) |psi(q-k,n) B(k,i)> <B(k,j) psi(q-k,n) |psi(q,t')>
00069 c        (i/2pi) <[w'=-inf,inf] Wc(k,w')(i,j)/(w'+w-e(q-k,n)+i*delta)>
00070
00071 c the analytic structure of GWc for w .le. ef
00072 c                               |
00073 c                               |   o = pole of G
00074 c                               ^   x = pole of Wc
00075 c                               |
00076 c                               |   ef-w
00077 c                               |----<-----
00078 c                               |          |
00079 c                 o  o  o  o  o |o  o  o   ^
00080 c               x  x  x  x  x  x|          |
00081 c  ------------------------------|---->------------------------------
00082 c                               |x  x  x  x  x  x  x  x
00083 c                               |             o  o  o  o  o
00084 c                               |       <----->
00085 c                               ^        gap in insulator
00086 c                               |
00087 c                               |
00088
00089 c the analytic structure of GWc for w .gt. ef
00090 c                               |
00091 c                               |   o = pole of G
00092 c                               |   x = pole of Wc
00093 c                               |
00094 c         gap in insulator      ^
00095 c                  <----->      |
00096 c      o  o  o  o               |
00097 c         x  x  x  x  x  x  x  x|
00098 c  ----------------------->----|----------------------------------
00099 c                  |           |x  x  x  x  x  x  x  x
00100 c                  ^   o  o  o  o  o  o  o
00101 c                  |           |
00102 c                   ------<----|
00103 c                      w-ef    |
00104 c                              ^
```

```
00105 c                               |
00106 c integration along the real axis from -inf to inf is equivalent to
00107 c the integration along the path shown
00108 c-------------------------------------------------------------
00109 c integration along the imaginary axis: wint (s. also wint.f) (takao ->wintz)
00110 c   (i/2pi) < [w'=-inf,inf] Wc(k,w')(i,j)/(w'+w-e(q-k,n) >
00111 c the i*delta becomes irrelevant
00112 c-------------------------------------------------------------
00113 c
00114 c omit k and basis index for simplicity and denote e(q-k,n) = e
00115 c wint = (i/2pi) < [w'=-inf,inf] Wc(w')/(w+w'-e) >
00116 c
00117 c w' ==> iw', w' is now real
00118 c wint = - (1/pi) < [w'=0,inf] Wc(iw') (w-e)/{(w-e)^2 + w'^2} >
00119 c
00120 c transform: x = 1/(1+w')
00121 c this leads to a denser mesh in w' around 0 for equal mesh x
00122 c which is desirable since Wc and the lorentzian are peaked around w'=0
00123 c wint = - (1/pi) < [x=0,1] Wc(iw') (w-e)x^2/{(w-e)^2 + w'^2} >
00124 c
00125 c the integrand is peaked around w'=0 or x=1 when w=e
00126 c to handel the problem, add and substract the singular part as follows:
00127 c wint = - (1/pi) < [x=0,1] { Wc(iw') - Wc(0)exp(-a^2 w'^2) }
00128 c                           * (w-e)/{(w-e)^2 +w'^2}x^2 >
00129 c        - (1/2) Wc(0) sgn(w-e) exp(a^2 (w-e)^2) erfc(a|w-e|)
00130 c
00131 c the second term of the integral can be done analytically, which
00132 c results in the last term
00133 c a is some constant
00134 c
00135 c when w = e, (1/pi) (w-e)/{(w-e)^2 + w'^2} ==> delta(w') and
00136 c the integral becomes -Wc(0)/2
00137 c this together with the contribution from the pole of G (s.u.)
00138 c gives the so called static screened exchange -Wc(0)
00139
00140 c-------------------------------------------
00141 c contribution from the poles of G: SEc(pole)
00142 c-------------------------------------------
00143 c
00144 c for w .le. ef
00145 c SEc(pole) = - S[k=FBZ] S[n=occ] S[i,j=1,nbloch]
00146 c        <psi(q,t) |psi(q-k,n) B(k,i)> <B(k,j) psi(q-k,n) |psi(q,t')>
00147 c             Wc(k,e(q-k,n)-w)(i,j) theta(e(q-k,n)-w)
00148 c
00149 c for w .gt. ef
00150 c SEc(pole) = + S[k=FBZ] S[n=unocc] S[i,j=1,nbloch]
00151 c        <psi(q,t) |psi(q-k,n) B(k,i)> <B(k,j) psi(q-k,n) |psi(q,t')>
00152 c             Wc(k,w-e(q-k,n))(i,j) theta(w-e(q-k,n))
00153 c
00154 c theta(x)  = 1   if x > 0
00155 c           = 1/2 if x = 0
00156 c           = 0   if x < 0
00157
00158 c FBZ = 1st BZ
00159 c NOTE: the routine only calculates the diagonal elements of the SE
00160 c       i.e. SEc(q,t)
00161
00162 c q       = q-vector in SEc(q,t)
00163 c itq     = states t at q
00164 c ntq     = no. states t
00165 c eq      = eigenvalues at q
00166 c ef      = fermi level in Rydberg
00167 c tr      = translational vectors in rot*R = R' + T
00168 c iatomp(R) = R'
00169 c ifrw,ifcw,ifrwi,ifcwi
00170 c   = direct access unit files for Re and Im coulomb matrix
00171 c     along real and imaginary axis
00172 c ifrb,ifcb,ifrhb,ifchb
00173 c         = direct access unit files for Re and Im b,hb
00174 c qbas    = base reciprocal lattice vectors
00175 c ginv    = inverse of qbas s. indxrk.f
00176 cxxxxx ippb,ipdb,idpb,iddb = pointers to work array w for
00177 c  ppb     = <phi(RLn) phi(RL'n') B(R,i)>
00178 c  pdb     = <phi(RLn) phidot(RL'n') B(R,i)>
00179 c  dpb     = <phidot(RLn) phi(RL'n') B(R,i)>
00180 c  ddb     = <phidot(RLn) phidot(RL'n') B(R,i)>
00181 c freq    = frequencies along real axis
00182 c freqx   = gaussian frequencies x between (0,1)
00183 c freqw   = (1-freqx)/freqx
00184 c wx      = weights at gaussian points x between (0,1)
00185 c ua      = constant in exp(-ua^2 w'^2) s. wint.f
00186 c expa    = exp(-ua^2 w'^2) s. wint.f
00187 c dw      = frequency mesh along real axis
00188 c deltaw  = energy mesh in SEc(qt,w) ---Not used now
00189 c iclass  = given an atom, tells the class
00190 c wk      = weight for each k-point in the FBZ
00191 c indexk  = k-point index
```

```
00192 c qbz     = k-points in the 1st BZ
00193 c nstar   = no. stars for each k
00194 c irk(k,R,nq) = gives index in the FBZ with k{IBZ, R=rotation
00195 c mdim    = dimension of B(R,i) for each atom R
00196 c work arrays:
00197 c rbq,cbq     = real and imaginary part of b(q)
00198 c rhbq,chbq   = real and imaginary part of hb(q)
00199 c rbkq,cbkq   = real and imaginary part of b(q-k)
00200 c rhbkq,chbkq = real and imaginary part of hb(q-k)
00201 c   b is the eigenvector of the LMTO-Hamiltonian
00202 c ekq     = eigenvalues at q-k
00203 c rmel,cmel = real and imaginary part of
00204 c             <psi(q,t') | psi(q-k,t) B(k,R,i)>
00205 c wr1 ... = work arrays
00206 c dimensions:
00207 c nqibz   = number of k-points in the irreducible BZ
00208 c n1,n2,n3= divisions along base reciprocal lattice vectors
00209 c natom   = number of atoms
00210 c nctot   = no. allowed core states
00211 c nbloch  = total number of Bloch basis functions
00212 c nlnmx   = maximum number of l,n,m
00213 c nlmto   = total number of LMTO basis functions
00214 c ngrp    = no. group elements (rotation matrices)
00215 c niw     = no. frequencies along the imaginary axis
00216 c nw      = no. frequencies along the real axis
00217 c niwx    = max(niw,nw)
00218 c
00219 c secq(t) = <psi(q,t) |SEc| psi(q,t)>
00220 c-------------------------------------------------------------------------
00221       intent(in)::
00222      i kount,ixc,deltaw,shtw,qip,itq, ntq,ef,ef2,esmr,esmr2,
00223      i nsp,isp,                !tiat,miat,
00224      i qbas,ginv,
00225      i qibz,qbz,wk,nstbz,wik,
00226      i nstar,irkip,
00227 c     i iclass,mdim,nlnmv,nlnmc,
00228 c     i icore,ncore,imdim,
00229 c     i ppb,
00230      i freq_r,freqx,wx,
00231      i dw,ecore,
00232      d nlmto,nqibz,nqbz,nctot,
00233 c     d nl,nnc,nclass,natom,mdimx,
00234      d nbloch,ngrp, nw_i,nw ,niw,niwx,nq, !nlnmx,
00235      & nblochpmx ,ngpmx,ngcmx,
00236      & wgt0,nq0i,q0i,symgg,alat, nband, ifvcfpout, !shtvg,
00237      & exchange,tote,screen,cohtest, ifexsp,
00238      i iwini,iwend,
00239      i nbmx,ebmx,
00240      i wklm,lxklm
00241 c     i   pomatr, qrr,nnr,nor,nnmx,nomx,nkpo,
00242 c     i invg,!il,in,im,nn_, lx,nx_,nxx_,dwplot !ppbrd, !cgr,,nlnm
00243
00244       integer :: ntq, nqbz,nqibz,ngrp,nq,nw,niw, !natom,
00245      & nband,  nlmto, nq0i,nctot,mbytes,iwksize,nlmtobnd,nstate,nstatex,
00246      & irot,  iqisp,ikpisp,isp,nsp, !nlnmx, !iq, idxk,
00247 c     &  iwr1,iwr2,iwr3,iwr4,iwc1,iwc2,iwc3,iwc4
00248      & ip, it,itp,             !ifcphi,    ! ifrb,ifcb,ifrhb,ifchb,
00249 c     i iiclass,                 !mdim(*),
00250      i ifrcw,ifrcwi,           !iindxk,
00251      & ifvcfpout,ndummy1,ndummy2,kx,kr,ngc,ngb,nbloch, !n1,n2,n3, k,
00252      & kp,nt0,nocc, nt0p,nt0m,irkp,i,nt0org,nmax,nt,ntp0,
00253      & nbmax,nblochpmx,ix,nx,iw,iwp,ixs,ixsmx, !nclass,nl,nnc,
00254      &  nwx,niwx,
00255      & itq(ntq), !,iatomp(natom),  !,miat(natom,ngrp),
00256      & nstar(nqibz),irkip(nqibz,ngrp,nq),kount(nqibz,nq)
00257 c
00258       real(8) :: q(3),qbas(3*3),ginv(3*3), !tr(3,natom), !,tiat(3,natom,ngrp)
00259      & wk(nqbz),wik(nqibz),qibz(3,nqibz),qbz(3,nqbz),
00260      & freqx(niw),wx(niw),      !expa(niw),
00261      & eq(nband,nq),
00262      & ekq(nband), ekc(nctot+nband),
00263      & tpi,ef,ef2,esmr,esmr2,efp,efm,wtx,wfac,wfacx,we,esmrx, !ua,
00264      & dw,wtt,wexx,www,exx,exxq ,wfacx2,weavx2,wex
00265 c      complex(8) :: zsec(-1:1,ntq,nq)
00266 c      real(8)    ::  shtw
00267 c                       ! This shft is  to avoid some artificial resonance effects.
00268 c                       ! shtw can be zero for esmr/=0 given by takao.
00269 c
00270       integer(4):: ngpmx, ngcmx, !ngcni(nqibz), !ngpn(nqbz),
00271      & igc,                     !ngvecpB(3,ngpmx,nqbz),ngveccBr(3,ngcmx,nqibz),
00272      & nadd(3)
00273       real(8) :: wgt0(nq0i,ngrp),qk(3), !qfbz(3),
00274      & qdiff(3),add(3),symgg(3,3,ngrp),symope(3,3), !qbasinv(3,3), det,
00275      & qxx(3),q0i(1:3,1:nq0i),shtv(3),alat,ecore(nctot), !shtvg(3,ngrp),
00276 c     &   ppb(1), !pdb(1),dpb(1),ddb(1), !*
00277      & coh(ntq,nq)             !, pos(3,natom)
00278       complex(8)::   alagr3zz,wintz !geigB  (ngpmx,nband,nqbz),
```

```
00279
00280 c
00281 c      real(8),allocatable:: !rmel(:,:,:),cmel(:,:,:),
00282 c     &                      rmelt(:,:,:),cmelt(:,:,:)
00283       complex(8),allocatable :: zz(:),zzmel(:,:,:),
00284      & zw(:,:), zwz(:,:,:), zwz0(:,:),zwzi(:,:),zwz00(:,:)
00285 c for exchange --------------------
00286       logical :: exchange,screen,cohtest,tote
00287       real(8),allocatable::
00288      & w1p(:,:,:),w2p(:,:,:),w3p(:,:)
00289       complex(8),allocatable :: z1p(:,:,:),vcoul(:,:),vcoult(:,:)
00290       integer:: invrot,invr
00291 c      integer:: invg(ngrp),il(*),in(*),im(*),nn_,lx(*),nx_(*),nxx_ !nlnm(*),
00292 c      real(8)::  cgr(*),ppbrd(*)
00293
00294 c- debugwrite ---------------------
00295       logical :: debug=.false. ,onceww
00296
00297 ccccccccccccc
00298 c tetra
00299 c      integer(4) :: ntqx
00300 c      integer(4) :: ibzx(nqbz)
00301 c      real(8)    :: wtet  (nband,nqibz,1:ntqx), wtetef(nband,nqibz)
00302 c                 ! where the last index is 3*itq+iw-1,itq=1,ntq,iw=-1,1
00303 c     logical    :: tetraex
00304 cccccczzcccccc
00305
00306       complex(8) :: wintzav,wintzsg_npm,wintzsg
00307
00308       integer(4) :: ibl,iii,ivsumxxx,ifexsp ,iopen
00309       integer(4),save::ifzwz=-999
00310
00311       integer(4) :: iwini, iwend, ia
00312       real(8)    :: esec, omega(ntq, iwini:iwend)
00313       complex(8) :: zsec(iwini:iwend,ntq,nq)
00314 c      complex(8),allocatable:: expikt(:)
00315       complex(8):: img=(0d0,1d0)
00316 ctakao
00317 c      complex(8):: cphiq(nlmto,nband), cphikq(nlmto,nband)
00318
00319       integer(4) :: nt_max, igb1,igb2,iigb,  nw_i !nw_i is at feb2006 TimeReversal off case
00320       complex(8),allocatable:: zmel3(:) !zmel1(:),
00321       complex(8), allocatable :: zw_(:,:) !,zzmel(:,:)
00322       complex(8), allocatable :: zwz2(:,:),zw2(:,:),zmel2(:,:) !0 variant
00323       complex(8) ::  zz2 ,zwz3(3) ,zwz3x
00324       real(8) :: dd,omg_c,dw2,omg
00325       real(8) :: freq_r(nw_i:nw)
00326       complex(8), allocatable :: zw3(:,:,:)
00327
00328
00329       real(8)::weavx,wfaccut=1d-10,qqqq
00330
00331       logical :: gausssmear=.true.,gass
00332       real(8) :: ebmx,ddw
00333       integer(4):: nbmx,nbmxe,nstatetot
00334
00335 c      integer(4):: n_index_qbz
00336 c      integer(4):: index_qbz(n_index_qbz,n_index_qbz,n_index_qbz)
00337
00338 c      integer(4)::icore(*),ncore(*),imdim(*) !,iclass(*),nlnmv(*),nlnmc(*),
00339
00340       integer(4)::verbose,nstbz(nqbz),bzcase=1,iqini,iqend
00341       real(8):: wgtq0p
00342
00343       integer(4):: nrec,kxx
00344       real(8)::quu(3),qibz_k(3),qbz_kr(3)
00345       logical ::  onlyimagaxis
00346
00347       logical ::zwz3mode
00348
00349
00350       real(8):: ua_,expa_(niw),ua2,freqw,freqw1,ratio,ua2_(niw)
00351 c$$$      logical :: ua_auto !fixed to be .false.
00352       integer(4):: icc=0
00353       real(8),allocatable:: uaa(:,:)
00354
00355 c      logical ::testimx=.false.
00356 ccccc zvz test ccccccccccccccccccccccccc
00357       integer(4):: ngbx
00358 c      complex(8):: vcoul(ngbx,ngbx)
00359       complex(8),allocatable:: vzz(:,:,:),aaa(:), zwzs(:)
00360       complex(8):: zvz,zvz1
00361       integer(4):: ib1,ib2,ifix
00362 ccccccccccccccccccccccccccccccccc
00363       logical ::iww2=.true., oncew
00364
00365
```

```
00366 C...
00367 c      logical::smbasis
00368       integer(4):: iclose,isx,iqx !nn,no,ifpomat,
00369 c      complex(8),allocatable:: pomat(:,:)
00370       real(8):: q_r(3)
00371 c      integer(4):: nnmx,nomx,nkpo, nnr(nkpo),nor(nkpo)
00372 c      complex(8):: pomatr(nnmx,nomx,nkpo)
00373 c      real(8):: qrr(3,nkpo)
00374
00375       real(8):: elxx,ehxx,ekxx,efxx
00376       integer(4):: ixsmin,iwm,iir,nwxi, itini,itend, npm
00377       real(8)   :: fffr(3),ppp
00378       complex(8):: zwzz(3)
00379
00380       real(8),allocatable:: ebb(:)
00381       integer(4):: ii,iq
00382       logical ::evaltest        !, imgonly
00383
00384       integer:: lxklm,ivc,ifvcoud,idummy,iy,ngb0
00385       real(8):: wklm((lxklm+1)**2),pi,fpi,vc,qvv(3),aaaa
00386       complex(8)::zmelt1,zmelt0
00387       real(8)::voltot
00388 c      logical :: newaniso !fixed to be T
00389
00390       complex(8),allocatable:: ppovl(:,:),zcousq(:,:) !,ppovlz(:,:)
00391       real(8),allocatable::vcoud(:),vcousq(:)
00392       integer:: mrecl,nprecx,ifwd
00393       character(5):: charnum5
00394
00395       integer:: ixc
00396       real(8):: qip(3,*),deltaw,shtw,eqx(nband),dwplot
00397       complex(8),allocatable:: zmelt(:,:)
00398       integer:: ntqxx,nrot
00399 c-------------------------------------------------------------------
00400       write(6,*)'sxcf_fal3z'
00401 c      timemix=.false.
00402       pi  = 4d0*datan(1d0)
00403       fpi = 4d0*pi
00404       debug=.false.
00405       if(verbose()>=90) debug=.true.
00406 !!
00407       if(.not.exchange) then
00408         ifwd = iopen('WV.d',1,-1,0)
00409         read (ifwd,*) nprecx,mrecl
00410         ifwd = iclose('WV.d')
00411 c$$$!! --- gauss_img : interpolation gaussion for W(i \omega).
00412 c$$$      call getkeyvalue("GWinput","gauss_img",ua_,default=1d0)
00413 c$$$      if(ua_<=0d0) then
00414 c$$$        ua_auto =.true.
00415 c$$$        write(6,"(' ua_auto=T')")
00416 c$$$      else
00417 c$$$        ua_auto =.false.
00418 c$$$        do ix = 1,niw
00419 c$$$          freqw     = (1d0 - freqx(ix))/ freqx(ix)
00420 c$$$          expa_(ix) = exp(-(ua_*freqw)**2)
00421 c$$$        enddo
00422 c$$$      endif
00423         call getkeyvalue("GWinput","gauss_img",ua_,default=1d0)
00424         do ix = 1,niw             !! Energy mesh; along im axis.
00425           freqw     = (1d0 - freqx(ix))/ freqx(ix)
00426           expa_(ix) = exp(-(ua_*freqw)**2)
00427         enddo
00428         npm = 1                   ! npm=1    Timeveversal case
00429         if(nw_i/=0) npm = 2       ! npm=2 No TimeReversal case. Need negative energy part of W(omega)
00430       endif
00431
00432       tpi         = 8d0*datan(1.d0)
00433       if(nctot/=0) ekc(1:nctot)= ecore(1:nctot) ! core
00434       nlmtobnd    = nlmto*nband
00435       nstatetot      = nctot + nband
00436 c      call dinv33(qbas,0,qbasinv,det)
00437 c      allocate(expikt(natom))
00438
00439
00440 !! == ip loop to spedify external q ==
00441       do 1001 ip = 1,nq
00442         if(sum(irkip(:,:,ip))==0) cycle
00443         q = qip(1:3,ip)
00444         write (*,*) ip,'  out of ',nq,'  k-points ' ! call cputid  (0)
00445         if(ixc==2) then
00446           call readeval(q,isp,eqx)
00447           do iw = iwini,iwend
00448             do i  = 1,ntq
00449               omega(i,iw) = eqx(itq(i)) + 2d0*(dble(iw)-shtw)*deltaw
00450             enddo
00451           enddo
00452         endif
```

```
00453 !!
00454         if(ixc==4) then
00455 c        dwplot=0.01
00456         do iw = iwini,iwend
00457           omega(1:ntq,iw) =  dwplot* iw + ef
00458         enddo
00459         endif
00460
00461         call readeval(q, isp, eq(1,ip))
00462 !! we only consider bzcase()==1
00463         if(abs(sum(qibz(:,1)**2))/=0d0) call rx( ' sxcf assumes 1st qibz/=0 ')
00464         if(abs(sum( qbz(:,1)**2))/=0d0) call rx( ' sxcf assumes 1st qbz /=0 ')
00465         If (tote) exxq = 0.d0
00466
00467 !! == Big loop for kx ==
00468 !! kx is for irreducible k points, kr=irk(kx,irot) runs all k points in the full BZ.
00469         iqini=1
00470         iqend=nqibz             !no sum for offset-Gamma points.
00471         do 1100 kx = iqini,iqend
00472           if(sum(irkip(kx,:,ip))==0) cycle
00473           write(6,*) ' ### do 1100 start kx=',kx,' from ',iqini,' through', iqend
00474 c          if( kx <= nqibz ) then
00475             qibz_k= qibz(:,kx)
00476 c          else
00477 c            qibz_k= 0d0
00478 c         endif
00479           if(verbose()>=40)  write(6,*) ' sxcf_fal3z: loop 1100 kx=',kx
00480           call readqg0('QGcou',qibz_k,ginv,  quu,ngc)
00481           ngb = nbloch + ngc    !oct2005
00482           if(debug) write(6,*) ' sxcf: ngb=',ngb,nbloch
00483
00484 !! ===Readin diagonalized Coulomb interaction===
00485 !!  Vcoud file is sequential file Vcoulomb matrix for qibz_k.
00486 !!  A possible choice for paralellization is "Vcoud.ID" files where ID=kx
00487 !!  Vould file is written in hvccfp0.m.F.
00488 !! For correlation, W-v is read instead of Vcoud file (ifrcw,ifrcwi for WVR and WVI)
00489 !! These can be also separeted into WVR.ID and WVI.ID files.
00490 !! NOTE: vcoud and zcousq are in module m_zmelt.
00491 c          if(kx<=nqibz) qxx=qibz_k
00492 c          if(kx>nqibz ) qxx=q0i(:,kx-nqibz)
00493           qxx=qibz_k
00494           ifvcoud = iopen('Vcoud.'//charnum5(kx),0,0,0)
00495           do
00496             read(ifvcoud) ngb0
00497             read(ifvcoud) qvv
00498             if(allocated(vcoud)) deallocate(vcoud)
00499             allocate( zcousq(ngb0,ngb0),vcoud(ngb0) )
00500             read(ifvcoud) vcoud
00501             read(ifvcoud) zcousq
00502             if(sum(abs(qvv-qxx))<1d-6) goto 1133
00503           enddo
00504           if(sum(abs(qvv-qxx))>1d-6) then
00505             write(6,*)'qvv =',qvv
00506             write(6,*)'qxx=',qxx,kx
00507             call rx( 'sxcf_fal2: qvv/=qibz(:,kx) hvcc is not consistent')
00508           endif
00509  1133     continue
00510           if( ngb0/=ngb ) then  !sanity check
00511             write(6,*)' qxx ngb0 ngb=',qxx,ngb0,ngb
00512             call rx( 'hsfp0.m.f:ngb0/=ngb')
00513           endif
00514 !! used in get_zmel
00515 !! <I|v|J>= \sum_mu ppovl*zcousq(:,mu) v^mu (Zcousq^*(:,mu) ppovl)
00516 !! zmel contains O^-1=<I|J>^-1 factor. zmel(phi phi J)= <phi phi|I> O^-1_IJ
00517 !! ppovlz= O Zcousq
00518 !! (V_IJ - vcoud_mu O_IJ) Zcousq(J, mu)=0, where Z is normalized with O_IJ.
00519           if(allocated(ppovl)) deallocate(ppovl,ppovlz)
00520           allocate(ppovl(ngc,ngc),ppovlz(ngb,ngb))
00521           call readppovl0(qibz_k,ngc,ppovl)
00522           ppovlz(1:nbloch,:) = zcousq(1:nbloch,:)
00523           ppovlz(nbloch+1:nbloch+ngc,:) = matmul(ppovl,zcousq(nbloch+1:nbloch+ngc,:))
00524           deallocate(zcousq)
00525 !! === open WVR,WVI for correlation mode ===
00526           if(.not.exchange) then
00527             ifrcw  = iopen('WVR.'//charnum5(kx),0,-1,mrecl)
00528             ifrcwi = iopen('WVI.'//charnum5(kx),0,-1,mrecl)
00529           endif
00530           nrot=0
00531           do irot = 1,ngrp
00532 c            if( kx <= nqibz) then
00533               kr = irkip(kx,irot,ip) ! index for rotated kr in the FBZ
00534               if(kr==0) cycle   ! next irot
00535               qbz_kr= qbz(:,kr)
00536 c            else
00537 c              kr=-99999         !for sanity check
00538 c              qbz_kr= 0d0
00539 c              if( wgt0(kx-nqibz,irot)==0d0 ) cycle ! next irot
```

```
00540 c            endif
00541             nrot=nrot+1
00542           enddo
00543
00544 !! == loop over rotations ==
00545 !! We may extend
00546           do 1000 irot = 1,ngrp
00547 c            if( kx <= nqibz) then
00548              kr = irkip(kx,irot,ip) ! index for rotated k in the FBZ
00549              if(kr==0) cycle
00550              qbz_kr= qbz(:,kr)
00551 c            else
00552 c              kr=-99999         !for sanity check
00553 c              qbz_kr= 0d0
00554 c              if( wgt0(kx-nqibz,irot)==0d0 ) cycle
00555 c            endif
00556             write(*,"('ip,kx irot=',3i5, ' out of',2i4)") ip,kx,irot, iqend,ngrp
00557
00558 c qk = q - rk, rk is inside 1st BZ, not restricted to the irreducible BZ
00559             qk =  q - qbz_kr    ! qbz(:,kr)
00560             call readeval(qk, isp, ekq)
00561             ekc(nctot+1:nctot+nband) = ekq(1:nband)
00562             nt0 = nocc(ekc,ef,.true.,nstatetot)
00563             ddw= .5d0
00564 c        if(GaussSmear) ddw= 10d0
00565             ddw=10d0
00566             efp= ef+ddw*esmr
00567             efm= ef-ddw*esmr
00568             nt0p = nocc(ekc,efp,.true.,nstatetot)
00569             nt0m = nocc(ekc,efm,.true.,nstatetot)
00570 !! nbmx1 ebmx1: to set how many bands of <i|sigma|j>  do you calculate.
00571 !! nbmx2 ebmx2: to restrict num of bands of G to calculate G \times W
00572             if(exchange) then
00573               nbmax = nt0p-nctot
00574               if(debug) write(6,*)' sxcf: nbmax nctot nt0p =',nbmax,nctot,nt0p
00575             else
00576               nbmax = nband
00577               nbmxe = nocc(ekc,ebmx,.true.,nstatetot)-nctot
00578               nbmax  = min(nband,nbmx,nbmxe)
00579               if(onceww(3)) write(6,*)' nbmax=',nbmax
00580             endif
00581             nstate = nctot + nbmax ! = nstate for the case of correlation
00582
00583 !! all are identical.
00584             ntp0 = ntq
00585             ntqxx= ntp0
00586
00587 !! Get matrix element zmelt= rmelt + img*cmelt, defined in m_zmel.F---
00588             if(debug) write(6,*)'zzBBB ppovlz =',sum(abs(ppovlz(:,:))),kx,irot
00589             if(allocated(zmel)) deallocate(zmel)
00590             if(allocated(zmeltt)) deallocate(zmeltt)
00591 ! this return zmeltt (for exchange), or zmel (for correlation)
00592             call get_zmelt(exchange,q,kx,qibz_k,irot,qbz_kr,kr,isp,
00593      &       ngc,ngb,nbmax,ntqxx,nctot,ncc=0)
00594             if(kx<= nqibz) then
00595               wtt = wk(kr)      !         wtx = 1d0
00596             else
00597               wtt = wk(1)*wgt0(kx-nqibz,irot) !       wtx = wgt0(kx-nqibz,irot)
00598               if(abs(wk(1)-1d0/dble(nqbz))>1d-10)call rx( 'sxcf:wk(1)inconsistent')
00599             endif
00600             if(debug) write(6,*) 'ssssssss',size(zmel),ntqxx*nstate*ngb
00601             if(debug) write(6,"(' kx wtt=',i4,f12.8)") kx,wtt
00602             if(debug) write(6,*)' 000 sumzmel=',ngb, nstate, ntp0,sum(abs(real(zmel))),sum(abs(imag(zmel)))
00603
00604 !!-----------------------------------------------------------
00605 !! === exchange section ===
00606 !!-----------------------------------------------------------
00607 c
00608 c S[i,j=1,nbloch] <psi(q,t) |psi(q-rk,n) B(rk,i)>
00609 c                       v(k)(i,j) <B(rk,j) psi(q-rk,n) |psi(q,t')>
00610 c
00611 c> z1p(j,t,t') = S[i=1,nbloch] <psi(q,t') | psi(q-rk,t) B(rk,i)> v(k)(i,j)
00612 !! NOTE: zmel(igb, nctot+nbmax, ntp0) ---> <phi phi  |igb>
00613
00614 c --- screened exchange case ----
00615 c          if(screen) then
00616 c            ix = 1
00617 c            nrec=(kx-iqini)*nw+ix
00618 c            if(bzcase()==2) nrec= (kx-1)*nw+ix
00619 c            read(ifrcw,rec=nrec) zw  ! Readin W(0) - v
00620 c            vcoul = vcoul + zw(1:ngb,1:ngb) !c  screen test
00621 c          endif
00622
00623 c          allocate( zmel(ngb, nctot+nbmax, ntp0), w3p( nctot+nbmax,ntp0))
00624 c          zmel  = dcmplx (rmelt,cmelt)
00625             if(exchange) then
00626               allocate( w3p( nctot+nbmax,ntp0))
```

```
00627               do 992 itp = 1,ntp0
00628                 do 993 it  = 1,nctot+nbmax
00629                   w3p(it,itp) = 0d0
00630                   do 994 ivc=1,ngb
00631                     if(ivc==1.and.kx==1) then
00632                       vc= wklm(1)* fpi*sqrt(fpi) /wk(kx)
00633 c                    write(6,*)'wklm(1) vc=',wklm(1),vc
00634                     else
00635                       vc= vcoud(ivc)
00636                     endif
00637                     w3p(it,itp) = w3p(it,itp)+ vc * abs(zmeltt(it,itp,ivc))**2
00638  994              continue
00639  993            continue
00640  992          continue
00641               if(debug) then
00642                 do  it  = 1,nctot+nbmax
00643                   do  itp = 1,ntp0
00644                     write(6,"(' w3p =',2i4,2d14.6)") it,itp,w3p(it,itp)
00645                   enddo
00646                 enddo
00647               endif
00648
00649 !! Write the Spectrum function for exchange May. 2001.
00650 !!!!!! Probably, Need to fix this....
00651               if(ifexsp/=0) then
00652                 do it  = 1, nctot+nbmax
00653                   do itp = 1,ntp0
00654                     write(ifexsp,"(3i4, 3f12.4, ' ',d23.15,'  ',d23.15)")
00655      &               ip,itp,it, qbz_kr, ekc(it), -wtt*w3p(it,itp)
00656                   enddo
00657                 enddo
00658               endif
00659
00660 !! --- Correct weigts wfac for valence by esmr
00661               do it = nctot+1, nctot+nbmax
00662                 wfac = wfacx(-1d99, ef, ekc(it), esmr) !gaussian
00663                 w3p(it,1:ntp0) = wfac * w3p(it,1:ntp0)
00664               enddo
00665
00666               if (.not.tote) then !total energy mode tote
00667                 do itp = 1,ntp0 !S[j=1,nbloch]  z1p(j,t,t') <B(rk,j) psi(q-rk,n) |psi(q,t')>
00668                   zsec(iwini,itp,ip) = zsec(iwini,itp,ip)
00669      &             - wtt * sum( w3p(:,itp) )
00670                 enddo
00671               else
00672                 do itp = 1,ntp0
00673                   wfac = wfacx(-1d99, ef2, eq(itq(itp),ip), esmr2) !gaussian
00674                   w3p(1:nctot+nbmax,itp) = wfac * w3p(1:nctot+nbmax,itp)
00675                   exxq = exxq - wtt * sum( w3p(:,itp) )
00676                 enddo
00677               endif
00678               deallocate( w3p)  !,rmelt,cmelt)
00679               cycle
00680             endif
00681 c-- End of exchange section --------------
00682
00683
00684
00685 c-------------------------------------------------------------------------
00686 c--- correlation section -------------------------------------------------
00687 c-------------------------------------------------------------------------
00688 c$$$c--- The matrix elements zmel.
00689 c$$$c        allocate( zmel (ngb, nstate, ntp0) )
00690 c$$$c        zmel = dcmplx (rmelt,-cmelt)
00691 c$$$c        if(newaniso) then
00692 c$$$c#ifdef USE_GEMM_FOR_SUM
00693 c$$$          if(verbose()>39)write(*,*)'info: USE GEMM FOR SUM (zmel=zmel*ppovlz), in sxcf_fal2.F'
00694 c$$$          allocate( zmelt (ngb, nstate) )
00695 c$$$          do itp=1,ntp0
00696 c$$$          zmelt = dcmplx(rmelt(:,:,itp),-cmelt(:,:,itp))
00697 c$$$          call zgemm('C','N',ngb,nstate,ngb,(1d0,0d0),
00698 c$$$     .      ppovlz,ngb,zmelt,ngb,(0d0,0d0),zmel(1,1,itp),ngb)
00699 c$$$          enddo
00700 c$$$          deallocate(zmelt)
00701 c$$$#else
00702 c$$$          do itp=1,ntp0
00703 c$$$            do it=1,nstate
00704 c$$$              zmel(:,it,itp) =  matmul(zmel(:,it,itp),dconjg(ppovlz(:,:)))
00705 c$$$            enddo
00706 c$$$          enddo
00707 c$$$#endif
00708 c$$$c        endif
00709 c        deallocate(rmelt,cmelt)
00710 c        if(debug) write(6,*)' end of zmel'
00711
00712 c=======================================================================
00713 c The correlated part of the self-energy:
```

```
00714 c S[n=all] S[i,j=1,nbloch]
00715 c <psi(q,t) |psi(q-rk,n) B(rk,i)>
00716 c  < [w'=0,inf] (1/pi) (w-e)/{(w-e)^2 + w'^2} Wc(k,iw')(i,j) >
00717 c                                <B(rk,j) psi(q-rk,n) |psi(q,t)>
00718 c e = e(q-rk,n), w' is real, Wc = W-v
00719 c=======================================================================
00720             allocate( zw(nblochpmx,nblochpmx) )
00721 c=======================================================================
00722 c contribution to SEc(qt,w) from integration along the imaginary axis
00723 c=======================================================================
00724 c-----------------------------------------------
00725 c loop over w' = (1-x)/x, frequencies in Wc(k,w')
00726 c {x} are gaussian points between (0,1)
00727 c-----------------------------------------------
00728             allocate( zwz0(nstate,ntp0) )
00729             ix = 1  - nw_i      !at omega=0
00730 c        nrec=(kx-iqini)*(nw-nw_i+1) +ix ! 2---> iqini
00731 c        if(bzcase()==2) nrec= (kx-1)*(nw-nw_i+1) +ix
00732             nrec=ix
00733             if(debug) write(6,*)' wvr nrec kx nw nw_i ix=',nrec,kx,nw,nw_i,ix
00734             read(ifrcw,rec=nrec) zw ! direct access read Wc(0) = W(0) - v
00735             zwz0=0d0
00736 !! this loop looks complicated but just in order to get zwz0=zmel*zwz0*zmel
00737 !! Is this really efficient???
00738 CCC!$OMP parallel do private(itp,it,igb2,zz2)
00739             do itp=1,ntp0
00740               do it=1,nstate
00741                 do igb2=2,ngb
00742                   zz2 = sum( dconjg(zmel(1:igb2-1,it,itp))*zw(1:igb2-1,igb2) )
00743                   zwz0(it,itp) = zwz0(it,itp)+zz2*zmel(igb2,it,itp)*2d0+
00744      &             dconjg(zmel(igb2,it,itp))*zw(igb2,igb2)*zmel(igb2,it,itp)
00745                 enddo           !igb2
00746                 zwz0(it,itp) = zwz0(it,itp)+
00747      &            dconjg(zmel(1,it,itp))*zw(1,1)*zmel(1,it,itp)
00748               enddo             !it
00749             enddo               !itp
00750             zwz0 = dreal(zwz0)
00751 c COH term test ----- The sum of the all states for zwz00 gives the delta function.
00752             if(cohtest) then
00753               do itp = 1,ntq
00754                 coh(itp,ip)  = coh(itp,ip)
00755      &           + .5d0*wtt*sum(dreal(zwz0(1:nstate,itp)))
00756               enddo
00757               deallocate(zw,zwz0,zmel)
00758               cycle
00759             endif
00760 c
00761             nx  = niw
00762             if(niw <1) call rx( " sxcf:niw <1")
00763             if(allocated(zwz)) deallocate(zwz)
00764             if(allocated(zwzi)) deallocate(zwzi)
00765             allocate( zwz(niw*npm, nstate,ntp0),  zwzi(nstate,ntp0) )
00766             if(screen) allocate( zwz00(nstate,ntp0) )
00767             if(verbose()>50) write(*,'("6 before matzwz in ix cycle ",$)')
00768             if(verbose()>50) call cputid(0)
00769
00770             zwz=0d0
00771             do ix = 1,nx        !*npm            ! imaginary frequency w'-loop
00772               nrec= ix
00773               if(debug) write(6,*)' wvi nrec=',nrec
00774               read(ifrcwi,rec=nrec) zw ! Readin W-v on imag axis
00775               if(npm==1) then   !then zwz is real so, we can use mode c2.
00776                 do itp= 1,ntp0
00777                   do it = 1,nstate
00778                     ppp=0d0
00779                     do igb2 = 2,ngb
00780                       zz2 = sum( dconjg(zmel(1:igb2-1,it,itp))*zw(1:igb2-1,igb2) )
00781 ! only take real part
00782                       ppp = ppp + dreal(zz2*zmel(igb2,it,itp)) * 2d0
00783      &                 + dconjg(zmel(igb2,it,itp))*zw(igb2,igb2)*zmel(igb2,it,itp)
00784                     enddo       !igb2
00785                     zwz(ix,it,itp) = ppp +
00786      &               dconjg(zmel(1,it,itp))*zw(1,1)*zmel(1,it,itp)
00787                   enddo         !it
00788                 enddo           !itp
00789               else              !we need to use mode2 because zwz is not real now.
00790                 call matzwz( zw(1:ngb,1:ngb), zmel, ntp0,nstate,ngb,
00791      o           zwz(ix,1:nstate,1:ntp0))
00792               endif
00793               if(debug) write(6,*)' sumzw=',sum(abs(zw))
00794             enddo               !ix
00795             if(verbose()>50) write(*,'("xxx:6.1 before matzwz in ix cycle ",$)')
00796             if(verbose()>50) call cputid(0)
00797             if(debug) write(6,*)' sumzmel=',ngb, nstate, ntp0,sum(abs(real(zmel))),sum(abs(imag(zmel)))
00798
00799 c-----------------------------------------------------------------
00800 c S[i,j] <psi(q,t) |psi(q-rk,n) B(rk,i)>
```

```
00801 c               Wc(k,0)(i,j) > <B(rk,j) psi(q-rk,n) |psi(q,t)>
00802 c needed to take care of the singularity in the w' integration
00803 c when w-e(q-rk,n) is small
00804 c-------------------------------------------------------------
00805             if(screen) then
00806               zwz00 = zwz0
00807               zwz0  = 0d0
00808               do ix = 1,nx
00809                 zwz(ix,:,:)=zwz(ix,:,:) - zwz00
00810               enddo
00811             endif
00812
00813 c-----------------------------------------------
00814 c loop over w in SEc(qt,w)
00815 c-----------------------------------------------
00816 c$$$        if(ua_auto) then
00817 c$$$          allocate(uaa(nstate,ntq))
00818 c$$$          do itp = 1,ntq
00819 c$$$            do  it = 1,nstate
00820 c$$$              ratio = abs(zwz(niw,it,itp)/zwz0(it,itp))
00821 c$$$              call gen_uaa(ratio,freqx(niw),  uaa(it,itp))
00822 c$$$              if(verbose()>45) then
00823 c$$$                write(6,"(' it itp uaa=',2i4,12f8.4)") it,itp,uaa(it,itp)
00824 c$$$              elseif(verbose()>40.and.mod(it,10)==1.and.mod(itp,10)==1) then
00825 c$$$                write(6,"(' it itp uaa=', 2i4,12f8.4)") it,itp,uaa(it,itp)
00826 c$$$              endif
00827 c$$$            enddo
00828 c$$$          enddo
00829 c$$$        endif
00830             allocate(zwzs(npm*nx))
00831             do iw = iwini,iwend
00832 c frequency integration along the imaginary axis, s. wint.f
00833 c for each e(q-rk,n) and w in SEc(qt,w)
00834               do 1385  itp = 1,ntq
00835                 do 1387 it = 1,nstate
00836                   we =.5d0*( omega(itp,iw) -ekc(it)) != .5d0*(
      eq(itq(itp),ip)+2d0*(dble(iw)-shtw)*deltaw-ekc(it))
00837                   if(verbose()>50) then
00838                     do  ix = 1,niw
00839                       ratio  = abs(zwz(ix,it,itp)/zwz0(it,itp))
00840                       freqw1 = (1d0 - freqx(ix))/ freqx(ix)
00841                       ua2_(ix) = sqrt(- 1d0/freqw1*log(ratio))
00842                     enddo
00843                     write(6,"(' sxcf_fal2: ua=sqrt(1/w1*log(v0/v1))=',12f8.4)") ua2_(1:niw)
00844                   endif
00845 c           if(ua_auto) then
00846 c             call gen_ua(abs(zwz(niw,it,itp)/zwz0(it,itp)), niw,freqx, expa_,ua_)
00847 c             if(iw==ini) then
00848 c             if(verbose()>45) then
00849 c               write(6,"(' it itp ua_=',2i4,12f8.4)")it,itp,ua_
00850 c             elseif(verbose()>40.and.mod(it,20)==1.and.mod(itp,20)==1) then
00851 c               write(6,"(' it itp ua_=',3i4,12f8.4)")it,itp,ua_
00852 c             elseif(irot==1.and.mod(it,10)==1.and.itp==it) then
00853 c               write(6,"(' it itp ua_=',3i4,12f8.4)")it,itp,ua_
00854 c             endif
00855 c             endif
00856 c           endif
00857 c$$$              if(ua_auto) then
00858 c$$$                ua_ = .5d0*uaa(it,itp)
00859 c$$$                call gen_expa(niw,freqx,ua_,  expa_)
00860 c$$$              endif
00861                   esmrx = esmr
00862                   if(it <= nctot) esmrx = 0d0
00863                   do ix=1,nx
00864                     zwzs(ix   ) = dreal( zwz(ix,it,itp)) ! w(iw) + w(-iw) symmetric part
00865                     if(npm==2) then
00866                       zwzs(ix+nx) = dimag( zwz(ix,it,itp)) ! w(iw) - w(-iw)
00867                     endif
00868                   enddo
00869 c                  if(GaussSmear) then
00870                     zwzi(it,itp) =
00871      &               wintzsg_npm(npm, zwzs,zwz0(it,itp),freqx,wx,ua_,expa_,we,nx, esmrx)
00872 c                  else
00873 c                    if(npm==2)
00874 c     &               call rx( ' ###Not impliment wintzav for npm=2. Use Gausssmear.')
00875 c                    zwzi(it,itp) =
00876 c     &               wintzav( zwzs,zwz0(it,itp),freqx,wx,ua_,expa_,we,nx, esmrx)
00877 c                  endif
00878 c     .    wintz (zwz(1,it,itp),zwz0(it,itp),freqx,wx,ua,expa,we,nx)
00879 ccccccccccccccccccccccccccccc
00880 c           if(verbose()>45) then
00881 c           if(it==50.and.itp==1) then
00882 c           write(6,"(' it itp abs(zwzi)=',2i4,12d13.5)")it,itp,abs( zwzi(it,itp))
00883 c           icc=icc+1
00884 c           if(icc==10) stop 'test end'
00885 c           endif
00886 c           endif
```

```
00887 ccccccccccccccccccccccccccc
00888  1387           continue
00889  1385         continue
00890 c sum over both occupied and unoccupied states and multiply by weight
00891               do  itp = 1,ntq
00892                 zsec(iw,itp,ip)  = zsec(iw,itp,ip) + wtt*sum(zwzi(:,itp))
00893               enddo
00894 c end of SEc w-loop
00895             enddo
00896             deallocate(zwzs)
00897             if(debug) then
00898               write(6,*)' ntq nstate sum(zwzi)=',ntq,nstate,sum(zwzi)
00899               write(6,*)' ntq nstate sum(zwz )=',ntq,nstate,sum(zwz)
00900               do itp = 1,ntq
00901                 write(6,'(" zsec=",i3,6d15.7)') itp,zsec(iwini:iwini+2,itp,ip)
00902               enddo
00903             endif
00904             deallocate(zwz,zwz0,zwzi)
00905
00906 c=======================================================================
00907 c contribution to SEc(qt,w) from the poles of G
00908 c=======================================================================
00909 !    We assume freq_r(i) == -freq_r(-i) in this code. feb2006
00910 c--------------------------------------
00911 c maximum ixs finder
00912 c--------------------------------------
00913 c      write(6,*)' ekc at nt0p nt0m+1=', ekc(nt0p),ekc(nt0m+1)
00914 c      write(6,*)'  nt0p nt0m+1=', nt0p, nt0m+1
00915             ixsmx =0
00916             ixsmin=0
00917             do 3001 iw  = iwini,iwend
00918               do 3002 itp = 1,ntq
00919                 omg = omega(itp,iw)
00920                 if (omg < ef) then
00921                   itini= 1
00922                   itend= nt0p
00923                 else
00924                   itini= nt0m+1
00925                   itend= nstate
00926                 endif
00927                 do 3011 it= itini,itend
00928                   esmrx = esmr
00929                   if(it<=nctot) esmrx = 0d0
00930                   wfac = wfacx2(omg,ef, ekc(it),esmrx)
00931                   if(gausssmear) then
00932                     if(wfac<wfaccut) cycle
00933                     we = .5d0*(omg-weavx2(omg,ef,ekc(it),esmr))
00934                   else
00935                     if(wfac==0d0) cycle
00936                     if(omg>=ef) we = max( .5d0*(omg-ekc(it)), 0d0) ! positive
00937                     if(omg< ef) we = min( .5d0*(omg-ekc(it)), 0d0) ! negative
00938                   endif
00939                   do iwp  = 1,nw ! may2006
00940                     ixs = iwp   ! ixs = iwp= iw+1
00941 c                 write (*,*) 'xxx freq we=',freq_r(iwp),abs(we)
00942                     if(freq_r(iwp) > abs(we)) exit
00943                   enddo
00944 c This change is because G(omega-omg') W(omg') !may2006
00945 c             if(ixs>ixsmx  .and. omg<=ef ) ixsmx  = ixs
00946 c             if(ixs>ixsmin .and. omg> ef ) ixsmin = ixs
00947                   if(ixs>ixsmx  .and. omg>=ef ) ixsmx  = ixs
00948                   if(ixs>ixsmin .and. omg< ef ) ixsmin = ixs
00949                   wexx  = we
00950                   if(ixs+1 > nw) then
00951                     write (*,*) ' nw_i ixsmin',nw_i, ixsmin
00952                     write (*,*) ' wexx, dw ',wexx,dw
00953                     write (*,*) ' omg ekc(it) ef ', omg,ekc(it),ef
00954 Cstop2rx 2013.08.09 kino                stop ' sxcf 222: |w-e| out of range'
00955                     call rx( ' sxcf 222: |w-e| out of range')
00956                   endif
00957  3011           continue
00958  3002         continue          !end of SEc w and qt -loop
00959  3001       continue            !end of SEc w and qt -loop
00960             if(nw_i==0) then
00961               nwxi = 0
00962               nwx  = max(ixsmx+1,ixsmin+1)
00963             else
00964               nwxi = -ixsmin-1
00965               nwx  =  ixsmx+1
00966             endif
00967             if (nwx > nw   ) then
00968               call rx( ' sxcf nwx check : |w-e| > max(w)')
00969             endif
00970             if (nwxi < nw_i) then
00971               call rx( ' sxcf nwxi check: |w-e| > max(w)')
00972             endif
00973             if(debug) write(6,*)' nwxi nwx nw=',nwxi,nwx,nw
```

```
00974
00975 C... Find nt_max -----------------------------------
00976             nt_max=nt0p         !initial nt_max
00977             do 4001 iw  = iwini,iwend
00978               do 4002 itp = 1,ntq
00979                 omg     = omega(itp,iw)
00980                 if (omg > ef) then
00981                   do  it = nt0m+1,nstate ! nt0m corresponds to efm
00982                     wfac = wfacx2(ef,omg, ekc(it),esmr)
00983                     if( (gausssmear.and.wfac>wfaccut)
00984      &                .or.(.not.gausssmear.and.wfac/=0d0)) then
00985                       if (it > nt_max) nt_max=it ! nt_max is  unocc. state
00986                     endif       ! that ekc(it>nt_max)-omega > 0
00987                   enddo
00988                 endif
00989  4002         continue
00990  4001       continue
00991
00992 C... Set zw3 or zwz ----------------------------------
00993             zwz3mode=.true.
00994             if(iwend-iwini>2) then
00995               zwz3mode=.false.
00996             endif
00997             if(zwz3mode) then
00998               allocate( zw3(ngb,ngb,nwxi:nwx))
00999               do ix = nwxi,nwx  ! real frequency w'-loop
01000                 nrec=ix-nw_i+1
01001                 if(debug) write(6,*)' wvr3 nrec=',nrec,nblochpmx,kx,ix,nw
01002                 read(ifrcw,rec=nrec) zw
01003                 zw3(1:ngb,1:ngb,ix) = zw(1:ngb,1:ngb)
01004                 if(evaltest()) then
01005                   write(6,"('iii --- EigenValues for zw --------')")
01006                   allocate(ebb(ngb))
01007                   call diagcvh2((zw(1:ngb,1:ngb)-transpose(dconjg(zw(1:ngb,1:ngb))))/2d0/img,
01008      &             ngb, ebb)
01009                   do ii=1,ngb
01010                     if(abs(ebb(ii))>1d-8.and.ebb(ii)>0) then
01011                       write(6,"('iii1xxx:  iw ii eb=',2i4,d13.5)") ix,ii,ebb(ii)
01012                     else
01013                       write(6,"('iii1:  iw ii eb=',2i4,d13.5)") ix,ii,ebb(ii)
01014                     endif
01015                   enddo
01016                   deallocate(ebb)
01017                 endif
01018               enddo
01019               deallocate(zw)
01020             else
01021               nstatex= max(ntp0,nt_max)
01022               if(allocated(zwz)) deallocate(zwz)
01023               allocate( zwz(nwxi:nwx,1:nstatex,ntp0) )
01024               do      ix = nwxi,nwx
01025                 nrec= ix-nw_i+1
01026                 read(ifrcw,rec=nrec) zw ! Readin (W-v)(k,w')(i,j) at k and w' on imag axis
01027 c zwz = S[i,j] <psi(q,t) |psi(q-rk,n) B(rk,i)> Wc(k,iw')(i,j) > <B(rk,j) psi(q-rk,n) |psi(q,t)>
01028                 call matzwz(zw(1:ngb,1:ngb), zmel(1:ngb,1:nstatex,1:ntp0), ntp0,nstatex,ngb,
01029      o           zwz(ix,1:nstatex,1:ntp0))
01030 ! zmel (ngb, nstate, ntp0)
01031               enddo
01032               deallocate(zmel)
01033               deallocate(zw)
01034             endif
01035 c-----------------------------------------------
01036             if(screen) then
01037               if(zwz3mode) call rx( ' this mode is not implimented')
01038               do ix = nw_i,nwx
01039                 zwz(ix,:,:)=zwz(ix,:,:) - zwz00
01040               enddo
01041               deallocate(zwz00)
01042             endif
01043
01044 c-------------------------------
01045 c loop over w and t in SEc(qt,w)
01046 c-------------------------------
01047             if(debug) write(6,*)' sss ngb, nstate, ntp0=',ngb,nstate,ntp0
01048             if(debug) write(6,*)' sss zmel=',sum(abs(zmel(:,:,:)))
01049
01050             if(verbose()>50) write(*,'("10 wfacx  iw,itp,it cycles ",$)')
01051             if(verbose()>50) call cputid(0)
01052             do 2001 iw  = iwini,iwend
01053               do 2002 itp = 1,ntq
01054                 if(debug) write(6,*)'2011 0 zmel=',sum(abs(zmel(:,:,:)))
01055                 omg = omega(itp,iw)
01056                 if (omg >= ef) then
01057                   itini= nt0m+1
01058                   itend= nt_max
01059                   iii=  1
01060                 else
```

```
01061                  itini= 1
01062                  itend= nt0p
01063                  iii= -1
01064                endif
01065
01066                do 2011 it= itini,itend
01067                  if(debug) write(6,*)'2011 1 loop--- it=',iw,itp,it,sum(abs(zmel(:,:,:)))
01068                  esmrx = esmr
01069                  if(it<=nctot) esmrx = 0d0
01070                  wfac = wfacx2(omg,ef, ekc(it),esmrx)
01071                  if(gausssmear) then
01072                    if(wfac<wfaccut) cycle
01073                    we = .5d0*abs(omg-weavx2(omg,ef, ekc(it),esmr))
01074                  else
01075                    if(wfac==0d0) cycle
01076                    if(omg>=ef) we = 0.5d0* abs(max(omg-ekc(it), 0d0)) ! positive
01077                    if(omg< ef) we = 0.5d0* abs(min(omg-ekc(it), 0d0)) ! negative
01078                  endif
01079
01080                  wfac= iii* wfac*wtt
01081 c three-point interpolation for Wc(we)
01082                  do iwp = 1,nw
01083                    ixs=iwp
01084                    if(freq_r(iwp)>we) exit
01085                  enddo
01086                  if(nw_i==0) then
01087                    if(ixs+1>nwx) then
01088                      write(6,*)' ixs,nwx, we =',ixs,nwx,we
01089                      call rx( ' sxcf: ixs+1>nwx xxx2')
01090                    endif
01091                  else          !   write(6,*)" ixs nwxi=",ixs,nwxi,freq_r(ixs-1),we,freq_r(ixs)
01092                    if(omg >=ef .and. ixs+1> nwx ) then
01093                      write(6,*)'ixs+1 nwx=',ixs+1,nwx
01094                      call rx( ' sxcf: ixs+1>nwx yyy2a')
01095                    endif
01096                    if(omg < ef .and. abs(ixs+1)> abs(nwxi) ) then
01097                      write(6,*)'ixs+1 nwxi=',ixs+1,nwxi
01098                      call rx( ' sxcf: ixs-1<nwi yyy2b')
01099                    endif
01100                  endif
01101
01102                  iir=1
01103                  if(omg < ef .and. nw_i/=0) iir = -1 !May2006 because of \int d omega' G(omega-omega')
       W(omega')
01104                  if(zwz3mode) then
01105                    zwz3=(0d0,0d0)
01106                    if(debug) write(6,"('wwwwwww ixs=',10i4)"),ixs,igb2,it,itp
01107                    if(debug) write(6,*)'2011 www zmel aaa=',sum(abs(zmel(:,:,:)))
01108                    do ix = ixs, ixs+2
01109                      do igb2=1,ngb
01110                        zz2 = sum(dconjg(zmel(1:ngb,it,itp))*zw3(1:ngb,igb2,iir*(ix-1)) )
01111                        zwz3(ix-ixs+1) = zwz3(ix-ixs+1)+zz2 *zmel(igb2,it,itp)
01112                      enddo     !igb2
01113                    enddo       !ix
01114                    if(debug) write(6,"('w xxxxxxxxxxxxx ixs loopend=',i4)"),ixs
01115                    if(debug) write(6,*)zwz3(1:3) !,freq_r(ixs-1),zwz3(1:3)
01116                    if(debug) write(6,*)'we frez zwz3=', we,ixs,freq_r(ixs-1:ixs+1)
01117                    if(debug) write(6,*)'2011 bbb www zmel=',sum(abs(zmel(:,:,:)))
01118
01119                    zsec(iw,itp,ip) = zsec(iw,itp,ip)
01120      &              + wfac *alagr3zz(we,freq_r(ixs-1),zwz3) !faleev
01121
01122                    if(debug) write(6,*)'2011 ccc www zmel=',sum(abs(zmel(:,:,:)))
01123                    if(debug) write(6,"('wwwwwww eo zsecsum')")
01124                  else
01125                    zwzz(1:3) = zwz(iir*(ixs-1):iir*(ixs+1):iir, it,itp)
01126                    zsec(iw,itp,ip) = zsec(iw,itp,ip)
01127      &              + wfac*alagr3zz(we,freq_r(ixs-1),zwzz)
01128                  endif
01129  2011          continue
01130  2002        continue          !end of SEc w and qt -loop
01131  2001      continue            !end of SEc w and qt -loop
01132            if(debug) write(6,*)' end of do 2001'
01133            if(verbose()>50) then
01134              write(*,'("11 after alagr3zz iw,itp,it cycles ",$)')
01135              call cputid(0)
01136            endif
01137            if(debug) then
01138              do itp = 1,ntq
01139                write(6,'(" zsec=",i3,6d15.7)') itp,zsec(iwini:iwini+2,itp,ip)
01140              enddo
01141            endif
01142            if(zwz3mode) then
01143              deallocate(zmel,zw3)
01144            else
01145              deallocate(zwz)
01146            endif
```

```
01147  1000     continue
01148 c      if(newaniso) ifvcoud =iclose('Vcoud.'//charnum5(kx))
01149           ifvcoud =iclose('Vcoud.'//charnum5(kx))
01150           if(.not.exchange) then
01151             ifrcw  = iclose('WVR.'//charnum5(kx))
01152             ifrcwi = iclose('WVI.'//charnum5(kx))
01153           endif
01154  1100   continue                ! end of k-loop
01155          if (tote) then
01156            exx = exx + wik(ip) * exxq * 0.25d0
01157          endif
01158          if (allocated(zz)) deallocate(zz)
01159          if (allocated(zmel)) deallocate(zmel)
01160          if (allocated(zzmel))deallocate(zzmel)
01161          if (allocated(zw)) deallocate(zw)
01162          if (allocated(zwz)) deallocate(zwz)
01163          if (allocated(zwz0)) deallocate(zwz0)
01164          if (allocated(zwzi)) deallocate(zwzi)
01165          if (allocated(zwz00)) deallocate(zwz00)
01166          if (allocated(w1p)) deallocate(w1p)
01167          if (allocated(w2p)) deallocate(w2p)
01168          if (allocated(w3p)) deallocate(w3p)
01169          if (allocated(z1p)) deallocate(w1p)
01170          if (allocated(vcoul)) deallocate(vcoul)
01171          if (allocated(vcoult)) deallocate(vcoul)
01172 c      if (allocated(zmel1)) deallocate(zmel1)
01173          if (allocated(zmel3)) deallocate(zmel3)
01174          if (allocated(zw_)) deallocate(zw_)
01175          if (allocated(zwz2)) deallocate(zwz2)
01176 c      if (allocated(zw2)) deallocate(zw2)
01177          if (allocated(zmel2)) deallocate(zmel2)
01178          if (allocated(zw3)) deallocate(zw3)
01179          if (allocated(uaa)) deallocate(uaa)
01180  1001 continue
01181 c       if (allocated(expikt)) deallocate(expikt)
01182        end
```

4.25 gwsrc/sxcf_fal2.sc.F File Reference

Data Types

- module m_sxcfsc

  this module is only because name=name argument binding. No data

Functions/Subroutines

- subroutine get_nwx (omega, ntq, ntqxx, nt0p, nt0m, nstate, freq_r, nw_i, nw, esmr, ef, ekc, wfaccut, nctot, nband, debug, nwxi, nwx, nt_max)

4.25.1 Function/Subroutine Documentation

4.25.1.1 subroutine get_nwx (real(8), dimension(ntq), intent(in) *omega,* integer, intent(in) *ntq,* integer, intent(in) *ntqxx,* integer, intent(in) *nt0p,* integer, intent(in) *nt0m,* integer, intent(in) *nstate,* real(8), dimension(nw_i:nw), intent(in) *freq_r,* integer, intent(in) *nw_i,* integer, intent(in) *nw,* real(8), intent(in) *esmr,* real(8), intent(in) *ef,* real(8), dimension(nctot+nband), intent(in) *ekc,* real(8), intent(in) *wfaccut,* integer, intent(in) *nctot,* integer, intent(in) *nband,* logical *debug,* integer, intent(out) *nwxi,* integer, intent(out) *nwx,* integer, intent(out) *nt_max*)

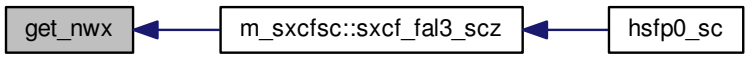
Parameters

in	*nctot*	Determine indexes of a range for calculation. It is better to clean this up...
in	*nw_i*	Determine indexes of a range for calculation. It is better to clean this up...
in	*nw*	Determine indexes of a range for calculation. It is better to clean this up...

in	*nstate*	Determine indexes of a range for calculation. It is better to clean this up...
in	*nt0p*	Determine indexes of a range for calculation. It is better to clean this up...
in	*nt0m*	Determine indexes of a range for calculation. It is better to clean this up...
in	*ntq*	Determine indexes of a range for calculation. It is better to clean this up...
in	*nband*	Determine indexes of a range for calculation. It is better to clean this up...
in	*ntqxx*	Determine indexes of a range for calculation. It is better to clean this up...

Definition at line 1306 of file sxcf_fal2.sc.F.

Here is the caller graph for this function:



4.26 sxcf_fal2.sc.F

```
00001 !> this module is only because name=name argument binding. No data
00002       module m_sxcfsc
00003       contains
00004       subroutine sxcf_fal3_scz(kount,qip,itq,ntq,ef,esmr,
00005      i nsp,isp,
00006      i qbas,ginv,
00007      i qibz,qbz,wk,nstbz,irkip,nrkip,
00008      i freq_r,nw_i,nw, freqx,wx,dw,
00009      i ecore,
00010      i nlmto,nqibz,nqbz,nctot,
00011      i nbloch,ngrp,niw,nq,
00012      i nblochpmx ,ngpmx,ngcmx,
00013      i wgt0,nq0i,q0i,symgg, alat, nband, ifvcfpout,
00014      i exchange,screen,cohtest, ifexsp,
00015      i nbmx,ebmx,
00016      i wklm,lxklm,
00017      i eftrue,
00018      i jobsw,                    != iSigma_en
00019      i hermitianw,
00020      o zsec,coh,nbandmx)
00021       use m_readqg,only   : readqg0
00022       use m_readeigen,only: readeval
00023       use keyvalue,only   : getkeyvalue
00024       use m_zmel,only     : get_zmelt,
00025      i ppovlz,
00026      o zmel,zmeltt
00027       implicit none
00028 !> \brief
00029 !! Calcualte full simga_ij(e_i)= <i|Re[Sigma](e_i)|j>
00030 !! ---------------------
00031 !! \param exchange
00032 !!   - T : Calculate the exchange self-energy
00033 !!   - F : Calculate correlated part of the self-energy
00034 !! \param zsec
00035 !!   - S_ij= <i|Re[S](e_i)|j>
00036 !!   - Note that S_ij itself is not Hermite becasue it includes e_i.
00037 !!     i and j are band indexes
00038 !! \param coh dummy
00039 !! \param screen dummy
00040 !!
00041 !! \remark
00042 !! \verbatim
00043 !! Jan2013: eftrue is added.
00044 !!   ef=eftrue(true fermi energy) for valence exchange and correlation mode.
00045 !!   but ef is not the true fermi energy for core-exchange mode.
00046 !!
00047 !! Jan2006
00048 !!     "zsec from im-axis integral part"  had been symmetrized as
00049 !!     &        wtt*.5d0*(   sum(zwzi(:,itp,itpp))+ !S_{ij}(e_i)
00050 !!     &        dconjg( sum(zwzi(:,itpp,itp)) )    ) !S_{ji}^*(e_j)= S_{ij}(e_j)
00051 !!     However, I now do it just the 1st term.
00052 !!     &        wtt* sum(zwzi(:,itp,itpp))   !S_{ij}(e_i)
```

```
00053 !!     This is OK because the symmetrization is in hqpe.sc.F
00054 !!     Now zsec given in this routine is simply written as <i|Re[S](e_i)|j>.
00055 !!     ( In the version until Jan2006 (fpgw032f8), only the im-axis part was symmetrized.
00056 !!     But it was not necessary from the begining because it was done in hqpe.sc.F
00057 !!
00058 !!     (Be careful as for the difference between
00059 !!     <i|Re[S](e_i)|j> and transpose(dconjg(<i|Re[S](e_i)|j>)).
00060 !!     ---because e_i is included.
00061 !!     The symmetrization (hermitian) procedure is inlucded in hqpe.sc.F
00062 !!
00063 !!     NOTE: matrix element is given by "call get_zmelt". It returns  zmelt or zmeltt.
00064 !!
00065 !! jobsw switch
00066 !!  1-5 scGW mode.
00067 !!   diag+@EF      jobsw==1 SE_nn'(ef)+delta_nn'(SE_nn(e_n)-SE_nn(ef))
00068 !!   xxx modeB (Not Available now)  jobsw==2 SE_nn'((e_n+e_n')/2)  !we need to recover comment out for
      jobsw==2, and test.
00069 !!   mode A        jobsw==3 (SE_nn'(e_n)+SE_nn'(e_n'))/2 (Usually usued in QSGW).
00070 !!   @Ef           jobsw==4 SE_nn'(ef)
00071 !!   diagonly      jobsw==5 delta_nn' SE_nn(e_n) (not efficient memoryuse; but we don't use this mode so
      often).
00072 !!
00073 !! Output file in hsfp0 should contain hermitean part of SE
00074 !!    ( hermitean of SE_nn'(e_n) means SE_n'n(e_n')^* )
00075 !!             we use that zwz(itp,itpp)=dconjg( zwz(itpp,itp) )
00076 !! Caution! npm=2 is not examined enough...
00077 !!
00078 !! Calculate the exchange part and the correlated part of self-energy.
00079 !! T.Kotani started development after the analysis of F.Aryasetiawan's LMTO-ASA-GW.
00080 !! We still use some of his ideas in this code.
00081 !!
00082 !! See paper
00083 !! [1]T. Kotani and M. van Schilfgaarde, ??Quasiparticle self-consistent GW method:
00084 !!     A basis for the independent-particle approximation, Phys. Rev. B, vol. 76, no. 16, p.
      165106[24pages], Oct. 2007.
00085 !! [2]T. Kotani, Quasiparticle Self-Consistent GW Method Based on the Augmented Plane-Wave
00086 !!    and Muffin-Tin Orbital Method, J. Phys. Soc. Jpn., vol. 83, no. 9, p. 094711 [11 Pages], Sep. 2014.
00087 !!
00088 !! ========================================================================
00089 !! Omega integral for SEc
00090 !!   The integral path is deformed along the imaginary-axis, but together with contribution of poles.
00091 !!   See Fig.1 and around in Ref.[1].
00092 !!
00093 !! ---Integration along imaginary axis.---
00094 !!   ( Current version for it, wintzsg_npm, do not assume time-reversal when npm=2.)
00095 !!   Integration along the imaginary axis: -----------------
00096 !!    (Here is a memo by F.Aryasetiawan.)
00097 !!     (i/2pi) < [w'=-inf,inf] Wc(k,w')(i,j)/(w'+w-e(q-k,n) >
00098 !!    Gaussian integral along the imaginary axis.
00099 !!    transform: x = 1/(1+w')
00100 !!     this leads to a denser mesh in w' around 0 for equal mesh x
00101 !!    which is desirable since Wc and the lorentzian are peaked around w'=0
00102 !!     wint = - (1/pi) < [x=0,1] Wc(iw') (w-e)x^2/{(w-e)^2 + w'^2} >
00103 !!
00104 !!     the integrand is peaked around w'=0 or x=1 when w=e
00105 !!     to handel the problem, add and substract the singular part as follows:
00106 !!     wint = - (1/pi) < [x=0,1] { Wc(iw') - Wc(0)exp(-a^2 w'^2) }
00107 !!     * (w-e)/{(w-e)^2 +w'^2}x^2 >
00108 !!     - (1/2) Wc(0) sgn(w-e) exp(a^2 (w-e)^2) erfc(a|w-e|)
00109 !!
00110 !!     the second term of the integral can be done analytically, which
00111 !!     results in the last term a is some constant
00112 !!
00113 !!     when w = e, (1/pi) (w-e)/{(w-e)^2 + w'^2} ==> delta(w') and
00114 !!     the integral becomes -Wc(0)/2
00115 !!     this together with the contribution from the pole of G (s.u.)
00116 !!     gives the so called static screened exchange -Wc(0)
00117 !!
00118 !! ---Integration along real axis (contribution from the poles of G: SEc(pole))
00119 !!    See Eq.(34),(55), and (58) and around in Ref.[1]. We now use Gaussian Smearing.
00120 !! ========================================================================
00121 !
00122 !! -----------------------------------------------
00123 !!     q     =qip(:,iq)  = q-vector in SEc(q,t).
00124 !!    itq     = states t at q
00125 !!    ntq     = no. states t
00126 !!    eq      = eigenvalues at q
00127 !!     ef      = fermi level in Rydberg
00128 !!   WVI, WVR: direct access files for W. along im axis (WVI) or along real axis (WVR)
00129 !!   freq_r(nw_i:nw)   = frequencies along real axis. freq_r(0)=0d0
00130 !!
00131 !!    qbas    = base reciprocal lattice vectors
00132 !!    ginv    = inverse of qbas s. indxrk.f
00133 !!
00134 !!     wk     = weight for each k-point in the FBZ
00135 !!    qbz     = k-points in the 1st BZ
00136 !!
```

```
00137 !!    wx      = weights at gaussian points x between (0,1)
00138 !!     ua_      = constant in exp(-ua^2 w'^2) s. wint.f
00139 !!     expa    = exp(-ua^2 w'^2) s. wint.f
00140 !!
00141 !!    irkip(k,R,nq) = gives index in the FBZ with k{IBZ, R=rotation
00142 !!
00143 !!   nqibz   = number of k-points in the irreducible BZ
00144 !!   nqbz    =                          full BZ
00145 !!    natom   = number of atoms
00146 !!    nctot   = total no. of allowed core states
00147 !!    nbloch  = total number of Bloch basis functions
00148 !!    nlmto   = total number of MTO+lo basis functions
00149 !!    ngrp    = no. group elements (rotation matrices)
00150 !!    niw     = no. frequencies along the imaginary axis
00151 !!    nw_i:nw  = no. frequencies along the real axis. nw_i=0 or -nw.
00152 !!    zsec(itp,itpp,iq)> = <psi(itp,q(:,iq)) |SEc| psi(iq,q(:,iq)>
00153 !!
00154 !! ------------------------------------------
00155 !! \endverbatim
00156       integer:: dummy4doxygen
00157
00158 ! input variables
00159       logical, intent(in) :: exchange,screen,cohtest
00160       integer, intent(in) :: ntq,nqbz,nqibz,ngrp,nq,niw !,natom
00161       integer, intent(in) :: nband,nlmto,nq0i,nctot,isp,nsp !,mdim(*) !,nlnmx
00162       integer, intent(in) :: ifvcfpout,nbloch,nblochpmx !nl,nnc, nclass
00163       integer, intent(in) :: itq(ntq) !,nstar(nqibz) !miat(natom,ngrp),mdimx,
00164       integer, intent(in) :: irkip(nqibz,ngrp,nq),nrkip(nqibz,ngrp,nq)
00165       integer, intent(in) :: kount(nqibz,nq),ngpmx,ngcmx,ifexsp,jobsw
00166       integer, intent(in) :: nbmx(2) !,nlnmv(*),nlnmc(*)!,iclass(*),icore(*)
00167       integer, intent(in) :: nstbz(nqbz) !,nomx !,nkpo,nnmx,imdim(*)ncore(*),
00168       integer, intent(in) :: lxklm !,invg(ngrp) !nnr(nkpo),nor(nkpo),
00169 c      integer, intent(in) :: il(*),in(*),im(*),nn_,lx(*),nx_(*),nxx_ !,nlnm(*)
00170       real(8), intent(in) :: wgt0(nq0i,ngrp),symgg(3,3,ngrp)
00171       real(8), intent(in) :: q0i(1:3,1:nq0i),alat,ecore(nctot) !shtvg(3,ngrp),
00172       real(8), intent(in) :: qbas(3,3),ginv(3,3)
00173       real(8), intent(in) :: wk(nqbz),qibz(3,nqibz) !tiat(3,natom,ngrp),
00174       real(8), intent(in) :: qbz(3,nqbz),freqx(niw),wx(niw),ef,esmr,dw
00175       real(8), intent(in) :: ebmx(2),wklm((lxklm+1)**2) !,qrr(3,nkpo)
00176       real(8), intent(in) :: qip(3,nq),eftrue
00177
00178 c      integer,intent(in):: iwini,iwend
00179 c      real(8),optional::exx
00180
00181 ! output variables
00182 c      real(8),intent(in),optional:: freqsig(iwini:iwend)
00183       integer, intent(in) ::nbandmx(nq)
00184       complex(8), intent(out),optional :: zsec(ntq,ntq,nq) , coh(ntq,nq)
00185 c      complex(8), intent(out),optional :: zsecd(iwini:iwend,ntq,nq)
00186
00187 ! local variables
00188 c     complex(8) :: zsecx(ntq,ntq,nq)
00189 c      complex(8), intent(in) :: pomatr(nnmx,nomx,nkpo)
00190 c$$$      logical :: ua_auto !fixed to be .false.
00191 c      real(8)::ppbrd ( 0:nl-1, nn_, 0:nl-1,nn_, 0:2*(nl-1),1:nxx_, 1:nsp*nclass)
00192
00193       integer :: ifrcw,ifrcwi
00194       logical :: initp=.true.
00195       real(8),allocatable:: vcoud(:)
00196
00197       integer :: ip, it, itp, i, ix, kx, irot, kr
00198       integer :: nt0p, nt0m,nstate , nbmax, ntqxx !iatomp(natom),
00199       integer :: nt,nw,ixs,iw,ivc,ifvcoud,ngb0
00200       integer :: nprecx,mrecl,ifwd,nrot,nwp,nw_i,ierr
00201       integer :: nstatetot,iqini,iqend, ngb,ngc !nbcut,
00202       integer :: invr,nbmxe,ia,nn,ntp0,no,itpp,nrec,npm,itini,itend
00203       integer :: iwp,nwxi,nwx,iir, igb1,igb2,ix0,iii
00204
00205       real(8) :: tpi, ekc(nctot+nband),ekq(nband), det, q(3),ua_
00206       real(8) :: expa_(niw), qxx(3), symope(3,3),shtv(3) !tr(3,natom),
00207       real(8) :: efp,efm,wtt,wfac,we,esmrx,qbasinv(3,3)
00208       real(8) :: qvv(3),pi,fpi,eq(nband),omega(ntq),quu(3),freqw,ratio
00209       real(8) :: qibz_k(3),qbz_kr(3),ddw,vc,omega0,omg
00210
00211       complex(8) :: cphiq(nlmto,nband), cphikq(nlmto,nband)
00212       complex(8) :: zwzs0,zz2,zwz3(3)
00213
00214 ! local arrays
00215       real(8),intent(in) :: freq_r(nw_i:nw)
00216       real(8),allocatable :: drealzzzmel(:,:,:), dimagzzzmel(:,:,:),uaa(:,:)
00217       complex(8),allocatable :: vcoul(:,:),w3p(:,:,:)
00218       complex(8),allocatable :: zzzmel(:,:,:),zw (:,:)
00219       complex(8),allocatable :: zwz(:,:,:,:), zwz0(:,:,:),zwzi(:,:,:)
00220       complex(8),allocatable :: zwix(:,:),zwzix(:,:,:),zmel1(:) !,expikt(:)
00221       complex(8), allocatable :: zmel1_(:,:,:), zw3(:,:,:),zw3x(:,:)
00222       complex(8), allocatable :: zwz4(:,:),zwz44(:,:),pomat(:,:), zwzs(:)
00223       complex(8),allocatable :: ppovl(:,:),zcousq(:,:)
```

```
00224       complex(8),allocatable :: z1r(:,:),z2r(:,:),w3pi(:,:)
00225
00226       real(8), parameter :: wfaccut=1d-8
00227       complex(8), parameter :: img=(0d0,1d0)
00228
00229 ! external function
00230 c      logical :: smbasis
00231 c      logical :: test_symmetric_W
00232 c      logical :: GaussSmear !fixed to be T
00233 c      logical :: newaniso !fixed to be T
00234 c      integer :: bzcase !fixed to be 1
00235       character(5) :: charnum5
00236       integer :: iopen,iclose
00237       integer :: invrot
00238       complex(8) ::  wintzsg_npm !wintzav,
00239       integer :: nocc
00240       real(8) :: wfacx
00241       real(8) :: wfacx2
00242       real(8) :: weavx2
00243       complex(8) :: alagr3z
00244       complex(8) :: alagr3z2
00245
00246       integer:: ndummy1,ndummy2,nlmtobnd,nt0
00247       real(8):: wexx
00248 c      complex(8),allocatable :: z1p(:,:,:),vcoult(:,:)
00249       logical :: debug, debugp,debug2=.false.
00250 c      logical :: gass           !external
00251 c      real(8):: wgtq0p
00252       integer::verbose,ififr,ifile_handle
00253       real(8):: ua2_(niw),freqw1
00254       integer :: istate,  nt_max !nbcutc,nbcutin,
00255       real(8):: q_r(3),qk(3),omegat
00256       logical::  oncew, onceww, eibz4sig,  timemix
00257
00258       integer,allocatable:: ixss(:,:),iirx(:)
00259       real(8),allocatable:: we_(:,:),wfac_(:,:)
00260       complex(8),allocatable:: zw3av(:,:),zmelw(:,:,:)
00261       integer:: noccx
00262       real(8)::polinta
00263       logical,allocatable:: ititpskip(:,:)
00264
00265       logical:: tote=.false.
00266       logical:: hermitianw
00267
00268       real(8),allocatable:: wcorehole(:,:)
00269       logical:: corehole
00270       integer:: ifcorehole
00271 c      real(8),allocatable:: ppb(:)
00272 c      allocate( ppb(nlnmx*nlnmx*mdimx*nclass))
00273
00274 c      real(8)::exxq
00275
00276 c-----------------------------------------------------------------------
00277 c!TIME0_0000
00278 c      write(6,*)'sxcf_fal3_scz'
00279       timemix=.false.
00280       pi  = 4d0*datan(1d0)
00281       fpi = 4d0*pi
00282       debug=.false.
00283       if(verbose()>=90) debug=.true.
00284
00285 ccccccccccccccc
00286       corehole=.false.
00287 ccccccccccccccc
00288
00289 !! core-hole
00290       if(corehole) then
00291         ifcorehole=ifile_handle()
00292         open(ifcorehole,file='CoreHole')
00293         if(allocated(wcorehole)) deallocate(wcorehole)
00294         allocate(wcorehole(nctot,nsp))
00295         do it=1,nctot
00296           read(ifcorehole,*) wcorehole(it,1:nsp)
00297         enddo
00298         close(ifcorehole)
00299       endif
00300
00301       if(.not.exchange) then
00302         ifwd = iopen('WV.d',1,-1,0)
00303         read (ifwd,*) nprecx,mrecl
00304         ifwd = iclose('WV.d')
00305 !! gauss_img : interpolation gaussion for W(i \omega).
00306         call getkeyvalue("GWinput","gauss_img",ua_,default=1d0)
00307         if(debug) write(6,*) ' sxcf_fal3_scz: Gausssmear=T'
00308         do ix = 1,niw           !! Energy mesh; along im axis.
00309           freqw     = (1d0 - freqx(ix))/ freqx(ix)
00310           expa_(ix) = exp(-(ua_*freqw)**2)
```

```
00311         enddo
00312         npm = 1                 ! npm=1    Timeveversal case
00313         if(nw_i/=0) npm = 2     ! npm=2 No TimeReversal case. Need negative energy part of W(omega)
00314       endif
00315
00316 c      call getkeyvalue("GWinput","nbcutlow_sig",nbcut, default=0 )
00317 c      nbcutc=nctot+nbcut
00318       tpi         = 8d0*datan(1d0)
00319       if(nctot/=0) ekc(1:nctot)= ecore(1:nctot) ! core
00320       nlmtobnd    = nlmto*nband
00321       nstatetot      = nctot + nband
00322
00323
00324 !!== ip loop to spedify external q ==
00325       do 1001 ip = 1,nq
00326         if(sum(irkip(:,:,ip))==0) cycle ! next ip
00327         write (6,*) ip,'  out of ',nq,'  k-points(extrnal q) '
00328         q(1:3)= qip(1:3,ip)
00329         call readeval(q,isp,eq)
00330         do i  = 1,ntq
00331           omega(i) = eq(itq(i))
00332         enddo
00333
00334 !! we only consider bzcase()==1
00335         if(abs(sum(qibz(:,1)**2))/=0d0) call rx( ' sxcf assumes 1st qibz/=0 ')
00336         if(abs(sum( qbz(:,1)**2))/=0d0) call rx( ' sxcf assumes 1st qbz /=0 ')
00337
00338 !! NOTE total number of
00339 !!    kx loop(do 1100) and irot loop (do 1000) makes all the k mesh points.
00340 !!    When iqini=1 (Gamma point), we use effective W(q=0) defined in the paper.
00341         iqini=1
00342         iqend=nqibz             !no sum for offset-Gamma points.
00343         do 1100 kx = iqini,iqend
00344           if(sum(irkip(kx,:,ip))==0) cycle ! next kx
00345 !TIME0_01000
00346           write(6,*) ' ### do 1100 start kx=',kx,' from ',iqini,' through', iqend
00347 c          if( kx <= nqibz ) then
00348             qibz_k= qibz(:,kx)
00349 c          else
00350 c            qibz_k= 0d0
00351 c          endif
00352           if(timemix) call timeshow("11111 k-cycle")
00353           call readqg0('QGcou',qibz_k,ginv,  quu,ngc)
00354           ngb = nbloch + ngc
00355           if(debug) write(6,*) ' sxcf: ngb=',ngb,nbloch
00356
00357 !! ===Readin diagonalized Coulomb interaction===
00358 !!  Vcoud file is sequential file Vcoulomb matrix for qibz_k.
00359 !!  A possible choice for paralellization is "Vcoud.ID" files where ID=kx
00360 !!  Vould file is written in hvccfp0.m.F.
00361 !! For correlation, W-v is read instead of Vcoud file (ifrcw,ifrcwi for WVR and WVI)
00362 !! These can be also separeted into WVR.ID and WVI.ID files.
00363 !! NOTE: vcoud and zcousq are in module m_zmelt.
00364           qxx=qibz_k
00365 c          if(kx<=nqibz) qxx=qibz_k
00366 c          if(kx>nqibz ) qxx=q0i(:,kx-nqibz)
00367           ifvcoud = iopen('Vcoud.'//charnum5(kx),0,0,0)
00368           do
00369             read(ifvcoud) ngb0
00370             read(ifvcoud) qvv
00371             if(allocated(vcoud)) deallocate(vcoud)
00372             allocate( zcousq(ngb0,ngb0),vcoud(ngb0) )
00373             read(ifvcoud) vcoud
00374             read(ifvcoud) zcousq
00375             if(sum(abs(qvv-qxx))<1d-6) goto 1133
00376           enddo
00377           if(sum(abs(qvv-qxx))>1d-6) then
00378             write(6,*)'qvv =',qvv
00379             write(6,*)'qxx=',qxx,kx
00380             call rx( 'sxcf_fal2: qvv/=qibz(:,kx) hvcc is not consistent')
00381           endif
00382  1133     continue
00383           if( ngb0/=ngb ) then  !sanity check
00384             write(6,*)' qxx ngb0 ngb=',qxx,ngb0,ngb
00385             call rx( 'hsfp0.m.f:ngb0/=ngb')
00386           endif
00387 !! ppovlz is used in get_zmel
00388 !! <I|v|J>= \sum_mu ppovl*zcousq(:,mu) v^mu (Zcousq^*(:,mu) ppovl)
00389 !! zmel contains O^-1=<I|J>^-1 factor. zmel(phi phi J)= <phi phi|I> O^-1_IJ
00390 !! ppovlz= O Zcousq
00391 !! (V_IJ - vcoud_mu O_IJ) Zcousq(J, mu)=0, where Z is normalized with O_IJ.
00392           allocate(ppovl(ngc,ngc),ppovlz(ngb,ngb))
00393           call readppovl0(qibz_k,ngc,ppovl)
00394           ppovlz(1:nbloch,:) = zcousq(1:nbloch,:)
00395           ppovlz(nbloch+1:nbloch+ngc,:)=matmul(ppovl,zcousq(nbloch+1:nbloch+ngc,:))
00396 c             write(6,*)'sumcheck ppovlz 00000 =',sum(abs(ppovlz(:,:)))
00397           deallocate(zcousq,ppovl)
```

```
00398 !! === open WVR,WVI ===
00399           if(.not.exchange) then
00400             ifrcw  = iopen('WVR.'//charnum5(kx),0,-1,mrecl)
00401             ifrcwi = iopen('WVI.'//charnum5(kx),0,-1,mrecl)
00402           endif
00403           nrot=0
00404           do irot = 1,ngrp
00405 c            if( kx <= nqibz) then
00406              kr = irkip(kx,irot,ip) ! index for rotated kr in the FBZ
00407              if(kr==0) cycle   ! next irot
00408              qbz_kr= qbz(:,kr)
00409 c            else
00410 c              kr=-99999         !for sanity check
00411 c              qbz_kr= 0d0
00412 c              if( wgt0(kx-nqibz,irot)==0d0 ) cycle ! next irot
00413 c            endif
00414             nrot=nrot+1
00415           enddo
00416 !TIME1_01000 ":BeforDo1000"
00417
00418
00419 !! === loop 1000 over rotations irot ===
00420           do 1000 irot = 1,ngrp
00421 c            if( kx <= nqibz) then
00422              kr = irkip(kx,irot,ip) ! index for rotated kr in the FBZ
00423              if(kr==0) cycle
00424              qbz_kr= qbz(:,kr)
00425 c            else
00426 c              kr=-99999         !for sanity check
00427 c              qbz_kr= 0d0
00428 c              if( wgt0(kx-nqibz,irot)==0d0 ) cycle
00429 c            endif
00430
00431 !TIME0_1010
00432 !! no. occupied (core+valence) and unoccupied states at q-rk
00433             qk =  q - qbz_kr
00434             call readeval(qk, isp, ekq)
00435             ekc(nctot+1:nctot+nband) = ekq(1:nband)
00436             nt0 = nocc(ekc,ef,.true.,nstatetot)
00437             ddw= .5d0
00438 c            if(GaussSmear()) ddw= 10d0
00439             ddw= 10d0
00440             efp= ef+ddw*esmr
00441             efm= ef-ddw*esmr
00442             nt0p = nocc(ekc,efp,.true.,nstatetot)
00443             nt0m = nocc(ekc,efm,.true.,nstatetot)
00444 !! nbmx1 ebmx1: to set how many bands of <i|sigma|j>  do you calculate.
00445 !! nbmx2 ebmx2: to restrict num of bands of G to calculate G \times W
00446             if(exchange) then
00447               nbmax = nt0p-nctot
00448             else
00449               nbmax = nband
00450               nbmxe = nocc(ekc,ebmx(2),.true.,nstatetot)-nctot
00451               nbmax  = min(nband,nbmx(2),nbmxe)
00452               if(initp) then
00453                 write(6,*)' nbmax=',nbmax
00454                 initp=.false.
00455               endif
00456             endif
00457 c$$$!! ntqxx is number of bands for <i|sigma|j>.
00458 c$$$            ntqxx = nocc (omega-eftrue,ebmx(1),.true.,ntq)
00459 c$$$!bug -ef is added jan2013
00460 c$$$!previous version do not give wrong results, but inefficient.
00461 c$$$            ntqxx = min(ntqxx, nbmx(1))
00462 c$$$            if(ntqxx<nband) then
00463 c$$$              do i=ntqxx,1,-1   !redudce ntqxx when band tops are degenerated. !sep2012
00464 c$$$                if(omega(i+1)-omega(i)<1d-2) then
00465 c$$$                  ntqxx=i-1
00466 c$$$                else
00467 c$$$                  exit
00468 c$$$                endif
00469 c$$$              enddo
00470 c$$$            endif
00471 c$$$            nbandmx(ip)=ntqxx   !number of bands to be calculated Sep2012.
00472
00473             ntqxx = nbandmx(ip) !mar2015
00474             if(debug) write(6,*)' sxcf: nbmax nctot nt0p =',nbmax,nctot,nt0p
00475             nstate = nctot + nbmax ! = nstate for the case of correlation
00476
00477 !! Get matrix element zmelt= rmelt + img*cmelt, defined in m_zmel.F---
00478 c             if(debug) write(6,*)'zzBBB ppovlz =',sum(abs(ppovlz(:,:))),kx,irot
00479             if(allocated(zmel)) deallocate(zmel)
00480             if(allocated(zmeltt)) deallocate(zmeltt)
00481 !TIME1_1010 "Beforeget_zmelt"
00482 ! this return zmeltt (for exchange), or zmel (for correlation)
00483 !TIME0_1088
00484             call get_zmelt(exchange,q,kx,qibz_k,irot,qbz_kr,kr,isp,
```

```
00485      &       ngc,ngb,nbmax,ntqxx,nctot,ncc=0)
00486 !TIME1_1088 "get_zmelt"
00487
00488 c$$$!! ccccccccc START: old version, instead of get_zmelt ccccccccccc
00489 c$$$         call  readcphi(q, nlmto,isp, quu, cphikq)
00490 c$$$         if(debug) write(6,*) ' sxcf: 2'
00491 c$$$         do      it = 1,ntq
00492 c$$$            itp        = itq(it)
00493 c$$$            cphiq(1:nlmto,it) = cphikq(1:nlmto,itp)
00494 c$$$            write(*,*)'svvvv ',it, itp, sum(cphiq(:,it))
00495 c$$$         enddo
00496 c$$$              write(*,*)'srrrrr 1c',sum(cphiq(:,1:ntq)),ntq
00497 c$$$
00498 c$$$         call dinv33(qbas,0,qbasinv,det)
00499 c$$$         if(debug) write(6,*) ' sxcf: 1'
00500 c$$$         if(allocated(expikt)) deallocate(expikt)
00501 c$$$         allocate(expikt(natom))
00502 c$$$ccccccccccccccccccccccccccccccc
00503 c$$$!!    rotate atomic positions invrot*R = R' + T
00504 c$$$               invr  = invrot (irot,invg,ngrp)
00505 c$$$               tr    = tiat(:,:,invr)
00506 c$$$               iatomp= miat(:,invr)
00507 c$$$               symope= symgg(:,:,irot)
00508 c$$$               shtv  = matmul(symope,shtvg(:,invr))
00509 c$$$!TIME1 "before ppbafp_v2"
00510 c$$$!TIME0
00511 c$$$
00512 c$$$!! -- ppb= <Phi(SLn,r) Phi(SL'n',r) B(S,i,Rr)>
00513 c$$$c              call ppbafp_v2 (irot,ngrp,isp,nsp,
00514 c$$$c     i             il,in,im,nlnm, !w(i_mnl),
00515 c$$$c     d             nl,nn_,nclass,nlnmx,
00516 c$$$c     i             mdimx,lx,nx_,nxx_, !Bloch wave
00517 c$$$c     i             cgr, nl-1,  !rotated CG
00518 c$$$c     i             ppbrd,      !radial integrals
00519 c$$$c     o             ppb)
00520 c$$$               ppb = ppbir(:,irot,isp)
00521 c$$$!! qk = q-rk. rk is inside 1st BZ, not restricted to the irreducible BZ
00522 c$$$               qk =  q - qbz_kr !qbz(:,kr)
00523 c$$$               call readcphi(qk, nlmto,isp, quu, cphikq)
00524 c$$$!TIME1 "before expikt"
00525 c$$$!TIME0
00526 c$$$
00527 c$$$!! ================================================
00528 c$$$!!     matrix elements <psi(q,t') | psi(q-rk,t) B(rk,R,i)>
00529 c$$$!!     including the phase factor exp(ik.T)
00530 c$$$!!     B(rot*k,r) = B(k,invrot*r)
00531 c$$$!! ================================================
00532 c$$$!! phase factors expikt(ia) is for exp(ik.T(R))
00533 c$$$               do ia = 1,natom
00534 c$$$                  expikt(ia) = exp(img*tpi* sum(qibz_k*tr(:,ia)) )
00535 c$$$               end do
00536 c$$$!! matrix elements
00537 c$$$!!   core
00538 c$$$               nt   = nctot + nbmax ! = nstate for the case of correlation
00539 c$$$               allocate( zzzmel(nbloch,nt,ntqxx))
00540 c$$$               call psicb_v2  (icore,ncore,ntqxx,iclass,
00541 c$$$     i             dreal(expikt(1:natom)),dimag(expikt(1:natom)),
00542 c$$$     i             cphiq,
00543 c$$$     i             ppb,
00544 c$$$     i             nlnmv,nlnmc,mdim,
00545 c$$$     i             imdim,iatomp,
00546 c$$$     d             mdimx,nlmto,nbloch,nlnmx,nt,ntqxx,natom,nclass,
00547 c$$$     d             nl,nnc,
00548 c$$$     o             zzzmel)
00549 c$$$               if(debug)  write(6,*) ' sxcf_fal2sc: goto psi2bc1'
00550 c$$$ccccccccc ccccccccccccccccccccccccccccccccccccccc
00551 c$$$               write(*,*)'srrrrr 1',sum(cphikq(1:nlmto,1:ntq))
00552 c$$$               write(*,*)'srrrrr 1',sum(cphiq(1:nlmto,1:ntq))
00553 c$$$               write(*,*)'srrrrr 1',sum(ppb)
00554 c$$$               write(*,*)'srrrrr 1',sum(expikt)
00555 c$$$               write(*,*)'srrrrr 1',sum(zzzmel)
00556 c$$$
00557 c$$$!!   valence
00558 c$$$               call psi2b_v2  (nbmax, ntqxx,iclass,
00559 c$$$     i             dreal(expikt(1:natom)),dimag(expikt(1:natom)),
00560 c$$$     i             cphikq,     !occ    q-rk nband
00561 c$$$     i             cphiq,      !unocc  q    ntq
00562 c$$$     i             ppb,
00563 c$$$     i             nlnmv,nlnmc,mdim,nctot,
00564 c$$$     i             imdim,iatomp,
00565 c$$$     d             mdimx,nlmto,nbloch,nlnmx, nband, nt,ntqxx,
00566 c$$$     d             natom,nclass,
00567 c$$$     o             zzzmel)
00568 c$$$               if(verbose()>50) call timeshow("4 after psi2bc1")
00569 c$$$c              if(debug2) then
00570 c$$$                 write(6,"('sum of zmel abszmel=',4d23.16)") sum(zzzmel),sum(abs(zzzmel) )
00571 c$$$c              end if
```

```
00572 c$$$!TIME1 "bfore psi2b_v2"
00573 c$$$!TIME0
00574 c$$$!! -- IPW part.
00575 c$$$               if(debug) write(6,*) ' sxcf_fal1: goto drvmelp2 xxx111'
00576 c$$$               allocate(drealzzzmel(nbloch,nt,ntqxx),dimagzzzmel(nbloch,nt,ntqxx))
00577 c$$$               drealzzzmel=dreal(zzzmel)
00578 c$$$               dimagzzzmel=dimag(zzzmel)
00579 c$$$               deallocate(zzzmel)
00580 c$$$               allocate( rmelt(ngb, nctot+nbmax, ntqxx), ! nstate= nctot+nband
00581 c$$$     &              cmelt(ngb, nctot+nbmax, ntqxx))
00582 c$$$               call drvmelp2( q,             ntqxx, ! q in FBZ
00583 c$$$     i              q-qbz_kr,  nbmax, ! q-rk
00584 c$$$     i              qibz_k,  ! k in IBZ for mixed product basis. rk = symope(qibz_k)
00585 c$$$     i              isp,ginv,
00586 c$$$     i              ngc,ngcmx, ngpmx,nband,itq,
00587 c$$$     i              symope, shtv, qbas, qbasinv,qibz,qbz,nqbz,nqibz,
00588 c$$$     i              drealzzzmel, dimagzzzmel, nbloch, nt,nctot,
00589 c$$$     o              rmelt,cmelt)
00590 c$$$               if(debug) write(6,*) ' sxcf_fal1: end of drvmelp2'
00591 c$$$               deallocate(drealzzzmel,dimagzzzmel)
00592 c$$$               if(verbose()>50) call timeshow("5 after drvmelp")
00593 c$$$               if(nbcut/=0.and.(.not.exchange)) then
00594 c$$$                  do it= nctot+1,nctot+min(nbcut,nbmax)
00595 c$$$                     rmelt(:, it,:) =0d0
00596 c$$$                     cmelt(:, it,:) =0d0
00597 c$$$                  enddo
00598 c$$$               endif
00599 c$$$               write(6,"('sum of rmelt cmelt=',4d23.16)")sum(rmelt),sum(cmelt)
00600 c$$$
00601 c$$$!TIME1 "after drvmelp2"
00602 c$$$!! NOTE:=======================================
00603 c$$$!! zmelt = rmelt(igb(qbz_kr), iocc(q), iunocc(q-qbz_kr)) + i* cmelt
00604 c$$$!! iunocc: band index at target  q.
00605 c$$$!! iocc:   band index at intermediate vector qk = q - qbz_kr
00606 c$$$!! igb: index of mixed product basis       at qbz_kr (or written as rk)
00607 c$$$!!   igb=1,ngb
00608 c$$$!!   ngb=nbloch+ngc  ngb: # of mixed product basis
00609 c$$$!!                   nbloch: # of product basis (within MTs)
00610 c$$$!!                   ngc: # of IPW for the Screened Coulomb interaction.
00611 c$$$!!                   igc is for given
00612 c$$$!! See readgeig in drvmelp2.
00613 c$$$!! ================================================
00614 c$$$!! smbasis ---need to fix this
00615 c$$$c$$$               if(smbasis()) then !
00616 c$$$c$$$                  ntp0= ntqxx
00617 c$$$c$$$                  nn= nnr(kx)
00618 c$$$c$$$                  no= nor(kx)
00619 c$$$c$$$                  allocate( pomat(nn,no) )
00620 c$$$c$$$                  pomat= pomatr(1:nn,1:no,kx)
00621 c$$$c$$$                  if( sum(abs(qibz_k-qrr(:,kx)))>1d-10 .and.kx <= nqibz ) then
00622 c$$$c$$$                     call rx( 'qibz/= qrr')
00623 c$$$c$$$                  endif
00624 c$$$c$$$                  if(no /= ngb.and.kx <= nqibz) then
00625 c$$$c$$$!!     A bit sloppy check only for kx<nqibz because qibze is not supplied...
00626 c$$$c$$$                     write(6,"(' q  ngb  ',3d13.5,3i5)")  qibz_k,ngb
00627 c$$$c$$$                     write(6,"(' q_r  nn no',3d13.5,3i5)") q_r,nn,no
00628 c$$$c$$$                     call rx( 'x0kf_v2h: POmat err no/=ngb')
00629 c$$$c$$$                  endif
00630 c$$$c$$$                  if(timemix) call timeshow("xxx2222 k-cycle")
00631 c$$$c$$$                  ngb = nn      ! Renew ngb !!!
00632 c$$$c$$$                  allocate ( zmel  (nn, nctot+nbmax, ntp0) )
00633 c$$$c$$$                  call matm( pomat, dcmplx(rmelt,cmelt), zmel,
00634 c$$$c$$$     &                 nn, no, (nctot+nbmax)*ntp0 )
00635 c$$$c$$$                  deallocate(rmelt, cmelt)
00636 c$$$c$$$                  allocate( rmelt(ngb, nctot+nbmax, ntp0), !ngb is reduced.
00637 c$$$c$$$     &                 cmelt(ngb, nctot+nbmax, ntp0) )
00638 c$$$c$$$                  rmelt = dreal(zmel)
00639 c$$$c$$$                  cmelt = dimag(zmel)
00640 c$$$c$$$                  deallocate(zmel,pomat)
00641 c$$$c$$$               else
00642 c$$$c$$$                  nn=ngb
00643 c$$$c$$$                  no=ngb
00644 c$$$c$$$               endif
00645 c$$$               nn=ngb
00646 c$$$               no=ngb
00647 c$$$               if( oncew() ) then
00648 c$$$                  write(6,"('ngb nn no=',3i6)") ngb,nn,no
00649 c$$$               endif
00650 c$$$               if(timemix) call timeshow("22222 k-cycle")
00651 c$$$!! === End of zmelt ; we now have matrix element zmelt= rmelt + img* cmelt ===
00652 c$$$               if(allocated(zzzmel))deallocate(zzzmel) !rmel,cmel)
00653 c$$$               if(debug) write(6,*) ' sxcf: goto wtt'
00654 c$$$               if(debug) write(6,"('sum of rmelt cmelt=',4d23.16)")sum(rmelt),sum(cmelt)
00655 c$$$
00656 c$$$!! === End of zmelt ; we now have matrix element zmelt= rmelt + img* cmelt ===
00657 c$$$!! ccccccccc  END: old version, instead of get_zmelt ccccccccccc
00658
```

```
00659
00660 !! --- wtt setcion ---
00661 c$$$               if(bzcase()==2)then
00662 c$$$                  if(kx<=nqibz) then
00663 c$$$                     wtt = wk(kr)
00664 c$$$                     if(nstbz(kr)/=0) wtt = wk(kr)*(1d0-wgtq0p()/nstbz(kr))
00665 c$$$                  elseif(kx>nqibz) then !   wtx= wgt0(kx-nqibz,irot)/dble(nqbz)
00666 c$$$                     wtt= wgt0(kx-nqibz,irot)
00667 c$$$                  endif
00668 c$$$               else
00669 c             if(kx<= nqibz) then !  wtx = 1d0
00670               wtt = wk(kr)
00671 c             else                !  wtx = wgt0(kx-nqibz,irot)
00672 c               wtt = wk(1)*wgt0(kx-nqibz,irot)
00673 c               if(abs(wk(1)-1d0/dble(nqbz))>1d-10) call rx( 'sxcf:wk(1) inconsistent')
00674 c             endif
00675 !!
00676             if(eibz4sig()) then
00677               wtt=wtt*nrkip(kx,irot,ip)
00678             endif
00679
00680 !!-----------------------------------------------------------
00681 !! --- exchange section ---
00682 !!-----------------------------------------------------------
00683             if(exchange) then   !At the bottom of this block, cycle do 1000 irot.
00684 !! We use the matrix elements zmeltt. Now given by "call get_zmelt"
00685 !!
00686 c need to check following comments ----
00687 c     S[i,j=1,nbloch] <psi(q,t) |psi(q-rk,n) B(rk,i)>
00688 c     v(k)(i,j) <B(rk,j) psi(q-rk,n) |psi(q,t')>
00689 c
00690 c     > z1p(j,n,t) = S[i=1,nbloch] <psi(q,t) | psi(q-rk,n) B(rk,i)> v(k)(i,j)
00691 c
00692 c     --- screened exchange case
00693 c     if(screen) then
00694 c     allocate( zw (nblochpmx,nblochpmx))
00695 c     ix = 1
00696 c     ! write(*,*)(kx-2)*(nw_w+1)+ix
00697 c     read(ifrcw,rec=((kx-2)*nw+ix)) zw  ! Readin W(0) - v         !sf 22May02
00698 c     !nw is number of frequency points in general mesh: freq_r(nw), freq_r(1)=0
00699 c     vcoul = vcoul + zw(1:ngb,1:ngb) !c  screen test
00700 c     deallocate(zw)
00701 c     endif
00702 !TIME0_0130
00703               vc = vcoud(1)     ! save vcoud(1)
00704               if (kx == iqini) vcoud(1) = wklm(1)* fpi*sqrt(fpi) /wk(kx)
00705               allocate(z1r(ntqxx,ngb),z2r(ntqxx,ngb),w3pi(ntqxx,ntqxx))
00706               allocate(w3p(nctot+nbmax,ntqxx,ntqxx))
00707               do  it = 1, nctot+nbmax
00708                 do  ivc = 1, ngb
00709                   do  itp = 1, ntqxx
00710                     z1r(itp,ivc) = zmeltt(it,itp,ivc) * vcoud(ivc)
00711                     z2r(itp,ivc) = zmeltt(it,itp,ivc)
00712                   enddo         ! ivc
00713                 enddo           ! it
00714                 call zgemm('N','C',ntqxx,ntqxx,ngb,(1d0,0d0),z1r,ntqxx,
00715      .           z2r,ntqxx,(0d0,0d0),w3pi,ntqxx)
00716 C              call zprm('w3pi',w3p,ntqxx,ntqxx,ntqxx)
00717 C             Faster, but harder to parallelize
00718 !                   call zqsmpy(11,'N','C',ntqxx,ngb,z1r,ntqxx,z2r,ntqxx,
00719 !     .                        (0d0,0d0),w3pi,ntqxx)
00720 C             call zprm('w3pi',w3p,ntqxx,ntqxx,ntqxx)
00721               do  itp = 1, ntqxx
00722                 do itpp = 1, ntqxx
00723                   w3p(it,itp,itpp) = w3pi(itp,itpp)
00724                 enddo
00725               enddo
00726             enddo
00727             vcoud(1) = vc     !restore vcoud(1)
00728             deallocate(z1r,z2r,w3pi)
00729             if(verbose()>=30) call cputid2(' complete w3p',0)
00730             deallocate(zmeltt)
00731             if(debug) then
00732               do  it  = 1,nctot+nbmax; do  itp = 1,ntqxx
00733                 write(6,"(' w3p =',2i4,2d14.6)") it,itp,w3p(it,itp,itp)
00734               enddo;    enddo
00735             endif
00736 !TIME1_0130 "end_of_w3p"
00737
00738 c$$$#else
00739 c$$$!kino 2014-08-13  !$OMP parallel  private(vc)
00740 c$$$!kino 2014-08-13  !$OMP do
00741 c$$$                  do itp= 1,ntqxx
00742 c$$$                     do it = 1,nctot+nbmax
00743 c$$$                        do ivc=1,ngb
00744 c$$$                           zmeltt(it,itp,ivc) =  sum( zmel(:,it,itp)* ppovlz(:,ivc) )
00745 c$$$                        enddo
```

```
00746 c$$$                        enddo
00747 c$$$                     enddo
00748 c$$$!kino 2014-08-13  !$OMP end do
00749 c$$$!kino 2014-08-13  !$OMP do
00750 c$$$                     do 992 itpp= 1,ntqxx
00751 c$$$                        do 993 itp = 1,ntqxx
00752 c$$$                           if(diagonly.and.(itpp/=itp)) cycle
00753 c$$$!! sep2013t a test:c           if(itpp>ntqxxd .and.itp/=itpp) cycle
00754 c$$$                           do 994 it  = 1,nctot+nbmax
00755 c$$$                              w3p(it,itp,itpp) = 0d0
00756 c$$$                              do ivc=1,ngb
00757 c$$$                                 if(ivc==1.and.kx==iqini) then
00758 c$$$                                    vc= wklm(1)* fpi*sqrt(fpi) /wk(kx)
00759 c$$$c     write(6,*)'wklm(1) vc=',wklm(1),vc
00760 c$$$                                 else
00761 c$$$                                    vc= vcoud(ivc)
00762 c$$$                                 endif
00763 c$$$c     zmelt1 =  sum( zmel(:,it,itp)  *ppovlz(:,ivc) )
00764 c$$$c     zmelt2 =  sum( zmel(:,it,itpp) *ppovlz(:,ivc) )
00765 c$$$                                 w3p(it,itp,itpp) = w3p(it,itp,itpp)
00766 c$$$     &                                + vc * zmeltt(it,itp,ivc)*dconjg(zmeltt(it,itpp,ivc))
00767 c$$$                              enddo
00768 c$$$ 994                        continue
00769 c$$$ 993                     continue
00770 c$$$ 992                  continue
00771 c$$$!kino 2014-08-13  !$OMP end do
00772 c$$$!kino 2014-08-13  !$OMP end parallel
00773 c$$$#endif
00774 !KINO                     write(*,*)'kino: w3p checksum=',sum(w3p)
00775 c                    deallocate(zmeltt)
00776 c$$$                  else
00777 c$$$!kino 2014-08-13  !$OMP parallel do
00778 c$$$                     do itpp= 1,ntqxx
00779 c$$$                        do itp = 1,ntqxx
00780 c$$$                           if(diagonly.and.(itpp/=itp)) cycle
00781 c$$$c sep2013t a test:c  if(itpp>ntqxxd .and.itp/=itpp) cycle
00782 c$$$                           do it  = 1,nctot+nbmax
00783 c$$$                              w3p(it,itp,itpp) =dcmplx(
00784 c$$$     &                          sum ( dreal(z1p(:,it,itp))*rmelt(:,it,itpp)
00785 c$$$     &                          +   dimag(z1p(:,it,itp))*cmelt(:,it,itpp) ) ,
00786 c$$$     &                          sum ( dimag(z1p(:,it,itp))*rmelt(:,it,itpp)
00787 c$$$     &                          -   dreal(z1p(:,it,itp))*cmelt(:,it,itpp) ) )
00788 c$$$                           enddo
00789 c$$$                        enddo
00790 c$$$                     enddo
00791 c$$$!kino 2014-08-13  !$OMP end parallel do
00792 c$$$                     deallocate(z1p)
00793 c$$$                  endif
00794 c                 deallocate(zmel)
00795 c$$$!!-- Write the Spectrum function for exchange May. 2001
00796 c$$$                  if(ifexsp/=0) then
00797 c$$$                     do it  = 1, nctot+nbmax
00798 c$$$                        do itp = 1,ntqxx
00799 c$$$                           write(ifexsp,"(3i4, 3f12.4, ' ',d23.15,'  ',d23.15)")
00800 c$$$     &                        ip,itp,it, qbz_kr, ekc(it), -wtt*dreal(w3p(it,itp,itp))
00801 c$$$                        enddo
00802 c$$$                     enddo
00803 c$$$                  endif
00804 c$$$!TIME1 "end of write ifsexsp"
00805
00806 !TIME0_0180
00807 !! --- Correct weigts wfac for valence by esmr
00808               do it = nctot+1, nctot+nbmax
00809                 wfac = wfacx(-1d99, ef, ekc(it), esmr) !gaussian
00810                 w3p(it,1:ntqxx,1:ntqxx) = wfac * w3p(it,1:ntqxx,1:ntqxx)
00811               enddo
00812
00813 !! apr2015 correct weights for core-hole case
00814               if(corehole) then
00815               do it = 1, nctot
00816                 w3p(it,1:ntqxx,1:ntqxx) = wcorehole(it,isp) * w3p(it,1:ntqxx,1:ntqxx)
00817               enddo
00818               endif
00819
00820               do itpp=1,ntqxx
00821                 do itp = 1,ntqxx !S[j=1,nbloch]  z1p(j,t,n) <B(rk,j) psi(q-rk,n) |psi(q,t')>
00822                   if(jobsw==5.and.(itpp/=itp)) cycle
00823 c sep2013t a test:c  if(itpp>ntqxxd .and.itp/=itpp) cycle
00824                    zsec(itp,itpp,ip) = zsec(itp,itpp,ip)
00825     &               - wtt * sum( w3p(:,itp,itpp) )
00826                 enddo
00827               enddo
00828               deallocate( w3p)
00829 c$$$                  if(.not.newaniso()) deallocate(vcoul)
00830 !TIME1_0180 "enddo_zsec_wtt_sum"
00831               cycle             ! next irot do 1000 loop
00832             endif               ! end of if(exchange)
```

```
00833 !! ============== End of exchange section ======================
00834             if(timemix) call timeshow("33333 k-cycle")
00835 cc!TIME1 "end of exchange section"
00836
00837
00838 !!-------------------------------------------------------------
00839 !!---  correlation section -------------------------------
00840 !!-------------------------------------------------------------
00841 !!  We use the matrix elements zmel, which is given by "call get_zmelt"
00842 !!
00843 !!=============================================================
00844 !! need to check the following notes.
00845 !!     The correlated part of the self-energy:
00846 !!     S[n=all] S[i,j=1,nbloch]
00847 !!     <psi(q,t) |psi(q-rk,n) B(rk,i)>
00848 !!     < [w'=0,inf] (1/pi) (w-e)/{(w-e)^2 + w'^2} Wc(k,iw')(i,j) >
00849 !!     <B(rk,j) psi(q-rk,n) |psi(q,t)>
00850 !!     e = e(q-rk,n), w' is real, Wc = W-v
00851 !!=============================================================
00852 !! Get zwz0(omega=0, m, i, j), and zwz(i omega, m, i, j)
00853 !! m intermediate state. zwz= \sum_I,J <i|m I> W_IJ(i omega) <J m|j>
00854 !!
00855 !! sum over both occupied and unoccupied states and multiply by weight
00856 !     new from Jan2006! I think this should be OK.  ----------------------------
00857 !     The output of sxcf_fal2 is  <i|Re[S](e_i)|j> ------------
00858 !     Im-axis integral gives Hermitian part of S.
00859 !     (Be careful as for the difference between
00860 !     <i|Re[S](e_i)|j> and transpose(dconjg(<i|Re[S](e_i)|j>)).
00861 !     ---because e_i is included.
00862 !     The symmetrization (hermitian) procedure is inlucded in hqpe.sc.F
00863 !     old befor Jan2006
00864 !     &        wtt*.5d0*(   sum(zwzi(:,itp,itpp))+ !S_{ij}(e_i)
00865 !     &        dconjg( sum(zwzi(:,itpp,itp)) )   ) !S_{ji}^*(e_j)= S_{ij}(e_j)
00866 !-------------------------------------------------------------------------------
00867 !! omega integlation along im axis.
00868 !! zwzi(istate,itqxx1,itqxx2)=\int_ImAxis d\omega' zwz(omega',istate,itqxx1,itqxx2) 1/(omt-omega')
00869 !! ,where omt=omegat is given in the following 1385-1386 loop.
00870 !!
00871
00872
00873 !! ---------------------------------------------------------------------
00874 !! Contribution to SEc(qt,w) from integration along the imaginary axis
00875 !!     loop over w' = (1-x)/x, frequencies in Wc(k,w')
00876 !!     {x} are gaussian-integration points between (0,1)
00877 !!---------------------------------------------------------------------
00878 !! Readin W(omega=0) and W(i*omega)
00879 !! Then get zwz0 and zwz
00880 !! zwz0 = (zmel*)*(W(*omega=0)   -v)*zmel
00881 !! zwz =  (zmel*)*(W(i*omega(ix))-v)*zmel
00882 !TIME0_0200
00883             allocate( zwz0(        nstate,ntqxx,ntqxx))
00884             allocate( zwz(niw*npm,nstate,ntqxx,ntqxx))
00885             allocate( zw(nblochpmx,nblochpmx))
00886             ix = 1 + (0 - nw_i) !at omega=0 ! nw_i=0 (Time reversal) or nw_i =-nw
00887             read(ifrcw,rec=ix) zw ! direct access read Wc(0) = W(0) - v
00888             call matzwz2(2, zw(1:ngb,1:ngb), zmel, ntqxx, nstate,ngb,
00889      o       zwz0)
00890             do 1380 istate=1,nstate
00891               zwz0(istate,1:ntqxx,1:ntqxx) = ! w(iw) + w(-iw) Hermitian part.
00892      &         (zwz0(istate,1:ntqxx,1:ntqxx)
00893      &         + dconjg(transpose(zwz0(istate,1:ntqxx,1:ntqxx))))/2d0
00894  1380       continue
00895             do 1390 ix=1,niw    !niw is usually ~10 points.
00896               read(ifrcwi,rec=ix) zw ! direct access read Wc(i*omega)=W(i*omega)-v
00897               call matzwz2(2, zw(1:ngb,1:ngb), zmel, ntqxx, nstate,ngb,
00898      o         zwz(ix,1:nstate,1:ntqxx,1:ntqxx)) ! zwz = zmel*(W(0)-v)*zmel
00899               do 1395 istate=1,nstate
00900                 zw(1:ntqxx,1:ntqxx)= zwz(ix,istate,1:ntqxx,1:ntqxx)
00901                 zwz(ix,istate,1:ntqxx,1:ntqxx) = ! w(iw) + w(-iw)  Harmitian part
00902      &           ( zw(1:ntqxx,1:ntqxx)
00903      &           + dconjg(transpose(zw(1:ntqxx,1:ntqxx))) )/2d0
00904                 if(npm==2) then ! w(iw) - w(-iw) Anti Hermitian part
00905                   zwz(ix+niw,istate,1:ntqxx,1:ntqxx) =
00906      &             ( zw(1:ntqxx,1:ntqxx)
00907      &             - dconjg(transpose(zw(1:ntqxx,1:ntqxx))) )/2d0/img
00908                 endif
00909  1395         continue
00910  1390       continue
00911             deallocate(zw)
00912 !TIME1_0200 "endofdo1390"
00913 !! Integration along imag axis for zwz(omega) for given it,itp,itpp
00914 !! itp  : left-hand end of expternal band index.
00915 !! itpp : right-hand end of expternal band index.
00916 !! it   : intermediate state of G.
00917 !TIME0_0210
00918             allocate(zwzi(nstate,ntqxx,ntqxx))
00919             do 1400 itpp= 1,ntqxx
```

```
00920               do 1410 itp = 1,ntqxx
00921                 if((jobsw==5).and.(itpp/=itp)) cycle
00922                 if (jobsw==1.or.jobsw==4) then
00923                   omegat = ef
00924 c               elseif (jobsw==2)               omegat=.5d0*(omega(itp)+omega(itpp))
00925                 else
00926                   omegat = omega(itp)
00927                 endif
00928                 do 1420  it = 1,nstate
00929                   we =.5d0*( omegat -ekc(it))
00930                   if(it <= nctot) then
00931                     esmrx = 0d0
00932                   else
00933                     esmrx = esmr
00934                   endif
00935 !! ua_auto may be recovered in future...
00936 c     if(ua_auto) then
00937 c     ratio = .5d0 *( abs(zwz(niw,it,itp,itp  )/zwz0(it,itp,itp  ))
00938 c     &                     +abs(zwz(niw,it,itpp,itpp)/zwz0(it,itpp,itpp)) )
00939 c     call gen_ua(ratio,niw,freqx,  expa_,ua_)
00940 c     endif
00941 !! Gaussian smearing. Integration along im axis. zwz(1:niw) and zwz0 are used.
00942                   zwzi(it,itp,itpp) =
00943      &             wintzsg_npm(npm, zwz(1,it,itp,itpp), zwz0(it,itp,itpp)
00944      &             ,freqx,wx,ua_,expa_,we,niw,esmrx)
00945 c                    zwzi(it,itp,itpp) =  !rectangular smearing only for npm=1
00946 c     &                       wintzav ( zwz(1,it,itp,itpp),zwz0(it,itp,itpp)
00947 c     &                       ,freqx,wx,ua_,expa_,we,niw, esmrx)
00948  1420           continue
00949  1410         continue
00950  1400       continue
00951             deallocate(zwz0,zwz) !zwzs
00952             if(debug) print *,'zzzzzzzzz sum zwzi ',sum(abs(zwzi(:,:,:)))
00953 !TIME1_0210 "endofdo1400"
00954 !! Contribution to Sigma_{ij}(e_i)
00955             do  1500 itpp= 1,ntqxx
00956               do 1510 itp = 1,ntqxx
00957                 if( jobsw==5.and.(itpp/=itp)) cycle
00958                 zsec(itp,itpp,ip) = zsec(itp,itpp,ip) + wtt*sum(zwzi(:,itp,itpp))
00959  1510         continue
00960  1500       continue
00961             deallocate(zwzi)
00962             if(jobsw==4) goto 2002
00963
00964 !! ---------------------------------------------------------------------
00965 !!  Contribution to SEc(qt,w) from the poles of G (integral along real axis)
00966 !!    Currently, jobsw =1,3,5 are allowed...
00967 !!    The variable we means \omega_epsilon in Eq.(55) in PRB76,165106 (2007)
00968 !! ---------------------------------------------------------------------
00969 !TIME0_0310
00970             if(timemix) call timeshow("goto Sec pole part k-cycle")
00971             if(debug)  write(6,*)'GOTO contribution to SEc(qt,w) from the poles of G'
00972             if (.not.(jobsw == 1 .or. jobsw == 3.or.jobsw==5)) then
00973               call rx( 'sxcf_fal3_scz: jobsw /= 1 3 5')
00974             endif
00975 !! Get index nwxi nwx nt_max. finish quickly. We can simplify this...
00976             call get_nwx(omega,ntq,ntqxx,nt0p,nt0m,nstate,freq_r,
00977      i       nw_i,nw,esmr,ef,ekc,wfaccut,nctot,nband,debug,
00978      o       nwxi,nwx,nt_max)
00979 !! assemble small arrays first.
00980             allocate(we_(nt_max,ntqxx),wfac_(nt_max,ntqxx),ixss(nt_max,ntqxx),ititpskip(nt_max,ntqxx),iirx(
      ntqxx))
00981             call weightset4intreal(nctot,esmr,omega,ekc,freq_r,nw_i,nw,
00982      i       ntqxx,nt0m,nt0p,ef,nwx,nwxi,nt_max,wfaccut,wtt,
00983      o       we_,wfac_,ixss,ititpskip,iirx)
00984
00985 !! We need zw3, the Hermitian part, because we need only hermitean part of Sigma_nn'
00986 !! This can be large array; nwx-nwxi+1 \sim 400 or so...
00987             allocate( zw3(ngb,ngb,nwxi:nwx))
00988             allocate( zw(nblochpmx,nblochpmx))
00989             do ix = nwxi,nwx
00990               nrec = ix-nw_i+1  !freq_r(ix is in nw_i:nx)
00991               read(ifrcw,rec=nrec) zw ! direct access Wc(omega) = W(omega) - v
00992               if(hermitianw) then
00993                 zw3(:,:,ix)=(zw(1:ngb,1:ngb)+transpose(dconjg(zw(1:ngb,1:ngb))))/2d0
00994               else
00995                 zw3(:,:,ix)=zw(1:ngb,1:ngb)
00996               endif
00997             enddo
00998             deallocate(zw)
00999 !! rearrange index of zmel
01000             allocate(zmel1(ngb))
01001             if(jobsw==3) then
01002               allocate(zmel1_(ntqxx,ngb,nstate))
01003               do itpp= 1,ntqxx
01004                 do it  = 1,nstate
01005                   zmel1_(itpp,1:ngb,it) = zmel(1:ngb,it,itpp)
```

```
01006                 enddo
01007               enddo
01008             endif
01009 !! jobsw==3
01010             if( jobsw==3) then
01011               allocate(zwz44(3,ntqxx),zwz4(ntqxx,3))
01012               do itp=1,ntqxx
01013                 do it=1,nt_max
01014                   if(ititpskip(it,itp)) cycle
01015                   we =  we_(it,itp)
01016                   ixs=  ixss(it,itp)
01017                   zmel1(:)=dconjg(zmel(:,it,itp))
01018                   zwz4=0d0
01019                   do ix0=1,3
01020                     ix=ixs+ix0-2
01021                     do igb2=1,ngb
01022 ! !                     **** most time consuming part ******
01023                       zz2=sum(zmel1(1:ngb)*zw3(1:ngb,igb2, iirx(itp)*ix)  )
01024                       call zaxpy(ntqxx,zz2,zmel1_(1,igb2,it),1,zwz4(1,ix0),1)
01025                     enddo
01026                   enddo
01027                   zwz44 = transpose(zwz4)
01028                   do itpp=1,ntqxx
01029                     if(npm==1) then
01030                       zsec(itp,itpp,ip) = zsec(itp,itpp,ip)
01031      .                 + wfac_(it,itp) * alagr3z2(we,freq_r(ixs-1),zwz44(1,itpp),itp==itpp ) !mar015
       ,itp,itpp)
01032                     else
01033                       zsec(itp,itpp,ip) = zsec(itp,itpp,ip)
01034      .                 + wfac_(it,itp) * alagr3z(we,freq_r(ixs-1),zwz44(1,itpp))
01035                     endif
01036                   enddo
01037                 enddo
01038               enddo
01039               deallocate(zwz44,zwz4)
01040             endif
01041
01042 !! jobsw=1,5 Sigma are calculated.
01043             if( jobsw==1.or.jobsw==5) then
01044               do itp=1,ntqxx
01045                 do it=1,nt_max
01046                   if(ititpskip(it,itp)) cycle
01047                   we =  we_(it,itp)
01048                   ixs=  ixss(it,itp)
01049                   zmel1(:)=dconjg(zmel(:,it,itp))
01050                   zwz3=0d0
01051                   do ix0=1,3
01052                     ix=ixs+ix0-2
01053 !!                 **** most time consuming part for jobsw=1 ******
01054 !!                 To reduce computational time, confusing treatment only uses lower half of zw3 (zw3 is
       Hermitan)
01055 !!                 Clean up needed.
01056
01057 !! zwz3 contains <itp| it I> wz3_IJ(we)  <J it| itp>
01058 !!    when zw3 is hermitian.
01059                     if(hermitianw) then
01060                       do igb2=2,ngb
01061                       zz2 = sum(zmel1(1:igb2-1)*zw3(1:igb2-1,igb2,iirx(itp)*ix)  ) +
01062      &                  .5d0* zmel1(igb2)*zw3(igb2,igb2,iirx(itp)*ix)
01063                       zwz3(ix0) = zwz3(ix0)+zz2*zmel(igb2,it,itp)
01064                       enddo       !igb2
01065                       zwz3(ix0) = 2d0*dreal(zwz3(ix0))+ !I think 2d0 is from upper half.
01066      &                zmel1(1)*zw3(1,1, iirx(itp)*ix)*zmel(1,it,itp)
01067 !!    when zw3 is not need to be hermitian case. This gives life time
01068                     else
01069                       zwz3(ix0) = sum( matmul(zmel1(1:ngb), zw3(1:ngb,1:ngb,iirx(itp)*ix))*zmel(1:ngb,it,
       itp) )
01070                     endif
01071                   enddo
01072                   if(npm==1) then
01073                     zsec(itp,itp,ip) = zsec(itp,itp,ip)
01074      .               + wfac_(it,itp)*alagr3z2(we,freq_r(ixs-1),zwz3,.true.)
01075                   else
01076                     zsec(itp,itp,ip) = zsec(itp,itp,ip)
01077      .               + wfac_(it,itp)*alagr3z(we,freq_r(ixs-1),zwz3)
01078                   endif
01079                 enddo
01080               enddo
01081             endif
01082 !TIME1_0310 "EndReCorrelation"
01083 c           goto 2012
01084
01085 cxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
01086 c$$$
01087 c$$$
01088 c$$$
01089 c$$$ccccccccc old code ccccccccccccccccccccccc
```

```
01090 c$$$            if(timemix) call timeshow("55555 k-cycle")
01091 c$$$            if(debug) write(*,'(a,5i6)')'kino: ntqxx,itini,itend,ngb=',ntqxx,itini,itend,ngb
01092 c$$$c$$$               if(test_symmetric_W().and.npm==2) then
01093 c$$$c$$$                  if(onceww(4)) write(6,*)' test_symmetric_W()=',test_symmetric_W(),nwxi,nwx
01094 c$$$c$$$                  allocate(zw3x(ngb,ngb))
01095 c$$$c$$$                  do ix= 1,min(abs(nwxi),nwx)
01096 c$$$c$$$                     zw3x = 0.5d0* (zw3(:,:,ix) + zw3(:,:,-ix))
01097 c$$$c$$$                     zw3(:,:, ix)=zw3x
01098 c$$$c$$$                     zw3(:,:,-ix)=zw3x
01099 c$$$c$$$                  enddo
01100 c$$$c$$$                  deallocate(zw3x)
01101 c$$$c$$$               endif
01102 c$$$!TIME1 "before 2001"
01103 c$$$!TIME0
01104 c$$$            allocate(zwz44(3,ntqxx),zwz4(ntqxx,3))
01105 c$$$            do 2001 itp = 1,ntqxx ! loop over states (q-k,n)
01106 c$$$              omg = omega(itp)
01107 c$$$              if (omg >= ef) then
01108 c$$$                itini= nt0m+1
01109 c$$$                itend= nt_max
01110 c$$$                iii=  1
01111 c$$$              else
01112 c$$$                itini= 1
01113 c$$$                itend= nt0p
01114 c$$$                iii= -1
01115 c$$$              endif
01116 c$$$              do 2011 it = itini,itend ! nt0p corresponds to efp
01117 c$$$                esmrx = esmr
01118 c$$$                if(it<=nctot) esmrx = 0d0
01119 c$$$                wfac = wfacx2(omg,ef, ekc(it),esmrx)
01120 c$$$                if(wfac<wfaccut) cycle ! next it
01121 c$$$                we = .5d0* abs( omg-weavx2(omg,ef, ekc(it),esmr) ) !Gaussian smearing
01122 c$$$                if(it<=nctot .and.wfac>wfaccut) call rx( "sxcf: it<=nctot.and.wfac/=0")
01123 c$$$c$$$                       Rectangular smearing
01124 c$$$c$$$                        if( wfac==0d0) cycle ! next it
01125 c$$$c$$$                        if( omg >= ef) we = 0.5d0* abs( max(omg-ekc(it), 0d0) )
01126 c$$$c$$$                        if( omg <  ef) we = 0.5d0* abs( min(omg-ekc(it), 0d0) )
01127 c$$$c$$$                        if( it<=nctot) then !faleev
01128 c$$$c$$$                           if(wfac/=0) call rx( "sxcf:  it<=nctot.and.wfac/=0")
01129 c$$$c$$$                        endif
01130 c$$$c$$$                      endif
01131 c$$$                if(debug) write(6,"( ' xxx1',10d13.6)") omg,ef, ekc(it),wfac
01132 c$$$                wfac= iii* wfac*wtt
01133 c$$$                do iwp = 1,nw
01134 c$$$                  ixs=iwp
01135 c$$$                  if(freq_r(iwp)>we) exit
01136 c$$$                enddo
01137 c$$$                if(nw_i==0) then
01138 c$$$                  if(ixs+1>nwx) call rx( ' sxcf: ixs+1>nwx xxx2')
01139 c$$$                else
01140 c$$$                  if(omg >=ef .and. ixs+1> nwx ) then
01141 c$$$                    write(6,*)'ixs+1 nwx=',ixs+1,nwx
01142 c$$$                    call rx( ' sxcf: ixs+1>nwx yyy2a')
01143 c$$$                  endif
01144 c$$$                  if(omg < ef .and. abs(ixs+1)> abs(nwxi) ) then
01145 c$$$                    write(6,*)'ixs+1 nwxi=',ixs+1,nwxi
01146 c$$$                    call rx( ' sxcf: ixs-1<nwi yyy2b')
01147 c$$$                  endif
01148 c$$$                endif
01149 c$$$                iir = 1
01150 c$$$                if(omg < ef .and. nw_i/=0) iir = -1
01151 c$$$                zmel1(:)=dconjg(zmel(:,it,itp))
01152 c$$$
01153 c$$$                if (jobsw == 1.or.jobsw==5) then
01154 c$$$                  zwz3=(0d0,0d0)
01155 c$$$!kino 2014-08-13  !$OMP parallel do private(ix,zz2)
01156 c$$$                  do 2014 ix0=1,3
01157 c$$$                    ix=ixs+ix0-2
01158 c$$$                    do igb2=2,ngb
01159 c$$$! !**** most time consuming part for jobsw=1 ******
01160 c$$$                      zz2=sum(zmel1(1:igb2-1)*zw3(1:igb2-1,igb2,iir*ix)  ) +
01161 c$$$     &                 .5d0* zmel1(igb2)*zw3(igb2,igb2,iir*ix)
01162 c$$$                      zwz3(ix0)=zwz3(ix0)+zz2*zmel(igb2,it,itp)
01163 c$$$                    enddo       !igb2
01164 c$$$                    zwz3(ix0)=2d0*dreal(zwz3(ix0))+
01165 c$$$     &               zmel1(1)*zw3(1,1, iir*ix)*zmel(1,it,itp)
01166 c$$$ 2014             continue      !ix
01167 c$$$!kino 2014-08-13  !$OMP end parallel do
01168 c$$$                  if(npm==1) then
01169 c$$$                    zsec(itp,itp,ip) = zsec(itp,itp,ip)
01170 c$$$     .               + wfac*alagr3z2(we,freq_r(ixs-1),zwz3,itp,itp)
01171 c$$$                  else
01172 c$$$                    zsec(itp,itp,ip) = zsec(itp,itp,ip)
01173 c$$$     .               + wfac*alagr3z(we,freq_r(ixs-1),zwz3)
01174 c$$$                  endif
01175 c$$$!!  this contribution to zsec_nn is real (hermitean)
01176 c$$$
```

```
01177 c$$$               elseif(jobsw == 3) then
01178 c$$$                 zwz4=(0d0,0d0)
01179 c$$$!$OMP parallel private(ix,zz2)
01180 c$$$                 do 2015 ix0=1,3
01181 c$$$                   ix=ixs+ix0-2
01182 c$$$!$OMP do reduction(+:zwz4)
01183 c$$$!! Next zaxpy is most time consuming part for jobsw=3.****
01184 c$$$!! I think we can speed up this section...
01185 c$$$                   do igb2=1,ngb
01186 c$$$                     zz2=sum(zmel1(1:ngb)*zw3(1:ngb,igb2, iir*ix)  )
01187 c$$$                     call zaxpy(ntqxx,zz2,zmel1_(1,igb2,it),1,zwz4(1,ix0),1)
01188 c$$$                   enddo
01189 c$$$ 2015            continue      !ix0
01190 c$$$!$OMP end parallel
01191 c$$$                 zwz44 = transpose(zwz4)
01192 c$$$                 do itpp=1,ntqxx
01193 c$$$                   if(jobsw==5.and.(itpp/=itp)) cycle
01194 c$$$                   if(npm==1) then
01195 c$$$                     zsec(itp,itpp,ip) = zsec(itp,itpp,ip)
01196 c$$$     .                + wfac*alagr3z2(we,freq_r(ixs-1),zwz44(1,itpp),itp,itpp)
01197 c$$$                   else
01198 c$$$                     zsec(itp,itpp,ip) = zsec(itp,itpp,ip)
01199 c$$$     .                + wfac*alagr3z(we,freq_r(ixs-1),zwz44(1,itpp))
01200 c$$$                   endif
01201 c$$$                 enddo         !itpp
01202 c$$$               endif           ! inner jobsw=1 or 3
01203 c$$$!!     this contribution to zsec_nn' is not hermitean because W(e_n)
01204 c$$$!!     and must be made hermitean when zsec will be written on disc
01205 c$$$ 2011         continue
01206 c$$$ 2001       continue            !itp
01207 c$$$ccccccccccccccccccccccccccccccccccccccccccccccccc
01208
01209  2012       continue
01210             deallocate(we_,wfac_,ixss,ititpskip,iirx)
01211  2002       continue
01212             deallocate(zw3, zmel, zmel1)
01213             if(allocated(zmel1_)) deallocate(zmel1_)
01214             if(verbose()>50) call timeshow("11after alagr3z iw,itp,it cycles")
01215             if(debug) then
01216               write(6,*)' end of do 2001 '
01217               do itp = 1,ntq
01218                 write(6,'(" zsec=",i3,2d15.7)') itp,zsec(itp,itp,ip)
01219               enddo
01220             endif
01221  1000     continue              ! end do irot
01222           ifvcoud =iclose('Vcoud.'//charnum5(kx))
01223           if(.not.exchange) then
01224             ifrcw  = iclose('WVR.'//charnum5(kx))
01225             ifrcwi = iclose('WVI.'//charnum5(kx))
01226           endif
01227           deallocate(ppovlz)
01228  1100   continue                ! end of kx-loop
01229         ifvcoud =iclose('Vcoud')
01230         if(irot==1) write(6,"('  sum(abs(zsec))=',d23.15)") sum(abs(zsec))
01231         if (allocated(vcoul))deallocate(vcoul)
01232  1001 continue                  ! end do ip
01233 c      if(allocated(freq_r))deallocate(freq_r)
01234 c      if (allocated(expikt))deallocate(expikt)
01235 c!TIME1_0000 "end of sxcf_fal3_scz"
01236       end subroutine sxcf_fal3_scz
01237
01238
01239       subroutine weightset4intreal(nctot,esmr,omega,ekc,freq_r,nw_i,nw,
01240      i ntqxx,nt0m,nt0p,ef,nwx,nwxi,nt_max,wfaccut,wtt,
01241      o we_,wfac_,ixss,ititpskip,iirx)
01242 !! generate required data set for main part of real part integration.
01243       implicit none
01244       integer,intent(in):: ntqxx,nctot,nw_i,nw,nt0m,nwx,nwxi,nt_max
01245       real(8),intent(in)::ef,omega(ntqxx),ekc(ntqxx),freq_r(nw_i:nw),esmr,wfaccut,wtt
01246       real(8),intent(out):: we_(nt_max,ntqxx),wfac_(nt_max,ntqxx)
01247       integer,intent(out) :: ixss(nt_max,ntqxx),iirx(ntqxx)
01248       logical,intent(out) :: ititpskip(nt_max,ntqxx)
01249       integer:: itini,iii,it,itend,wp,ixs,itp,iwp,nt0p
01250       real(8):: omg,esmrx,wfacx2,we,wfac,weavx2
01251       ititpskip=.false.
01252       do itp = 1,ntqxx          !this loop should finish in a second
01253         omg = omega(itp)
01254 !  jobsw==2
01255 !             if (jobsw==2)  omg=.5d0*(omega(itp)+omega(itpp))
01256         iirx(itp) = 1
01257         if( omg < ef .and. nw_i/=0) iirx(itp) = -1
01258         if (omg >= ef) then
01259           itini= nt0m+1
01260           itend= nt_max
01261           iii=  1
01262         else
01263           itini= 1
```

```
01264           itend= nt0p
01265           iii= -1
01266         endif
01267         ititpskip(:itini-1,itp)=.true.
01268         ititpskip(itend+1:,itp)=.true.
01269         do it = itini,itend     ! nt0p corresponds to efp
01270           esmrx = esmr
01271           if(it<=nctot) esmrx = 0d0
01272           wfac_(it,itp) = wfacx2(omg,ef, ekc(it),esmrx)
01273           wfac = wfac_(it,itp)
01274           if(wfac<wfaccut) then
01275             ititpskip(it,itp)=.true.
01276             cycle
01277           endif
01278           wfac_(it,itp)=  wfac_(it,itp)*wtt*iii
01279 !   Gaussian smearing we_= \bar{\omega_\epsilon} in sentences next to Eq.58 in PRB76,165106 (2007)
01280 !   wfac_ = $w$ weight (smeared thus truncated by ef). See the sentences.
01281           we_(it,itp) = .5d0* abs( omg-weavx2(omg,ef, ekc(it),esmr) )
01282           we= we_(it,itp)
01283           if(it<=nctot .and.wfac>wfaccut) call rx( .and."sxcf: it<=nctotwfac/=0")
01284           do iwp = 1,nw
01285             ixs = iwp
01286             if(freq_r(iwp)>we) exit
01287           enddo
01288           ixss(it,itp) = ixs
01289           if(nw_i==0) then
01290             if(ixs+1>nwx) call rx( ' sxcf: ixs+1>nwx xxx2')
01291           else
01292             if(omg >=ef .and. ixs+1> nwx ) then
01293               write(6,*)'ixs+1 nwx=',ixs+1,nwx
01294               call rx( ' sxcf: ixs+1>nwx yyy2a')
01295             endif
01296             if(omg < ef .and. abs(ixs+1)> abs(nwxi) ) then
01297               write(6,*)'ixs+1 nwxi=',ixs+1,nwxi
01298               call rx( ' sxcf: ixs-1<nwi yyy2b')
01299             endif
01300           endif
01301         enddo
01302       enddo
01303       end subroutine weightset4intreal
01304       end module m_sxcfsc
01305 !! ------------------------------------------------------------
01306       subroutine get_nwx(omega,ntq,ntqxx,nt0p,nt0m,nstate,freq_r,
01307      i nw_i,nw,esmr,ef,ekc,wfaccut,nctot,nband,debug,
01308      o nwxi,nwx,nt_max)
01309 !> Determine indexes of a range for calculation.
01310 !! It is better to clean this up...
01311       implicit none
01312       integer,intent(in) :: nctot,nw_i,nw,nstate,nt0p,nt0m,ntq,
01313      & nband,ntqxx
01314       real(8),intent(in):: omega(ntq),esmr,ef,ekc(nctot+nband),wfaccut,
01315      & freq_r(nw_i:nw)
01316       integer,intent(out) :: nt_max,nwxi,nwx
01317
01318       integer:: itp,it,itini,itend,iwp,ixs,ixsmin,ixsmx,verbose
01319       real(8):: omg,wfac,wfacx2,we,weavx2,esmrx,wexx
01320       logical::debug
01321 !!     maximum ixs reqired.
01322       ixsmx =0
01323       ixsmin=0
01324       do 301 itp = 1,ntqxx
01325         omg  = omega(itp)
01326         if (omg < ef) then
01327           itini= 1
01328           itend= nt0p
01329         else
01330           itini= nt0m+1
01331           itend= nstate
01332         endif
01333         do 311 it=itini,itend
01334           esmrx = esmr
01335           if(it<=nctot) esmrx = 0d0
01336           wfac = wfacx2(omg,ef, ekc(it),esmrx)
01337           if(wfac<wfaccut) cycle !Gaussian case
01338           we = .5d0*(weavx2(omg,ef,ekc(it),esmr)-omg)
01339 cc Gaussian=F case  keep here just as a memo
01340 c           if(wfac==0d0) cycle ! next it
01341 c           if(omg>=ef) we = max( .5d0*(omg-ekc(it)), 0d0) ! positive
01342 c           if(omg< ef) we = min( .5d0*(omg-ekc(it)), 0d0) ! negative
01343           if(it<=nctot) then
01344             if(wfac>wfaccut) call rx( .and."sxcf: it<=nctotwfac/=0")
01345           endif
01346           do iwp = 1,nw
01347             ixs=iwp
01348             if(freq_r(iwp)>abs(we)) exit
01349           enddo
01350 c     This change is because G(omega-omg') W(omg') !may2006
```

```
01351 c     if(ixs>ixsmx  .and. omg<=ef ) ixsmx  = ixs
01352 c     if(ixs>ixsmin .and. omg> ef ) ixsmin = ixs
01353           if(ixs>ixsmx  .and. omg>=ef ) ixsmx  = ixs
01354           if(ixs>ixsmin .and. omg< ef ) ixsmin = ixs
01355           wexx  = we
01356           if(ixs+1 > nw) then
01357             write (*,*) ' nw_i ixsmin',nw_i, ixsmin
01358             write (*,*) ' wexx ',wexx
01359             write (*,*) ' omg ekc(it) ef ', omg,ekc(it),ef
01360             call rx( ' sxcf 222: |w-e| out of range')
01361           endif
01362  311    continue
01363  301   continue                  !end of SEc w and qt -loop
01364 !!
01365       if(nw_i==0) then           !time reversal
01366         nwxi = 0
01367         nwx  = max(ixsmx+1,ixsmin+1)
01368       else                       !no time revarsal
01369         nwxi = -ixsmin-1
01370         nwx  =  ixsmx+1
01371       endif
01372       if (nwx > nw   ) then
01373         call rx( ' sxcf_fal3_sc nwx check : |w-e| > max(w)')
01374       endif
01375       if (nwxi < nw_i) then
01376         call rx( ' sxcf_fal3_sc nwxi check: |w-e| > max(w)')
01377       endif
01378       if(debug) write(6,*)'nw, nwx=',nw,nwx
01379       if(verbose()>50)call timeshow("10before alagr3z iw,itp,it ")
01380 !!  Find nt_max
01381       nt_max=nt0p                !initial nt_max
01382       do 401 itp = 1,ntqxx
01383         omg     = omega(itp)
01384         if (omg > ef) then
01385           do  it = nt0m+1,nstate ! nt0m corresponds to efm
01386             wfac = wfacx2(ef,omg, ekc(it),esmr)
01387 c                        if( (GaussSmear().and.wfac>wfaccut)
01388 c     &                       .or.(.not.GaussSmear().and.wfac/=0d0)) then
01389             if(wfac>wfaccut) then
01390               if (it > nt_max) nt_max=it ! nt_max is  unocc. state
01391             endif                ! that ekc(it>nt_max)-omega > 0
01392           enddo                  ! so it > nt_max does not contribute to omega pole integral
01393         endif
01394  401   continue                  !end of  w and qt -loop
01395       end subroutine get_nwx
```

4.27 gwsrc/x0kf_v4h.F File Reference

Functions/Subroutines

- subroutine x0kf_v4hz (npm, ncc,ihw, nhw, jhw, whw, nhwtot,n1b, n2b, nbnbx, nbnb,q,nsp, isp_k, isp_kq, symmetrize,qbas, ginv, rk, wk,
- subroutine dpsion5 (frhis, nwhis, freqr, nw_w, freqi, niwt,realomega,imagomega,rcxq, npm, nw_i, nmbas1, nmbas2,zxq, zxqi,
- logical function checkbelong (qin, qall, nq, ieibz)
- subroutine hilbertmat (zz, nwhis, his_L, his_C, his_R, rmat)
- real(8) function wcutef (e, ecut, ecuts)

4.27.1 Function/Subroutine Documentation

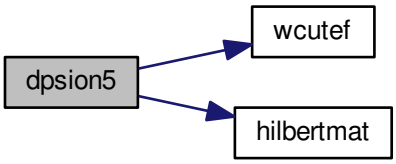
4.27.1.1 logical function checkbelong (real(8), dimension(3) *qin,* real(8), dimension(3,nq) *qall,* integer *nq,* integer *ieibz*)

Definition at line 1459 of file x0kf_v4h.F.

4.27.1.2 subroutine dpsion5 (real(8), dimension(nwhis+1) *frhis,* integer(4) *nwhis,* real(8), dimension(0:nw_w) *freqr,* integer(4) *nw_w,* real(8), dimension(niwt) *freqi,* integer(4) *niwt,* logical *realomega,* logical *imagomega,* complex(8), dimension(nmbas1,nmbas2, nwhis,npm) *rcxq,* integer(4) *npm,* integer(4) *nw_i,* integer(4) *nmbas1,* integer(4) *nmbas2,* complex(8), dimension (nmbas1,nmbas2, nw_i: nw_w) *zxq, zxqi*)

Definition at line 1153 of file x0kf_v4h.F.

Here is the call graph for this function:



Here is the caller graph for this function:

4.27.1.3 subroutine hilbertmat (complex(8) *zz,* integer(4) *nwhis,* real(8), dimension(-nwhis:nwhis) *his_L,* real(8), dimension(-nwhis:nwhis) *his_C,* real(8), dimension(-nwhis:nwhis) *his_R,* complex(8), dimension(-nwhis:nwhis) *rmat*)

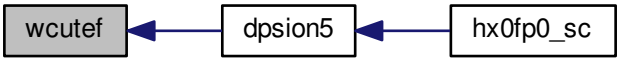
Definition at line 1474 of file x0kf_v4h.F.

Here is the caller graph for this function:

4.27.1.4 real(8) function wcutef (real(8) *e,* real(8) *ecut,* real(8) *ecuts*)

Definition at line 1575 of file x0kf_v4h.F.

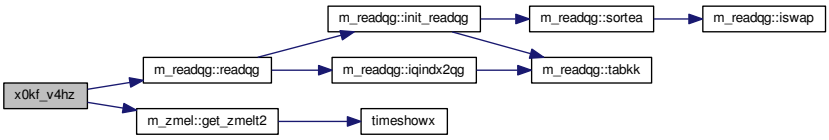
Here is the caller graph for this function:



4.27.1.5 subroutine x0kf_v4hz (integer(4) *npm,* integer(4) *ncc,* integer(4), dimension(nbnbx,nqbz,npm) *ihw,* integer(4), dimension(nbnbx,nqbz,npm) *nhw,* integer(4), dimension(nbnbx,nqbz,npm) *jhw,* real(8), dimension(nhwtot) *whw,* integer(4) *nhwtot,* integer(4), dimension(nbnbx,nqbz,npm) *n1b,* integer(4), dimension(nbnbx,nqbz,npm) *n2b,* integer(4) *nbnbx,* integer(4), dimension(nqbz,npm) *nbnb,* real(8), dimension(3) *q,* integer(4) *nsp,* integer(4) *isp_k,* integer(4) *isp_kq,* logical *symmetrize,* real(8), dimension(3,3) *qbas,* real(8), dimension(3,3) *ginv,* real(8), dimension(3,nqbz) *rk,* real(8), dimension(nqbz) *wk*)

Definition at line 1 of file x0kf_v4h.F.

Here is the call graph for this function:



Here is the caller graph for this function:

4.28 x0kf_v4h.F

```
00001       subroutine x0kf_v4hz (npm,ncc,
00002      i             ihw,nhw,jhw,whw,nhwtot, ! tetwt5
00003      i             n1b,n2b,nbnbx,nbnb,     !  use whw by tetwt5 ,
00004      i                   q,
00005      i                   nsp,isp_k,isp_kq,symmetrize,
00006      i                   qbas,ginv,rk,wk,
00007 c     i                   mdim,
00008      d                   nlmto,nqbz,nctot,
00009 c     d                   natom,
00010      d                   nbloch,nwt,
00011      i      iq, ngbb, ngc, ngpmx,ngcmx,
00012      i      nqbze, nband,nqibz,
```

```
00013       o      rcxq,
00014       i     nolfco,zzr,nmbas, zcousq,
00015       i     chipmzzr,eibzmode,
00016       i     nwgt,igx,igxt,ngrp,eibzsym, crpa)
00017        use m_readqg,only   :  readqg
00018        use m_readeigen,only:  readeval
00019        use keyvalue,only   :  getkeyvalue
00020        use m_rotmpb,only   :  rotmpb2
00021        use m_readqgcou,only:
00022       o qtt_, nqnum
00023        use m_pkm4crpa,only : readpkm4crpa
00024        use m_zmel,only     : get_zmelt2,
00025       o zmel !,ppbir ,ppovlz
00026
00027 !! === calculate chi0, or chi0_pm ===
00028 !! We calculate imaginary part of chi0 along real axis.
00029 !!
00030 !! NOTE: rcxq is i/o variable for accumulation. We use E_mu basis when chipm=F.
00031 !!
00032 !!
00033 !! ppovl= <I|J> = O , V_IJ=<I|v|J>
00034 !! (V_IJ - vcoud_mu O_IJ) Zcousq(J, mu)=0, where Z is normalized with O_IJ.
00035 !! <I|v|J>= \sum_mu ppovl*zcousq(:,mu) v^mu (Zcousq^*(:,mu) ppovl)
00036 !!
00037 !! zmelt contains O^-1=<I|J>^-1 factor. Thus zmelt(phi phi J)= <phi |phi I> O^-1_IJ
00038 !! ppovlz(I, mu) = \sum_J O_IJ Zcousq(J, mu)
00039 !!
00040 !! when nmbas1=2, this works in a special manner for nolfco=T chipm=F. mar2012takao
00041 !!
00042 !!
00043 !! OUTPUT:
00044 !!  rcxq (nmbas,nmbas,nwt,npm): for given q,
00045 !!       rcxq(I,J,iw,ipm) =
00046 !!       Im (chi0(omega))= \sum_k <I_q psi_k|psi_(q+k)> <psi_(q+k)|psi_k> \delta(\omega- (e_i-ej))
00047 !!      When npm=2 we calculate negative energy part. (time-reversal asymmetry)
00048 !!
00049 c              See also tetwt5. and check weight mode=4 of hx0fp0 and (mode=5,6).
00050 c
00051 c- takao kotani Apr 2002   This originated from Ferdi's x0k.
00052 cr daxpy dominates the cpu time
00053 c
00054 c
00055 c x0(i,j)(q,w) = S[k=FBZ] S[t=occ] S[t'=unocc]
00056 c  <M(q,i) psi(k,t) |psi(k+q,t')> <psi(k+q,t')| psi(k,t) M(q,j)>
00057 c  { 1/[w-e(k+q,t')+e(k,t)+i*delta] - 1/[w+e(k+q,t')-e(k,t)-i*delta] }
00058 c  ; w is real. x0 is stored into rcxq.
00059 c
00060 c zzmel =  <psi(k+q,t') | psi(k,t) B(R,i)>
00061 c zmel  =  <psi(k+q,t') | psi(k,t) M(R,i)>
00062 c rcxq  =  zeroth order response function along the positive real axis.
00063 c          Note this is accmulating variable. Equivalnet with zxq. See rcxq2zxq below.
00064 c
00065 c q       = q-vector in x(q,iw)
00066 c ifchi   = direct access unit file for cphi, the coefficient of eigenfunction for argumentation wave.
00067 c qbas    = base reciprocal lattice vectors
00068 c ginv    = inverse of qbas s. indxrk.f
00069 c
00070 c ppb     = <phi(RLn) phi(RL'n') B(R,i)>
00071 c
00072 c iclass  = given an atom, tells the class
00073 c iindxk  = index for k-points in the FBZ
00074 c rk      = k-points in the 1st BZ
00075 c wk      = weight for each k-point in the 1st BZ
00076 c freq    = frequency points along positive imaginary axis
00077 c
00078 c
00079 c mdim    = dimension of B(R,i) for each atom R
00080 c nlnmx   = maximum number of l,n,m
00081 c nlmto   = total number of LMTO basis functions
00082 c nqbz    = number of k-points in the 1st BZ
00083 c n1,n2,n3= divisions along base reciprocal lattice vectors
00084 c natom   = number of atoms
00085 c noccx   = maximum number of occupied states
00086 c noccxv  = maximum number of occupied valence states
00087 c nbloch  = total number of Bloch basis functions
00088 c
00089 c cphi_k cphi_kq:  b(k) and b(k+q)
00090 c  : coefficients of eigenfunctions for argumentation waves in each MT
00091 c
00092       implicit none
00093       integer(4):: npm,ncc,ngbb,natom,nwt,nsp,isp_k,isp_kq,nlmto !,noccx,noccxv
00094      &  ,nl,nclass,nnc,nlnmx,nbloch,iq,nqibz,iatom,nctot,nbmx,iopen !mdimx,
00095      &  ,jpm,ibib,itps,nt0,ntp0,ngp_kq,ngp_k,it,itp,iw,igb2,igb1,ngb
00096      &  ,nn,no,isx,iclose,k,nbnbx,nqbz
00097       real(8):: q(3),qbas(3,3),ginv(3,3),rk(3,nqbz),wk(nqbz),ebmx
00098 c      complex   (8):: rcxq (ngbb,ngbb, nwt,npm),aaa
00099       complex   (8):: rcxq (nmbas,nmbas,nwt,npm)
```

```
00100       complex(8) :: imag=(0d0,1d0),trc,aaa !phase(natom),
00101       complex(8),allocatable:: cphi_k(:,:),cphi_kq(:,:),geig_kq(:,:),geig_k(:,:)
00102       integer(4):: ngpmx, ngcmx, nqbze, nband,
00103      &              ngc,nadd(3), !ngvecpB(3,ngpmx,nqbze), ngpn(nqbze),
00104      &             igc, !ngveccB(3,ngcmx),
00105      &             ngvecp_kq(3,ngpmx),ngvecp_k(3,ngpmx)
00106       complex(8),allocatable :: zmelt(:,:,:)!,zmelt1(:,:,:)
00107       real(8) :: qbasinv(3,3), det,qdiff(3),add(3),symope(3,3)
00108      &    ,shtv(3)=(/0d0,0d0,0d0/)
00109       data symope /1d0,0d0,0d0, 0d0,1d0,0d0, 0d0,0d0,1d0/
00110 c      real(8) :: ppb_unused(*)
00111
00112 c      integer(4) :: mdim(natom)
00113
00114       complex(8),allocatable :: ttx(:,:)
00115       complex(8),allocatable::  z1p(:,:)
00116       integer(4) ::  nbnb(nqbz,npm),
00117      &  n1b(nbnbx,nqbz,npm), n2b(nbnbx,nqbz,npm)
00118       complex(8),allocatable:: zzmel(:,:,:)
00119 c      integer(4)::imdim(natom),iatomp(natom)
00120       logical,parameter:: debug=.false.
00121 c---tetwt5
00122       logical:: hist      ,ipr
00123       integer(4):: nhwtot,
00124      &  ihw(nbnbx,nqbz,npm),nhw(nbnbx,nqbz,npm),jhw(nbnbx,nqbz,npm)
00125       real(8):: whw(nhwtot)
00126       complex(8) :: zmelt1,zmelt2,zmeltt(ngbb)      !.........................sf 21May02
00127 c         complex(8), allocatable :: zxq_(:,:,:) !...........sf 21May02
00128       real(8) :: imagweight !.........................sf 21May02
00129 c         logical :: takao=.false. !.......................sf 21May02
00130 c         allocate( zxq_( nbloch + ngc,nbloch + ngc,nwt)) !..sf 21May02
00131       integer(4)::nocc
00132       real(8):: eband(nband)
00133 c      integer(4):: n_index_qbz
00134 c      integer(4):: index_qbz(n_index_qbz,n_index_qbz,n_index_qbz)
00135 c-----
00136 c      integer(4):: nlnm(*),nlnmv(*),nlnmc(*)!,iclass(*)!,icore(*),ncore(*)
00137       integer(4):: verbose
00138 c---for iepsmode
00139       logical   :: nolfco !iepsmode
00140       integer(4):: nmbas, imb1,imb2, imb !nmbas1x !nmbas2,nmbas1,
00141       complex(8):: zq01,zq02
00142 c      real(8)   :: zq0zq0
00143       complex(8)   :: zq0zq0 !This is a bug for the case of two atoms per cell!!! oct2006
00144 c      complex(8):: rcxqmean(nwt,npm,nmbas1,nmbas2)
00145
00146       real(8):: vec_kq_g(3),vec_k_g(3),vec_kq(3),vec_k(3),quu(3),tolqu=1d-4,quu1(3),quu2(3)
00147
00148       integer(4):: nbcut,nbcut2
00149       logical :: iww1=.true.,iww2=.true.
00150
00151       logical:: smbasis
00152       integer(4):: ifpomat, nbloch_r, ngbo,iqxdummy
00153 c      integer(4),allocatable:: io(:),in(:),io_q(:),in_q(:)
00154       complex(8),allocatable:: pomat(:,:), zmeltn(:,:,:)
00155       real(8):: q_r(3)
00156       complex(8):: img=(0d0,1d0),zzz(ngbb)
00157
00158       integer(4):: nkmin,  nkmax, nkqmin, nkqmax
00159 c      real(8):: qq(3)
00160       integer(4):: ib1, ib2,      ngcx,ix,iy
00161
00162       integer(4),allocatable:: ngvecc(:,:)
00163       logical :: onceww !testtr,negative_testtr
00164
00165 !! takao apr2012
00166       logical :: z1offd !, z1stcol
00167       complex(8),target :: zzr(ngbb,nmbas) !ppovlz(ngbb,ngbb),
00168       integer:: igb
00169       logical:: symmetrize
00170
00171 !! jun2012takao
00172 c      real(8):: qeibz(3,nqbz),  !    aik(3,3,ngrpt)
00173       integer:: ngrp,nwgt(nqbz) !,ngrpt, aiktimereversal(ngrpt),nwgtieibz,ieibz
00174       integer:: igx(ngrp*2,nqbz),igxt(ngrp*2,nqbz),ieqbz
00175 !! nwgt(neibz
00176       logical:: checkbelong,eibzmode, chipmzzr
00177       complex(8):: zcousq(ngbb,ngbb)  !ppovl(ngc,ngc) ,
00178       complex(8),allocatable:: zcousqr(:,:,:),rcxq0(:,:),rcxq00(:,:),rcxq000(:,:),rcxqwww(:,:)
00179 c      complex(8):: zcousqrsum(ngbb,ngbb,2),    zmeltx(ngbb),zmelty(ngbb),zcousqrx(ngbb,ngbb)
00180       complex(8):: zmeltx(ngbb),zmelty(ngbb),zcousqrx(ngbb,ngbb) ,zcousqc(ngbb,ngbb)
00181      & ,rzc(ngbb,ngbb),zcousqinv(ngbb,ngbb),cmat(ngbb,ngbb)
00182       integer::  eibzsym(ngrp,-1:1),neibz,icc,ig,eibzmoden,ikp,i,j,itimer,icount,iele
00183       integer:: irotm,nrotmx,ixx,iyy,itt,ntimer, nccc, nxx,iagain,irotm1,irotm2
00184       integer,allocatable:: i1(:,:),i2(:,:),nrotm(:)
00185       complex(8),allocatable:: zrotm(:,:),zrr(:,:),zrrc(:,:),zrr_(:,:,:),zrrc_(:,:,:),zmmm(:),zrrx(:,:)
00186       complex(8),pointer:: zmat(:,:)
```

```
00187 c      complex(8),allocatable,target:: ppovl_(:,:)
00188 c#ifdef USE_GEMM_FOR_SUM
00189 c      complex(8),allocatable :: zmelt_tmp(:,:,:)
00190 c#endif
00191       complex(8),allocatable:: rcxq_core(:,:)
00192 !$    integer:: omp_get_num_threads
00193       logical:: eibz4x0
00194       logical :: crpa
00195       real(8):: wpw_k,wpw_kq
00196       real(8):: vec_kcrpa(3),vec_kqcrpa(3)
00197
00198       logical :: exchange=.false.
00199       integer:: irot=1
00200       integer:: ntqxx,nbmax
00201
00202
00203       if (symmetrize) goto 5000
00204
00205 c -----------------------------------------------
00206 !TIME0_1001
00207       write(6,'(" x0kf_v4hz: q=",3f8.4,$)')q
00208       call cputid(0)
00209 c$$$!! check eibzmode assumes nmbas1=nmbs2
00210 c$$$      if(eibzmode) then
00211 c$$$         if(nmbas1/=nmbas2) then
00212 c$$$            write(6,*)'x0kf_v4h: eibzmode=T only allow nmbas1=nmbas2.',nmbas1,nmbas2,nmbas
00213 c$$$            stop 'x0kf_v4h: eibzmode=T only allow nmbas1=nmbas2.'
00214 c$$$         endif
00215 c$$$      endif
00216 !!
00217 c$$$      imdim(1) = 1
00218 c$$$      do iatom = 1,natom
00219 c$$$        iatomp(iatom) = iatom
00220 c$$$        if(iatom<natom) imdim(iatom+1)=imdim(iatom)+mdim(iatom)
00221 c$$$      enddo
00222 c      nctot      = noccx - noccxv
00223       call dinv33(qbas,0,qbasinv,det)
00224 c      phase= (1d0,0d0) !coskt = 1d0; sinkt = 0d0
00225       allocate(cphi_k(nlmto,nband),cphi_kq(nlmto,nband), geig_kq(ngpmx,nband),  geig_k(ngpmx,nband) )
00226       call getkeyvalue("GWinput","nbcutlow",nbcut, default=0 )
00227       call getkeyvalue("GWinput","nbcutlowto",nbcut2, default=0 )
00228       call getnemx(nbmx,ebmx,7,.true.)
00229 c$$$!TIME0
00230 c$$$      if(smbasis()) then !need to check again, when we will make smbasis on.
00231 c$$$ccccccccccccccccccccccccccccccc
00232 c$$$        if(ncc/=0) then
00233 c$$$          write(6,*)"Timereversal=F(ncc/=0) not yet implemented for smbasis."
00234 c$$$          write(6,*)" pomat should be generated correctly                 ."
00235 c$$$Cstop2rx 2013.08.09 kino          stop "Timereversal=F(ncc/=0) not yet implemented for smbasis."
00236 c$$$          call rx( "Timereversal=F(ncc/=0) not yet implemented for smbasis.")
00237 c$$$        endif
00238 c$$$ccccccccccccccccccccccccccccccc
00239 c$$$
00240 c$$$        ifpomat = iopen('POmat',0,-1,0) !oct2005
00241 c$$$C... smoothed mixed basis !oct2005
00242 c$$$C This replace original zmelt with new zmelt based on smoothed mixed basis.
00243 c$$$        do
00244 c$$$          read(ifpomat) q_r,nn,no,iqxdummy !readin reduction matrix pomat
00245 c$$$c         write(6,"('ttt: q  =',3f12.5)") q
00246 c$$$c         write(6,"('ttt: q_r=',3f12.5)") q_r
00247 c$$$          allocate( pomat(nn,no) )
00248 c$$$          read(ifpomat) pomat
00249 c$$$          if( sum(abs(q-q_r))<1d-10) then ! .and.kx <= nqibz ) then
00250 c$$$            write(6,*) 'ok find the section for give qibz_k'
00251 c$$$            exit
00252 c$$$!         elseif (kx >nqibz ) then
00253 c$$$!           exit
00254 c$$$          endif
00255 c$$$          deallocate(pomat)
00256 c$$$        enddo
00257 c$$$        if( sum(abs(q-q_r))>1d-10 ) then
00258 c$$$          write(6,"('q  =',3f12.5)") q
00259 c$$$          write(6,"('q_r=',3f12.5)") q_r
00260 c$$$Cstop2rx 2013.08.09 kino          stop 'POmat reading err q/=q_r'
00261 c$$$          call rx( 'POmat reading err q/=q_r')
00262 c$$$        endif
00263 c$$$        isx = iclose('POmat')
00264 c$$$      endif
00265 c$$$!TIME1 "after if smbasis"
00266
00267 ckino
00268 !KINO      it=0
00269 !KINO      do k=1,nqbz
00270 !KINO        if(eibzmode.and.nwgt(k)==0 ) cycle
00271 !KINO        if(sum(nbnb(k,1:npm))==0) cycle
00272 !KINO        it=it+1
00273 !KINO      enddo
```

```
00274 !KINO      write(6,'(a,i3,1x,a,i4)')'iq=',iq,'active-k-points=',it
00275 ckinoend
00276
00277 !TIME1_1001 "beforedo1000"
00278 !! loop over k-points ---------------------------------------------------------------------
00279 c      qq=0d0
00280       do 1000 k = 1,nqbz
00281         if(eibzmode.and.nwgt(k)==0 ) cycle
00282         if(debug) write(6,'("do 1000  k=",i4)')k
00283         ipr=(k<5.or.k==nqbz.or.debug)
00284         if(sum(nbnb(k,1:npm))==0) cycle
00285 !TIME0_1101
00286         if(k<=5.or. (mod(k,max(10,nqbz/20))==1.or.k>nqbz-10) ) then
00287           write(6,"('  x0kf_v4hz: k rk=',i7,3f10.4,$)")k, rk(:,k)
00288           call cputid(0)
00289         endif
00290
00291 !! --- tetra ------ override nt0 itps ntp0 ---------------
00292         nkmin = 999999
00293         nkmax= -999999
00294         nkqmin= 999999
00295         nkqmax=-999999
00296         do jpm=1,npm !npm
00297           do ibib = 1, nbnb(k,jpm)
00298             nkmin  = min(n1b(ibib,k,jpm),nkmin)
00299             nkqmin = min(n2b(ibib,k,jpm),nkqmin)
00300             if(n1b(ibib,k,jpm)<=nband)   nkmax = max(n1b(ibib,k,jpm),nkmax)
00301             if(n2b(ibib,k,jpm)<=nband)  nkqmax = max(n2b(ibib,k,jpm),nkqmax)
00302           enddo
00303         enddo
00304         call readeval(q+rk(:,k),isp_kq,eband)
00305         nkqmax = nocc(eband,ebmx,.true.,nband)
00306         if(npm==2) then
00307           call readeval(rk(:,k),isp_k,eband)
00308           nkmax = nocc(eband,ebmx,.true.,nband)
00309         endif
00310         itps  = nkqmin          ! nkqmin = the num of min   n2 =unocc for jpm=1
00311         nt0   = nkmax
00312         ntp0  = nkqmax - nkqmin +1
00313         if( npm==2.and. nkqmin/=1) then
00314           write(6,*)' npm==2 nkqmin nkqmax  nkmin nkmax=',nkqmin,nkqmax,nkmin,nkmax
00315           call rx( " When npm==2, nkqmin==1 should be.")
00316         endif
00317         if(nkmin/=1) then
00318           call rx( " nkmin==1 should be.")
00319         endif
00320
00321 !... feb2006
00322 !  zzmel(1:nbloch, ib_k,ib_kq)
00323 !      ib_k =[1:nctot]              core
00324 !      ib_k =[nctot+1:nctot+nkmax]  valence
00325 !      ib_kq =[1:nctot]             core
00326 !      ib_kq =[ncc+1:ncc+nkqmax - nkqmin] valence range [nkqmin,nkqmax]
00327 !   If jpm=1, ncc=0.
00328 !   If jpm=2, ncc=ncore. itps=1 should be.
00329 ! There is a little confusion. n1b index contains cores are after valence.
00330 ! You can see codes to treat the confusion.
00331 ! NOTE:
00332 !  q+rk n2b vec_kq  vec_kq_g geig_kq cphi_kq  ngp_kq ngvecp_kq  isp_kq
00333 !    rk n1b vec_k   vec_k_g  geig_k  cphi_k   ngp_k  ngvecp_k   isp_k
00334
00335 c        if(ipr) then
00336 c          write(6,"(' nkRange  nkqRange=',2i6,2x,2i6)") nkmin,nkmax,nkqmin,nkqmax
00337 c        endif
00338
00339
00340
00341
00342 c$$$ccccccccccccccccccccccccccccccccccccccccccccccccccc
00343 c$$$        goto 8828
00344 c$$$ccccccccccccccccccccccccccccccccccccccccccccccccccc
00345 c$$$
00346 c$$$
00347 c$$$
00348 c$$$
00349 c$$$
00350 c$$$!TIME1 "before readphi"
00351 c$$$!TIME0
00352 c$$$!!--- calculate the matrix elements <psi(k+q,t') | psi(k,t) B(R,i)>
00353 c$$$!! Read cphi part of eigenfunctions for k and k+q
00354 c$$$        call  readcphi(q+rk(:,k)-qq, nlmto,isp_kq, quu2,cphi_kq)
00355 c$$$        call  readcphi(  rk(:,k)-qq, nlmto,isp_k, quu1,cphi_k) !quu is used q for eigenfunctions.
00356 c$$$!! ... core
00357 c$$$        if(debug)  call cputid(0)
00358 c$$$!!        allocate( zzmel(nbloch,noccx, ntp0) )
00359 c$$$!!                          q        k    q+k
00360 c$$$        if(debug)  write(6,*)nbloch,nctot,nt0,ncc,ntp0
```

```
00361 c$$$        allocate( zzmel(nbloch, nctot+nt0, ncc+ntp0) )
00362 c$$$        if(debug)  write(6,"('zzzw nkmin nkqmin=',2i5)") nkmin,nkqmin
00363 c$$$        if(onceww(5) ) write(6,*)' nctot ncc=',nctot,ncc
00364 c$$$c      allocate( ppb(nlnmx*nlnmx*mdimx*nclass))
00365 c$$$c      ppb=ppbir(:,irot,ispq)
00366 c$$$        call psicb_v3  ( nctot,ncc,nt0,ntp0,iclass,phase,
00367 c$$$     i             cphi_k (1,nkmin),
00368 c$$$     i             cphi_kq(1,nkqmin),
00369 c$$$     i             ppbir(:,irot,isp_k),!ppb,
00370 c$$$     i             nlnmv,nlnmc,mdim,
00371 c$$$     i             imdim,iatomp,
00372 c$$$     i             mdimx,nlmto,nbloch,nlnmx,natom,nclass,
00373 c$$$     i             icore,ncore,nl,nnc,
00374 c$$$     o             zzmel)
00375 c$$$C ... valence
00376 c$$$      if(debug) write(6,'("4 zzzqqbbb222 ",3d13.5)') sum(abs(zzmel)),sum(zzmel)
00377 c$$$        !call cputid(0)
00378 c$$$        call psi2b_v3  ( nctot,ncc,nt0,ntp0,iclass,phase,
00379 c$$$     i             cphi_k(1,nkmin),
00380 c$$$     i             cphi_kq(1,nkqmin),
00381 c$$$     i             ppbir(:,irot,isp_k),! ppb,
00382 c$$$     i             nlnmv, nlnmc,mdim,
00383 c$$$     i             imdim,iatomp,
00384 c$$$     d             mdimx,nlmto,nbloch,nlnmx,natom,nclass,
00385 c$$$     o             zzmel)
00386 c$$$        if(debug) call cputid(0)
00387 c$$$        if(debug) write(6,'("4 zzzqqbbb222 ",3d13.5)') sum(abs(zzmel)),sum(zzmel)
00388 c$$$!TIME1 "after psicb_v3"
00389 c$$$!TIME0
00390 c$$$!! --- IPW set
00391 c$$$        call readqg('QGpsi',q+rk(:,k)-qq,ginv, vec_kq, ngp_kq, ngvecp_kq)
00392 c$$$        call readqg('QGpsi',  rk(:,k)-qq,ginv, vec_k,  ngp_k,  ngvecp_k)
00393 c$$$!!      ngp_kq = ngpn(kp)  ! q+k   ntp0 in FBZ
00394 c$$$!!      ngp_k = ngpn(k)   ! k     np0  in FBZ
00395 c$$$!!      ngc              ! q          in IBZ
00396 c$$$        ngb  = nbloch + ngc ! This is not ngbb for smbasis()=T. oct2005
00397 c$$$        if(ngb/=ngbb) then
00398 c$$$           write(6,*)' x0kf_v4h: ngb ngbb=',ngb,ngbb
00399 c$$$           call rx( 'x0kf_v4h: ngb/=ngbb')
00400 c$$$        endif
00401 c$$$!!                         q       k        q+k
00402 c$$$        allocate( zmel(ngb,nctot+nt0,ncc+ntp0) )
00403 c$$$        allocate( z1p(ngb,ngb) )
00404 c$$$!! ... read plane wave part of eigenfunction. (note isp is opposite).
00405 c$$$        call readgeig(q+rk(:,k)-qq, ngpmx,isp_kq, vec_kq_g, geig_kq)
00406 c$$$        call readgeig(  rk(:,k)-qq, ngpmx,isp_k,  vec_k_g,  geig_k)
00407 c$$$        if(sum(abs(vec_kq_g-vec_kq))>tolqu) then
00408 c$$$          write(6,"('vec_kq_g :',3d23.10)") vec_kq_g
00409 c$$$          write(6,"('vec_kq  :',3d23.10)") vec_kq
00410 c$$$          call rx( 'x0kf_v4hz:vec_kq_g/=vec_kq')
00411 c$$$        endif
00412 c$$$        if(sum(abs(vec_k_g-vec_k))>tolqu) then
00413 c$$$          write(6,"('vec_k_g :',3d23.10)") vec_k_g
00414 c$$$          write(6,"('vec_k:  ',3d23.10)") vec_k
00415 c$$$          call rx( 'x0kf_v4hz:vec_k_g/=vec_k')
00416 c$$$        endif
00417 c$$$!!      qdiff = q   - qbkp(:) + rk(:,k)
00418 c$$$        qdiff = q    - vec_kq    + vec_k
00419 c$$$        ! q   - (q+k)   + k  is not zero.
00420 c$$$        ! qc  -  q1     + q2
00421 c$$$        add   = matmul(qbasinv, qdiff)
00422 c$$$        nadd  = idint( add + dsign(.5d0,add))  !  write(6,*)' qdif=',qdiff,qbkp(:),rk(:,k)
00423 c$$$        if(sum(abs(add-nadd))>1d-10) call rx( "sexc: abs(add-nadd))>1d-10")
00424 c$$$        zmel = 0d0
00425 c$$$!TIME1 "before melpln2t"
00426 c$$$!TIME0
00427 c$$$!! <Bq Pq2|Pq1> = < Bq Pqu2|  Pqu1> *exp(i2pi nadd)
00428 c$$$        if(ngc/=0) then !Aug2005
00429 c$$$          call melpln2t(ngp_kq, ngvecp_kq  ! q+k  ; kp ngp_kq 1:ntp0 q-point
00430 c$$$     &             , ngp_k, ngvecp_k       ! k    ; k  ngp_k 1:nt0  occupied
00431 c$$$     &           , ngc,  nadd,
00432 c$$$     &       geig_kq(1:ngp_kq, itps:itps+ntp0-1), ntp0, ! q+k  ; kq
00433 c$$$     &       geig_k(1:ngp_k, 1:nt0     ),  nt0, !    ; k
00434 c$$$     i       shtv,  q, q,  symope,qbas,
00435 c$$$     i       vec_kq, !qt oct2013
00436 c$$$     o       zmel (nbloch+1:nbloch+ngc, nctot+1:nctot+nt0,ncc+1:ncc+ntp0))
00437 c$$$        endif
00438 c$$$!! == zmelt contain O^-1=<I|J>^-1 factor. zzmel(J,it,itp)= \sum_I <phi phi|I> O^-1_IJ ==
00439 c$$$        zmel(1:nbloch, 1:nctot+nt0, 1:ncc+ntp0) =
00440 c$$$     &  zzmel(1:nbloch, 1:nctot+nt0, 1:ncc+ntp0)
00441 c$$$!!                       k           q+k
00442 c$$$        deallocate(zzmel)
00443 c$$$         if(debug) write(6,'("4 zzzppp222bbb ",3d13.5)') sum(abs(zmel)),sum(zmel)
00444 c$$$        if(debug) call cputid(0)
00445 c$$$!TIME1 "after melpln2t"
00446 c$$$!TIME0
00447 c$$$!! === zmelt conversion on different basis.
```

```
00448 c$$$        allocate(zmmm(nmbas))   ! this is also obsolete if USE_GEMM_FOR_SUM
00449 c$$$        if(chipmzzr) then       !spin moment basis.
00450 c$$$          zmat => zzr
00451 c$$$        elseif(nolfco .and. nmbas==1) then !for <e^iqr|x0|e^iqr>
00452 c$$$          zmat => ppovlz
00453 c$$$        else                    !may2013  this removes O^-1 factor from zmelt
00454 c$$$          allocate(ppovl_(ngb,ngb))
00455 c$$$          ppovl_=0d0
00456 c$$$          do i=1,nbloch
00457 c$$$            ppovl_(i,i)=1d0
00458 c$$$          enddo
00459 c$$$          ppovl_(nbloch+1:nbloch+ngc,nbloch+1:nbloch+ngc)=ppovl
00460 c$$$          if(.not.eibz4x0()) then !sep2014 added for eibz4x0=F
00461 c$$$            ppovl_=matmul(ppovl_,zcousq)
00462 c$$$          endif
00463 c$$$          zmat => ppovl_
00464 c$$$        endif
00465 c$$$!! :: zmelt conversion muplitpled by zzr.
00466 c$$$        if(verbose()>39) write(6,*)'info: USE GEMM FOR SUM (zmelt = zmelt*zmat) in x0kf_v4h.F'
00467 c$$$        allocate( zmelt_tmp(ngb,nctot+nt0,ncc+ntp0) )
00468 c$$$        call zcopy(ngb*(nctot+nt0)*(ncc+ntp0),zmel,1,zmelt_tmp,1)
00469 c$$$        call zgemm('T','N',ngb,(nctot+nt0)*(ncc+ntp0),ngb,(1d0,0d0),
00470 c$$$     .   zmat,ngb,zmelt_tmp,ngb,(0d0,0d0),zmel,ngb)
00471 c$$$        deallocate(zmelt_tmp)
00472 c$$$        deallocate(zmmm)
00473 c$$$        if(allocated(ppovl_)) deallocate(ppovl_)
00474 c$$$!TIME1  "after matmul zmel"
00475 c$$$c$$$ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00476 c$$$        print *,'xxxxxxxxxxx 8829 xxxxxxxxxxxxx'
00477 c$$$         goto 8829
00478 c$$$c        if(debug) write(6,'("4 zzzppp 111 ",3d15.6)') sum(abs(zmel)),sum(zmel)
00479 c$$$ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00480 c$$$
00481 c$$$
00482 c$$$
00483 c$$$
00484 c$$$
00485 c$$$c$$$!!march2013--> this if branch of nolfco is now unified to do 25
00486 c$$$c$$$!! No LocalFieldCorrection mode
00487 c$$$c$$$        if(nolfco) then !iepsmode==202) then ! just for <exp(iq r)|x0(q,\omega)|exp(iq r)>
00488 c$$$c$$$          do jpm  = 1,npm !
00489 c$$$c$$$            do ibib = 1, nbnb(k,jpm) !---  ibib loop
00490 c$$$c$$$              if(jpm==1) then
00491 c$$$c$$$                if( n1b(ibib,k,jpm) <= nbcut.and. n2b(ibib,k,jpm)>nbcut2) then !oct2005
00492 c$$$c$$$                  if(iww2) then
00493 c$$$c$$$                    write(6,"(' nband_chi0 nbcut nbcut2 n2b n1b=',4i6)")
       nbcut,n2b(ibib,k,jpm),n1b(ibib,k,jpm)
00494 c$$$c$$$                    iww2=.false.
00495 c$$$c$$$                  endif
00496 c$$$c$$$                  cycle
00497 c$$$c$$$                endif
00498 c$$$c$$$              else
00499 c$$$c$$$                if( n2b(ibib,k,jpm) <= nbcut.and. n1b(ibib,k,jpm)>nbcut2) then !oct2005
00500 c$$$c$$$                  if(iww2) then
00501 c$$$c$$$                    write(6,"(' nband_chi0 nbcut nbcut2 n2b n1b=',4i6)")
       nbcut,n2b(ibib,k,jpm),n1b(ibib,k,jpm)
00502 c$$$c$$$                    iww2=.false.
00503 c$$$c$$$                  endif
00504 c$$$c$$$                  cycle
00505 c$$$c$$$                endif
00506 c$$$c$$$              endif
00507 c$$$c$$$
00508 c$$$c$$$              if( jpm==1.and.n2b(ibib,k,jpm) > nbmx) cycle
00509 c$$$c$$$              if( jpm==2.and.n1b(ibib,k,jpm) > nbmx) cycle
00510 c$$$c$$$
00511 c$$$c$$$              if( n1b(ibib,k,jpm) <= nband) then
00512 c$$$c$$$                it = nctot + n1b(ibib,k,jpm) !valence
00513 c$$$c$$$              else
00514 c$$$c$$$                it = n1b(ibib,k,jpm) - nband !core
00515 c$$$c$$$              endif
00516 c$$$c$$$
00517 c$$$c$$$              if( n2b(ibib,k,jpm) <= nband) then
00518 c$$$c$$$                itp = ncc + n2b(ibib,k,jpm) - itps + 1 !val
00519 c$$$c$$$                if(itp > ncc + nkqmax-itps+1 ) cycle
00520 c$$$c$$$              else
00521 c$$$c$$$                itp =       n2b(ibib,k,jpm) - itps + 1 - nband !core
00522 c$$$c$$$              endif
00523 c$$$c$$$
00524 c$$$c$$$              do imb2=1,nmbas
00525 c$$$c$$$                zq02 = zmelt(imb2,it,itp)
00526 c$$$c$$$                do imb1=1,imb2
00527 c$$$c$$$                  zq01 = zmelt(imb1,it,itp)
00528 c$$$c$$$                  zq0zq0 = dconjg(zq01)*zq02
00529 c$$$c$$$                  do iw = ihw(ibib,k,jpm),ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-1
00530 c$$$c$$$                    if (iw .gt. nwt) stop "x0kf_v4hz: iw > nwt"
00531 c$$$c$$$                    !iiww=iw+ihw(ibib,k)-1
00532 c$$$c$$$                    imagweight   = whw(jhw(ibib,k,jpm)+iw-ihw(ibib,k,jpm))
```

```
00533 c$$$c$$$                    if(eibzmode) imagweight = nwgt(k)*imagweight
00534 c$$$c$$$c                    rcxqmean(iw,jpm,imb1,imb2) =  ! here we  sum over ibib (or n, n') and k.
00535 c$$$c$$$c     &              rcxqmean(iw,jpm,imb1,imb2) + zq0zq0*imagweight
00536 c$$$c$$$                    rcxq(imb1,imb2,iw,jpm) =  ! here we  sum over ibib (or n, n') and k.
00537 c$$$c$$$     &              rcxq(imb1,imb2,iw,jpm) + zq0zq0*imagweight !sum over spin in hx0fp0
00538 c$$$c$$$                  enddo ! iw
00539 c$$$c$$$                enddo ! imb1
00540 c$$$c$$$              enddo ! imb2
00541 c$$$c$$$            enddo ! ----- ibib loop
00542 c$$$c$$$          enddo ! ----- jpm loop
00543 c$$$c$$$          deallocate(zmelt,z1p)
00544 c$$$c$$$          cycle !cycye do 1000 here
00545 c$$$c$$$        endif
00546 c$$$c$$$!TIME1 "before jpm ibib loop"
00547 c$$$c$$$!TIME0
00548 c$$$
00549 c$$$
00550 c$$$
00551 c$$$
00552 c$$$
00553 c$$$ 8828   continue
00554
00555
00556 c$$$ this (ppovlz generation) is moved to hx0fp0 and/or hx0fp0_sc.
00557 c$$$!! === zmelt conversion on different basis.
00558 c$$$        if(chipmzzr) then       !spin moment basis.
00559 c$$$c          if(allocated(ppovlz)) deallocate(ppovlz)
00560 c$$$c          allocate(ppovlz(ngb,nmbas))
00561 c$$$c          ppovlz= zzr
00562 c$$$        elseif(nolfco .and. nmbas==1) then !for <e^iqr|x0|e^iqr>
00563 c$$$          continue
00564 c$$$        else                    !may2013  this removes O^-1 factor from zmelt
00565 c$$$          allocate(ppovl_(ngb,ngb))
00566 c$$$          ppovl_=0d0
00567 c$$$          do i=1,nbloch
00568 c$$$            ppovl_(i,i)=1d0
00569 c$$$          enddo
00570 c$$$          ppovl_(nbloch+1:nbloch+ngc,nbloch+1:nbloch+ngc)=ppovl
00571 c$$$          if(.not.eibz4x0()) then !sep2014 added for eibz4x0=F
00572 c$$$            ppovl_=matmul(ppovl_,zcousq)
00573 c$$$          endif
00574 c$$$          ppovlz = ppovl_
00575 c$$$          deallocate(ppovl_)
00576 c$$$        endif
00577 c$$$        if(allocated(zmel)) deallocate(zmel)
00578 c        nbmax= nctot+nt0
00579 c        ntqxx= ncc+ntp0
00580 cc        call get_zmelt(exchange,q,kx,qibz_k,irot,qbz_kr,kr,isp,
00581 cc     &       ngc,ngb,nbmax,ntqxx,isp_k,isp_kq)
00582 c
00583 !! ------------------------------------------------------------------------------
00584 !!note: for usual correlation mode, I think nctot=0
00585 !!--- For dielectric funciton, we use irot=1 kvec=rkvec=q
00586 !            < MPB      middle    |   end >
00587 !!             q      rkvec     | q + rkvec
00588 !                    nkmin:nt0 | nkqmin:ntp0
00589 !                       occ    | unocc
00590 !                    (nkmin=1)
00591 !                    (cphi_k  | cphi_kq !in x0kf)
00592 !
00593 !!     rkvec= rk(:,k)    ! <phi(q+rk,nqmax)|phi(rk,nctot+nmmax)  MPB(q,ngb )>
00594 !!     qbz_kr= rk(:,k)   !
00595 !!     qibz_k= rk(:,k)   ! k
00596         ngb  = nbloch + ngc
00597 !!Get the matrix element zmel   ZO^-1 <MPB psi|psi> , where ZO is ppovlz
00598 !!  Output is zmel(ngb, nctot+nt0,ncc+ntp0)  nkmin:nt0, nkqmin:ntp0
00599 ! nt0=nkmax-nkmin+1  , ntp0=nkqmax-nkqmin+1
00600         call get_zmelt2(exchange,
00601      &      q,irot,q,ngc,ngb,                       !MPB
00602      &                 nkmin, nkmax, isp_k,nctot, !middle state 1:nt0   --> true index of eigen is
      mkmin:mkmin+nt0-1     + nctot
00603      &      q+rk(:,k),nkqmin,nkqmax,isp_kq,ncc)  !end state    1:ntp0  -->                      is
      nkqmin:nkqmin+ntp0-1
00604         zmel = dconjg(zmel)
00605         allocate( z1p(ngb,ngb) )
00606
00607  8829   continue
00608         if(debug) write(6,'("4 zzzppp 222 ",3d15.6)') sum(abs(zmel)),sum(zmel)
00609
00610 !TIME1_1101 "before_get_zmelt2"
00611 !TIME0_1201
00612 c----------------------------
00613 !!  z1p = <M_ibg1 psi_it | psi_itp> < psi_itp | psi_it M_ibg2 >
00614 !!  zxq(iw,ibg1,igb2) = sum_ibib wwk(iw,ibib)* z1p(ibib, igb1,igb2)
00615 !KINO        write(6,'(a,i4)')'kino: npm=',npm
00616 !kino 2014-08-13  !$OMP parallel private(it,itp,iww1,iww2,zmelt2,imagweight)
00617         do 25 jpm  = 1, npm !
```

```
00618         do 25 ibib = 1, nbnb(k,jpm) !---  ibib loop
00619 !KINO           write(6,'(a,5i8)')'kino: ngb,hw(ibib,k,jpm),ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-1=',
00620 !KINO&      ngb,ihw(ibib,k,jpm),ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-1
00621           if(n1b(ibib,k,jpm) <= nband) then
00622             it = nctot + n1b(ibib,k,jpm) !valence
00623             if(it > nctot + nkmax ) cycle
00624           else
00625             it = n1b(ibib,k,jpm) - nband !core
00626           endif
00627           if( n2b(ibib,k,jpm) <= nband) then
00628             itp = ncc + n2b(ibib,k,jpm) - itps + 1 !val
00629             if(itp > ncc + nkqmax-itps+1 ) cycle
00630           else
00631             itp =  n2b(ibib,k,jpm) - itps + 1 - nband !core
00632           endif
00633
00634           if(jpm==1) then
00635             if(n2b(ibib,k,jpm)>nbmx)  then  !nbmx
00636               if(iww1) then
00637                 write(6,*)' nband_chi0 nbmx=',nbmx
00638                 iww1=.false.
00639               endif
00640               cycle
00641             endif
00642             if( n1b(ibib,k,jpm) <= nbcut .and. nbcut2<n2b(ibib,k,jpm) ) then
00643               if(iww2) then
00644                 write(6,"(' nband_chi0 nbcut nbcut2 n2b n1b=',4i6)") nbcut,n2b(ibib,k,jpm),n1b(ibib,k,jpm)
00645                 iww2=.false.
00646               endif
00647               cycle
00648             endif
00649
00650           else !jpm==2 ------------------------------
00651             if( n1b(ibib,k,jpm) > nbmx) then  !nbmx
00652               if(iww1) then
00653                 write(6,*)' nband_chi0 nbmx=',nbmx
00654                 iww1=.false.
00655               endif
00656               cycle
00657             endif
00658             if( n2b(ibib,k,jpm) <= nbcut .and. nbcut2<n1b(ibib,k,jpm) ) then
00659               if(iww2) then
00660                 write(6,"(' nband_chi0 nbcut nbcut2 n2b n1b=',4i6)") nbcut,n2b(ibib,k,jpm),n1b(ibib,k,jpm)
00661                 iww2=.false.
00662               endif
00663               cycle
00664             endif
00665           endif
00666
00667 ccccccccccccccccccccccccccccccc takao variant begin
00668 cc          if(takao) then
00669 cc
00670 cc          do ic = 1,ngb
00671 cc            z1p(1:ngb,ic) =
00672 cc     &      zmelt(ic,it,itp)*dconjg(zmelt(1:ngb,it,itp))
00673 cc          end do
00674 cc          ihww = ihw(ibib,k)
00675 cc
00676 c1ini------------
00677 cc          do iw = 1, nhw(ibib,k)
00678 cc            rviw = whw(jhw(ibib,k)+iw-1)
00679 cC ... this part dominates the cpu time --------------------!
00680 c!           call zaddr_(zxq(1,1,ihww+iw-1),rviw,z1p,ngb**2)
00681 cc           call daxpy(ngb**2*2,rviw,z1p,1,
00682 cc     &                zxq(1,1,ihww+iw-1),1)
00683 cc          enddo
00684 c1end---------
00685 c2ini --------
00686 cc          call  rcxq_zxq(rc1p,z1p,ngb,-1)
00687 cc          do iw = 1, nhw(ibib,k)
00688 cc            rviw = whw(jhw(ibib,k)+iw-1)
00689 C ... this part dominates the cpu time --------------------!
00690 !           call zaddr_(rcxq(1,1,ihww+iw-1),rviw,z1p,ngb**2)
00691 cc            call daxpy(ngb**2,rviw,rc1p,1,
00692 cc     &                rcxq(1,1,ihww+iw-1),1)
00693 cc          enddo
00694 c2end --------------
00695 cc          else
00696 ccccccccccccccccccccccc takao variant end
00697
00698 c$$$           if(newanisox.and.eibzmoden==1) then ! This is slow.
00699 c$$$             zmeltx =  zmelt(:,it,itp)
00700 c$$$             z1p=0d0
00701 c$$$             do ieqbz =1, nwgt(k) !equivalent points for ieibz
00702 c$$$                                  !igx,igxt specifies space-group operation (including ID)
00703 c$$$               call rotMPB(zcousq,nbloch,ngbb,q,igx(ieqbz,k),igxt(ieqbz,k),ginv,zcousqrx)
       !zcousqr=Rotate_igx(zcousq)
```

```
00704 c$$$              zmelty = matmul(zmeltx,zcousqrx)
00705 c$$$              do igb2=1, ngb !.................................
00706 c$$$                zmelt2 = zmelty(igb2)
00707 c$$$              do igb1=1,igb2
00708 c$$$                z1p(igb1,igb2) = z1p(igb1,igb2) + dconjg(zmelty(igb1)) * zmelt2
00709 c$$$              enddo
00710 c$$$              enddo
00711 c$$$            enddo
00712 c$$$          else
00713 Cstop2rx 2013.08.09 kino          if (ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-1 >nwt) stop "x0kf_v4hz: iw>nwt"
00714           if (ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-1 >nwt) call rx( "x0kf_v4hz: iw>nwt")
00715
00716
00717 !kino 2014-08-13  !$OMP do private(zmelt2)
00718           do igb2=1, nmbas     !.................................
00719              zmelt2 = zmel(igb2,it,itp) !zmelt(igb2,it,itp)
00720              do igb1=1,igb2
00721 c                z1p(igb1,igb2) = dconjg(zmelt(igb1,it,itp)) * zmelt2
00722                z1p(igb1,igb2) = dconjg(zmel(igb1,it,itp)) * zmelt2
00723              enddo
00724           enddo
00725
00726 !! --------------------------------
00727           if(crpa) then
00728 c            print *,'readout readqkm init'
00729             if(n1b(ibib,k,jpm) <= nband) then
00730               call readpkm4crpa(n1b(ibib,k,jpm),   rk(:,k), isp_k,   wpw_k) !k  n1b
00731             else
00732               wpw_k=0d0
00733             endif
00734             if(n2b(ibib,k,jpm) <= nband) then
00735               call readpkm4crpa(n2b(ibib,k,jpm), q+rk(:,k), isp_kq,  wpw_kq) !kq n2b
00736             else
00737               wpw_kq=0d0
00738             endif
00739 c            print *,'readout readqkm=',wpw_k,wpw_kq
00740           endif
00741
00742 ccccccccccccccccccccccccccccc
00743 !kino 2014-08-13  !$OMP end do
00744 c$$$  endif
00745
00746 !$OMP parallel private(imagweight)
00747 !$OMP master
00748 !$        if (jpm.eq.1 .and. ibib.eq.nbnb(k,1)) then
00749 !$          write(6,'(a,i5,a,i5)') 'OMP parallel iw, threads=', omp_get_num_threads(),
00750 !$    .      ' nw=',ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-ihw(ibib,k,jpm)
00751 !$        endif
00752 !$OMP end master
00753 !$OMP do
00754           do iw = ihw(ibib,k,jpm),ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-1 !iiww=iw+ihw(ibib,k)-1
00755              imagweight = whw(jhw(ibib,k,jpm)+iw-ihw(ibib,k,jpm))
00756              if(crpa) imagweight = imagweight*(1d0-wpw_k*wpw_kq)
00757              if(eibzmode) imagweight = nwgt(k)*imagweight
00758              do igb2=1,nmbas      !this part dominates cpu time most time consuming...........
00759                 do igb1=1,igb2
00760                    rcxq(igb1,igb2,iw,jpm) = !here we  sum over ibib (or n, n') and k.
00761     &                   rcxq(igb1,igb2,iw,jpm) + z1p(igb1,igb2)*imagweight !sum over spin in hx0fp0
00762                 enddo           !igb1
00763              enddo              !igb2
00764           enddo                 ! iw
00765 !$OMP end do
00766 !$OMP end parallel
00767  25     continue
00768 !kino 2014-08-13  !$OMP end parallel
00769 !TIME1_1201 "after_rcxq"
00770
00771 c$$$ccccccccccccccccccccccccccccccccccccccccccccccc
00772 c$$$c        if(ipr) then
00773 c$$$          do jpm=1,npm
00774 c$$$            write(6,"(' k jpm sum(rcxq) ngb ngbb=',2i5,2d23.15,2i8)")
00775 c$$$     &      k,jpm,sum(rcxq(:,:,:,jpm)),ngb,ngbb
00776 c$$$          enddo
00777 c$$$          do ib1 =1,ngbb
00778 c$$$             if(ib1<4) then
00779 c$$$             elseif(ib1>ngbb-3) then
00780 c$$$             else
00781 c$$$                cycle
00782 c$$$             endif
00783 c$$$          do ib2 =1,ngbb
00784 c$$$             if(ib2<4) then
00785 c$$$             elseif(ib2>ngbb-3) then
00786 c$$$             else
00787 c$$$                cycle
00788 c$$$             endif
00789 c$$$          do iw =1,nwt
00790 c$$$            write(6,"('uuu: k iw ib1 ib2 sum(rcxq)=',4i5,4d23.15)")
```

```
00791 c$$$     &       k,iw,ib1,ib2,(rcxq(ib1,ib2,iw,1)), (rcxq(ib1,ib2,iw,2))
00792 c$$$          enddo
00793 c$$$          enddo
00794 c$$$          enddo
00795 c$$$c        endif
00796 c$$$ccccccccccccccccccccccccccccccccccccccccccccc
00797         deallocate(z1p,zmel) !zmelt,z1p)
00798         if(debug) call cputid(0)
00799         if(debug) write(6,*)' end of kloop k jpm=',k,jpm
00800  1000 continue
00801
00802 !! Not need to be symmetrized
00803       if(nolfco .and. nmbas==1) then
00804          write(6,*)' nmbas=1 nolfco=T ---> not need to symmetrize'
00805          goto 9999
00806       endif
00807 !TIME0_1301
00808
00809
00810 !! ==== Hermitianize. jun2012takao moved from dpsion5 ====
00811 c      if(eibzmode) then !comment out sep2014.
00812         do jpm=1,npm
00813         do iw= 1,nwt
00814          do igb2= 1,nmbas !eibzmode assumes nmbas1=nmbas2
00815          do igb1= 1,igb2-1
00816            rcxq(igb2,igb1,iw,jpm) = dconjg(rcxq(igb1,igb2,iw,jpm))
00817          enddo
00818          enddo
00819         enddo
00820         enddo
00821 c       endif
00822 !TIME1_1301 "before_eibzmode_symmetrization"
00823
00824 !! == End of eibzmode=F (no symmetrization required). ==
00825       goto 9999 ! finalize
00826
00827
00828
00829 !! ---------------------------------------------------------------------
00830 !! == Symmetrizer of EIBZ PRB.81,125102(2010) Eq.(51) july2012takao ==
00831 !! This may be not so effective ---> only for limited cases?
00832 !! --- zrotm(J,J') = <Mbar^k_J| \hat{A}^k_i Mbar^k_J'>. ---
00833 !! We do \sum_i T_alpha_i [ zrotm_i^dagger (I,I') P_I'J' zrom_i(J'J) ]
00834 !! (exactrly speaking, we insert conversion matrix between Enu basis and M_I basis).
00835 !!
00836 !! input qin = q
00837 !! \hat{A}^k_i  is specified by symops(:,:,igx),and igxt (-1 for time-reversal).
00838 !! Note that k= \hat{A}^k_i(k) (S_A^k)
00839 !! See Eq.(51) around in PRB81 125102(2010)
00840 !!
00841  5000 continue
00842 !! === zmelt conversion ===
00843       if(nolfco .and. nmbas==1) then
00844          write(6,*)' nmbas=1 nolfco=T ---> not need to symmetrize'
00845          goto 9999
00846       endif
00847 !!
00848       if(eibzmode.and.symmetrize) then
00849         ngb  = nbloch + ngc ! This is not ngbb for smbasis()=T. oct2005
00850         if(ngb/=ngbb) then
00851            write(6,*)' x0kf_v4h: ngb ngbb=',ngb,ngbb
00852            call rx( 'x0kf_v4h: ngb/=ngbb')
00853         endif
00854
00855 !TIME0_1401
00856         call iqindx2(q, ginv, qtt_, nqnum, ikp,quu) !to get ikp for timereversal mode
00857 !TIME1_1401 "after_iqindx2"
00858 !TIME0_1501
00859         if(sum(abs(q-quu))>1d-8) call rx( 'x0kf_v2h: eibz 111 q/quu')
00860         neibz = sum(eibzsym(:,1))+sum(eibzsym(:,-1))
00861              !itimer=-1 means time reversal. eibzsym(ig,itimer) where ig: space rotation.
00862         write(6,"(' --- goto symmetrization --- ikp neibz q=',2i3,3f12.8)")ikp,neibz,q
00863         call cputid2(' --- x0kf: start symmetrization  ',0)
00864
00865 c        allocate(rcxq0(ngb,ngb),rcxq00(ngb,ngb),rcxq000(ngb,ngb),rcxqwww(ngb,ngb))
00866         ntimer=1
00867         if(sum(eibzsym(:,-1))>0) ntimer=2 !timereversal case
00868         allocate(zrotm(ngb,ngb),nrotm(ngrp*2))
00869 !!
00870         zcousqinv=zcousq
00871         call matcinv(ngb,zcousqinv)
00872
00873 !! == Assemble rotantion matrx zrr,zrrc ==
00874 !! Rotation matrix zrrx can be a sparse matrix.
00875 !! Thus it is stored to  "i1(nrotmx,nccc),i2(nrotmx,nccc),zrr(nrotmx,icc),nrotm(icc)".
00876 !! See folloings: matmul(rcxqwww,zrrx) is given by
00877 !!     do irotm1 = 1,nrotm(icc)
```

```
00878 !!       rcxq0(:,i2(irotm1,icc)) = rcxqwww(:,i1(irotm1,icc)) * zrr(irotm1,i2(irotm1,icc))
00879
00880         allocate(zrrx(nmbas,nmbas))
00881         nrotmx = 10000 !trial value
00882 !TIME1_1501 "before_1011"
00883         do 1011  !this loop is only in order to to set large enough nrotmx.
00884 !TIME0_1601
00885         if(allocated(i1)) deallocate(i1,i2,zrr,zrrc)!,zrr_,zrrc_)
00886         nccc=ngrp*2
00887         allocate(i1(nrotmx,nccc),i2(nrotmx,nccc),zrr(nrotmx,nccc),zrrc(nrotmx,nccc))
      !,zrr_(ngb,ngb,nccc),zrrc_(ngb,ngb,nccc))
00888         i1=-99999
00889         i2=-99999
00890         zrr=-99999d0
00891         zrrc=-99999d0
00892         call cputid2(' --- x0kf:11111   :',0)
00893 !TIME1_1601 "allocatezrr"
00894 !!
00895         icc=0
00896         do itimer=1,-1,-2
00897           if(ntimer==1.and.itimer==-1) exit
00898           if(itimer==1 ) itt=1
00899           if(itimer==-1) itt=2
00900           do ig=1,ngrp
00901             if(eibzsym(ig,itimer)==1) then
00902               icc=icc+1
00903 !TIME0_1701
00904
00905 !! Get rotation matrix zrrx, which can be a sparse matrix. Thus stored to zrr.
00906               call rotmpb2(nbloch,ngb,q,ig,itimer,ginv,zrotm)
00907               if(nolfco.and.chipmzzr) then
00908 !!   We assume <svec_I | svec_J >= \delta_IJ, In addition, we use fact that we have no IPW parts in svec.
00909 !!   If IPW part exist, we may have to take into account <IPW|IPW> matrix, e.g. as in ppovlz.
00910 !!   svec --> zzr
00911                if(itimer==1) then
00912                  zrrx= matmul(transpose(dconjg(zzr)), matmul(zrotm, zzr))
00913                else
00914                  zrrx= matmul(transpose(zzr), matmul(dconjg(zrotm), zzr))
00915                endif
00916               elseif(nolfco) then
00917                  call rx( 'x0kf_v4h: this case is not implemented xxxxxxxxxxxxx')
00918               else
00919 !! zrotm(J,J') is the rotation matrix = <Mbar^k_J| \hat{A}^k_i Mbar^k_J'>
00920 !! See rotMPB2 defined in readeigen.F.
00921 !! zrrx(mu nu)= dconjg(Zcousq(I, mu)) *zrotm(I,J)* Zcousq(J, nu)
00922 !! zrrx is very sparse matrix. Size is \sim ngb or something.
00923
00924 c$$$                if(itimer==1) then
00925 c$$$                  call matmmsparse(zcousqinv,zrotm,zcousq,zrrx,ngb,1d-8,iele)
00926 c$$$                  ! this means zrrx= matmul(zcousqinv,matmul(zrotm, zcousq))
00927 c$$$                else
00928 c$$$                  call matmmsparse(dconjg(zcousqinv),dconjg(zrotm),zcousq,zrrx,ngb,1d-8,iele)
00929 c$$$                  ! this means zrrx= matmul(dconjg(zcousqinv),matmul(dconjg(zrotm), zcousq))
00930 c$$$                endif
00931
00932                 if(itimer==1) then
00933                   zrrx=zrotm
00934 c                  call matmmsparse(zcousqinv,zrotm,zcousq,zrrx,ngb,1d-8,iele)
00935                   ! this means zrrx= matmul(zcousqinv,matmul(zrotm, zcousq))
00936                 else
00937                   zrrx=dconjg(zrotm)
00938 c                  call matmmsparse(dconjg(zcousqinv),dconjg(zrotm),zcousq,zrrx,ngb,1d-8,iele)
00939                   ! this means zrrx= matmul(dconjg(zcousqinv),matmul(dconjg(zrotm), zcousq))
00940                 endif
00941
00942               endif
00943 !TIME1_1701 "end_matmmsparse"
00944 !TIME0_1801
00945               i1(:,icc)=0
00946               i2(:,icc)=0
00947               irotm=0
00948               iagain=0
00949               do ix=1,ngb
00950               do iy=1,ngb
00951                 if(abs(zrrx(ix,iy))>1d-8) then
00952                   irotm=irotm+1
00953                   if(irotm>nrotmx) then
00954                     iagain=1
00955                   endif
00956                   if(iagain/=1) then
00957                   i1(irotm,icc)=ix
00958                   i2(irotm,icc)=iy
00959                   zrr(irotm,icc) = zrrx(ix,iy)
00960                   zrrc(irotm,icc)= dconjg(zrr(irotm,icc))
00961                   endif
00962                 endif
00963               enddo
```

```
00964               enddo
00965 !TIME1_1801 "before_iagain1"
00966 !TIME0_1901
00967               if(iagain==1) then
00968                 nrotmx=irotm !enlarge allocation and do things again.
00969                 write(6,*)' warn:(slow speed) xxxx goto 1011 xxxxxx nrotmx+=nrotmx+10000 again'
00970                 goto 1011
00971                 !enlarge nrotmx ang try it again.
00972               endif
00973               nrotm(icc)=irotm
00974               if(debug) write(6,*)'ig itimer icc nrotm=',ig,itimer,icc,nrotm(icc) ,iele
00975 !TIME1_1901 "end_ig_itimer_icc_nrotm"
00976             endif
00977           enddo
00978         enddo
00979         exit
00980  1011   continue !only when nrotmx overflow.
00981 !TIME0_2001
00982
00983 !! === main part to obtain symmetrized rcxq  ===
00984 !! neibz is total number of symmetrization operation.
00985 !!      rcxq is rotated and accumulated; finally divied by neibz
00986         zcousqc = dconjg(transpose(zcousq))
00987         call cputid2(' --- x0kf:qqqqq222ini:',0)
00988 !$OMP parallel private(rcxq000,icc,itt,icount,rcxqwww,rcxq00,rcxq0,rcxq_core)
00989         allocate(rcxq0(ngb,ngb),rcxq00(ngb,ngb),rcxq000(ngb,ngb),rcxqwww(ngb,ngb),rcxq_core(ngb,ngb))
00990 !$OMP master
00991 !$         write(6,'(a,i5,a,i5)') 'OMP parallel nwt, threads=',omp_get_num_threads(),' nwt=',nwt
00992 !$OMP end master
00993 !$OMP do
00994         do iw=1,nwt
00995         do jpm=1,npm
00996           rcxq000 = 0d0
00997           icc=0
00998           do itimer=1,-1,-2
00999             if(itimer==1 ) itt=1
01000             if(itimer==-1) itt=2
01001             icount=0
01002             if(itimer==1) then
01003               rcxqwww = rcxq(:,:,iw,jpm)
01004             else
01005               rcxqwww = transpose(rcxq(:,:,iw,jpm))
01006             endif
01007             rcxq00 = 0d0
01008             do ig=1,ngrp
01009               if(eibzsym(ig,itimer)==1) then
01010                icount=icount+1
01011                icc=icc+1
01012                rcxq0 =0d0
01013
01014 c$$$               if(itimer==1) then
01015 c$$$               do irotm1 = 1,nrotm(icc)
01016 c$$$               do irotm2 = 1,nrotm(icc)
01017 c$$$               rcxq0(i2(irotm2,icc),i2(irotm1,icc)) =rcxq0(i2(irotm2,icc),i2(irotm1,icc))
01018 c$$$     &              +    zrrc(irotm2,icc)* rcxq(i1(irotm2,icc),i1(irotm1,icc),iw,jpm)*zrr(irotm1,icc)
01019 c$$$               enddo
01020 c$$$               enddo
01021 c$$$               else
01022 c$$$               do irotm1 = 1,nrotm(icc)
01023 c$$$               do irotm2 = 1,nrotm(icc)
01024 c$$$               rcxq0(i2(irotm1,icc),i2(irotm2,icc)) =rcxq0(i2(irotm1,icc),i2(irotm2,icc)) !transpose
01025 c$$$     &              +    zrrc(irotm2,icc)* rcxq(i2(irotm2,icc),i1(irotm1,icc),iw,jpm)*zrr(irotm1,icc)
01026 c$$$               enddo
01027 c$$$               enddo
01028 c$$$               endif
01029
01030 !!  Followings are equivalent with
01031 !!            rcxq00= rcxq00 + matmul(zrrc_(:,:,icc),matmul(rcxqwww,zrr_(:,:,icc)))
01032               do irotm1 = 1,nrotm(icc)
01033 c                if(abs(zrr(irotm1,icc))<1d-8) cycle
01034                 rcxq0(:,i2(irotm1,icc)) =rcxq0(:,i2(irotm1,icc)) + rcxqwww(:,i1(irotm1,icc)) * zrr(irotm1,
      icc)
01035               enddo
01036               do irotm2 = 1,nrotm(icc)
01037 c                if(abs(zrrc(irotm2,icc))<1d-8) cycle
01038                 rcxq00(i2(irotm2,icc),:)= rcxq00(i2(irotm2,icc),:) + zrrc(irotm2,icc) * rcxq0(i1(irotm2,
      icc),:)
01039               enddo
01040
01041 c              if(itimer==1) then
01042 c                rcxq000 = rcxq000 + rcxq00
01043 c              else
01044 c                rcxq000 = rcxq000 + transpose(rcxq00)
01045 c              endif
01046 c
01047 c$$$              do irotm = 1,nrotm(icc)
01048 c$$$               iyy = i1(irotm,icc)
```

```
01049 c$$$                iy  = i2(irotm,icc)
01050 c$$$                rcxq0(:,iy)= rcxq0(:,iy)+ rcxq(:,iyy,iw,jpm)* zrr(irotm,icc)
01051 c$$$               enddo
01052 c$$$               do irotm = 1,nrotm(icc)
01053 c$$$                iyy = i1(irotm,icc)
01054 c$$$                iy  = i2(irotm,icc)
01055 c$$$                rcxq00(iy,:)= rcxq00(iy,:)+ dconjg(zrr(irotm,icc)) * rcxq0(iyy,:)
01056 c$$$               enddo
01057 c$$$
01058 ccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01059 c               if(iw==1.and.jpm==1) then
01060 c                  write(6,"('bbbbbbb ig icc iw jpm rcxq', 4i3, 13d13.6)")
01061 c     &                 ig,icc,iw,jpm, sum(abs(rcxq00)), rcxq00(1,1),sum(abs(rcxqwww)),sum((rcxqwww))
01062 c               endif
01063 ccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01064
01065              endif
01066            enddo
01067
01068 c$$$           if(itimer==1) then
01069 c$$$             rcxq000(:,:) = matmul(zcousqc,matmul(rcxq00,zcousq))
01070 c$$$c$$$c               call zgemm("N","N",ngb,ngb,ngb, (1d0,0d0), rcxq00, ngb, zcousq,ngb, (0d0,0d0),
       rzc,ngb)
01071 c$$$c$$$c               call zgemm("N","N",ngb,ngb,ngb, (1d0,0d0), zcousqc,ngb, rzc,ngb, (0d0,0d0),
       rcxq000,ngb)
01072 c$$$           elseif(icount>0) then
01073 c$$$c$$$c           write(6,*)'qqqqq icount=',icount
01074 c$$$c$$$c           rcxq000(:,:) = rcxq000(:,:) +
       transpose(matmul(transpose(zcousq),matmul(rcxq00,dconjg(zcousq))))
01075 c$$$             rcxq000(:,:) = rcxq000(:,:) +   matmul(matmul(zcousqc,transpose(rcxq00)),zcousq)
01076 c$$$           endif
01077
01078            if(itimer==1) then
01079               rcxq000=rcxq00
01080            else
01081               rcxq000=rcxq000+rcxq00
01082            endif
01083          enddo
01084          rcxq_core = rcxq000/neibz
01085 #if 1
01086 !! matmul(rcxq(:,:,iw,jpm),zcousq) fails in ifort 14.0.3.
01087 !! It looks that ifort 14.0.3 has a bug
01088 !! But, zgemm works. So I changed like that.
01089          call zgemm('N','N',ngb,ngb,ngb,(1.0d0,0.0d0),rcxq_core,ngb,zcousq, ngb, (0.0d0,0.0d0),rcxq000,ngb
     )
01090          call zgemm('N','N',ngb,ngb,ngb,(1.0d0,0.0d0),zcousqc  ,ngb,rcxq000,ngb, (0.0d0,0.0d0),rcxq_core,
     ngb)
01091          rcxq(:,:,iw,jpm) = rcxq_core
01092 #else
01093          rcxq(:,:,iw,jpm) = matmul(zcousqc,matmul(rcxq_core,zcousq))
01094 #endif
01095          enddo
01096          enddo
01097 !$OMP end  do
01098        deallocate(rcxq00,rcxq000,rcxq0,rcxqwww)
01099 !$OMP end parallel
01100 !TIME1_2001 "after_sym_rcxq"
01101        deallocate(zrotm,i1,i2)
01102
01103 c$$$       allocate(zcousqr(ngb,ngb,neibz),rcxq0(ngb,ngb),rcxq00(ngb,ngb),rcxqtr(ngb,ngb))
01104 c$$$       icc=0
01105 c$$$       do itimer=1,-1,-2
01106 c$$$       do ig=1,ngrp
01107 c$$$         if(eibzsym(ig,itimer)==1) then
01108 c$$$           icc=icc+1
01109 c$$$           if(itimer==1) then
01110 c$$$             call rotMPB(zcousq,nbloch,ngb,q,ig,itimer,ginv,zcousqr(1,1,icc))
01111 c$$$           else
01112 c$$$!! time reversal mapping ---
01113 c$$$             call rotMPB(dconjg(zcousq),nbloch,ngb,q,ig,itimer,ginv,zcousqr(1,1,icc))
01114 c$$$           endif
01115 c$$$         endif
01116 c$$$       enddo
01117 c$$$       enddo
01118 c$$$
01119 c$$$       do iw=1,nwt
01120 c$$$       do jpm=1,npm
01121 c$$$         rcxq0=0d0
01122 c$$$         icc=0
01123 c$$$c         do itimer=1,1 !1,-1,-2
01124 c$$$         do itimer=1,-1,-2
01125 c$$$         do ig=1,ngrp
01126 c$$$           if(eibzsym(ig,itimer)==1) then
01127 c$$$            icc=icc+1
01128 c$$$            rcxq00(:,:) = matmul(dconjg(transpose(zcousqr(:,:,icc))),
01129 c$$$     &                           matmul(rcxq(:,:,iw,jpm),zcousqr(:,:,icc)))
01130 c$$$!! time reversal mapping ---
```

```
01131 c$$$             if(itimer==-1) rcxq00(:,:) = transpose(rcxq00)
01132 c$$$             rcxq0(:,:) = rcxq0(:,:)+ rcxq00(:,:)
01133 c$$$           endif
01134 c$$$         enddo
01135 c$$$         enddo
01136 c$$$         rcxq(:,:,iw,jpm)=rcxq0(:,:)/neibz
01137 c$$$       enddo
01138 c$$$       enddo
01139 c$$$       deallocate(zcousqr,rcxq0,rcxq00,rcxqtr)
01140        call cputid2(' --- qqqqq222end:',0)
01141      endif
01142
01143  9999 continue
01144 !kino 2014.08.19 use automatic deallocation,      deallocate(cphi_k,cphi_kq,geig_kq,geig_k)
01145 c      if(smbasis()) deallocate(pomat)
01146      write(6,"(' --- x0kf_v4hz: end')")
01147      end subroutine x0kf_v4hz
01148
01149
01150
01151
01152 C=====================================================================
01153        subroutine dpsion5  (frhis,nwhis, freqr,nw_w, freqi,niwt,
01154     i                                realomega,   imagomega,      !freqr ->frhis ...sf
01155     i                       rcxq, npm, nw_i,nmbas1,nmbas2,
01156     o    zxq,zxqi,
01157 c     i    nolfco,chipm,schi,isp,  rcxqmean,nmbas,  !iepsmode, rcxqmean, ! epsmode
01158     i    chipm,schi,isp,   !No nolfco mode. Apr2012.
01159     i    ecut,ecuts)
01160 c     o    x0mean)
01161 C- Calculate W-v zxqi(on the imaginary axis) and zxq(real axis) from sperctum weight rcxq.
01162 Cr v4 works for timereversal=F (npm=2 case).
01163 Cr  See rcxq_zcxq for rcxq, which contains the spectrum weight for given bins along the real-axis.
01164 Cr ! Note that zxq and zxqi are not accumlating
01165 Ci frhis(1:nwhis+1) :: specify histgram bins i-th bin is [frhis(i), frhis(i+1)].
01166 Ci          We suppose "freqr(i)=moddle of i-th bin; freqr(0)=0."
01167 Ci          (I think called routine hilbertmat itself is not limited by this condition).
01168 Ci freqr (0:nw_w) : Calcualte zxq for these real energies.
01169 Ci freqi (1:niwt) : Calcualte zxqi for these imaginary energies.
01170 Ci   realomega  : A switch to calculate zxq or not.
01171 Ci   imagomega: : A switch to calculate zxqi or not.
01172 Ciw rcxq may be altered ---used as work area.
01173 Cio   zxq :  W-v along the real axis on freqr(0:nw_w)
01174 Cio   zxqi:  W-v along the imag axis on freqi(niwt)
01175 C!
01176 C! Feb2006:  v4 for timereversal=F
01177 C! July2005: v3Add spin chipm mode
01178 C! July2005: This version alter rcxq----it is used as work area.
01179 C! sergey faleev Apr 2002 ; Rebuiled by takao
01180 C---------------------------------------------------------------------
01181        implicit none
01182        integer(4):: nw_w,niwt,igb1,igb2, iw,iwp,nwhis,ix,npm,ifxx,nmbas1,nmbas2
01183        real(8) :: freqi(niwt),pi,px,omp,om,om2,om1, !omg2max from hx0fp0
01184     &  frhis(nwhis+1), freqr(0:nw_w), aaa,d_omg
01185        logical :: realomega, imagomega
01186        complex(8):: rcxq(nmbas1,nmbas2, nwhis,npm) !sf 13June
01187 c     logical   :: iepsmode
01188        logical :: chipm
01189
01190        integer(4)::isp,ispx !, nmbas
01191 c     complex(8):: rcxqmean(nwhis,npm,nmbas,nmbas)  !takao sep2006 add nmbas
01192 C... ecut mode
01193        real(8):: ecut,ecuts,wcut,wcutef,dee,schi
01194        logical ::debug=.false.
01195        real(8),allocatable :: his_l(:),his_r(:),his_c(:)
01196        integer(4) it
01197        real(8):: domega_r,domega_c,domega_l,delta_l,delta_r
01198        real(8),allocatable ::rmat(:,:,:),rmati(:,:,:),rmatt(:,:,:),imatt(:,:,:)
01199        complex(8),allocatable :: rmatic(:,:,:),imattc(:,:,:)
01200        complex(8) ::beta,wfac
01201        complex(8):: zz
01202        complex(8),allocatable :: zxqn(:,:),zxqn1(:,:,:),rx0mean1(:,:,:),rx0mean(:)
01203        complex(8),allocatable:: rrr(:)
01204
01205        integer(4)::nw_i,jpm,ipm,verbose,isgi
01206 c     complex(8):: x0mean(nw_i:nw_w,nmbas,nmbas)
01207        complex(8)::
01208     o   zxq(nmbas1,nmbas2, nw_i: nw_w), !iw=0 means omg=0,
01209        !iw=1:nw_w corresponds to iw's bit of the frequensy histogram
01210     o   zxqi(nmbas1,nmbas2,niwt),img !npm), img  !zxqi(...,npm) may2006
01211
01212        real(8),allocatable:: ebb(:)
01213        integer(4):: ii,i,ibas1,ibas2
01214        logical :: evaltest !,testtr
01215
01216 c       if(verbose()>89) debug=.true.
01217 c -------------------------------------------------
```

```
01218         write(6,'(" -- dpsion5: start...   ",$)')
01219         write(6,"('  nw_w nwhis=',2i5)") nw_w,nwhis
01220         if(debug) then
01221           write(6,*)' nmbas1 nmbas2 nwhis npm =',  nmbas1,nmbas2,nwhis,npm
01222           write(6,*)' sumchk rcxq=', sum(abs(rcxq))
01223         endif
01224         pi  = 4d0*datan(1d0)
01225         img = (0d0,1d0)
01226         call cputid(0)
01227         ispx = isp
01228         if(schi<0) then
01229           ispx = 3-isp !flip
01230         endif
01231
01232 C... Check freqr=frhis_m.
01233 C... But I think now this is not necessary. You can supply any freqr(iw). But be careful.
01234         if(realomega) then
01235           if( nwhis <= nw_w ) then
01236             write(6,*)nwhis,nw_w
01237 Cstop2rx 2013.08.09 kino            stop ' dpsion5: nwhis<=nw_w'
01238             call rx( ' dpsion5: nwhis<=nw_w')
01239           endif
01240 Cstop2rx 2013.08.09 kino          if( freqr(0)/=0d0 ) stop ' dpsion5: freqr(0)/=0d0'
01241           if( freqr(0)/=0d0 ) call rx( ' dpsion5: freqr(0)/=0d0')
01242           aaa = 0d0
01243           if(nw_w>0) then
01244             do iw = 1,nw_w
01245               aaa = aaa + abs( freqr(iw) - (frhis(iw)+frhis(iw+1))/2d0 )
01246               if(debug) write(6,"(' iw freqr frhis_m=',i5,2f13.6)" )
01247      &     iw,freqr(iw),  (frhis(iw)+frhis(iw+1))/2d0
01248             enddo
01249 Cstop2rx 2013.08.09 kino            if(aaa>1d-10)stop 'dpsion5:freqr/=frhis_m is not implimented yet'
01250             if(aaa>1d-10)call rx( 'dpsion5:freqr/=frhis_m is not implimented yet')
01251           endif
01252         endif !realomega
01253
01254 C-------------------------------------------------------------
01255 ! Each bin  [his_Left, his_Right]  his_Center is middle.
01256 ! his_C(0) is at zero. his_R(0) and his_L(0) are not defined.
01257         if(debug) write(6,*)' dpsion5: RRR 2222222222 '
01258         allocate(his_l(-nwhis:nwhis),his_r(-nwhis:nwhis),his_c(-nwhis:nwhis))
01259         his_l(1:nwhis) = frhis(  1:  nwhis)
01260         his_r(1:nwhis) = frhis(1+1:1+nwhis)
01261         his_c(1:nwhis) = (his_l(1:nwhis) + his_r(1:nwhis) )/2d0
01262         do iw= 1,nwhis
01263           his_l(-iw) = -his_r(iw)
01264           his_r(-iw) = -his_l(iw)
01265           his_c(-iw) = -his_c(iw)
01266         enddo
01267         his_c(0) = 0d0; his_r(0)=-999; his_l(0)=-999
01268 C
01269         if(debug) write(6,*)'sumchk 111 rcxq=', sum(abs(rcxq))
01270
01271         do iw= 1, nwhis
01272           if(ecut<1d9) then
01273             wfac= wcutef(his_c(iw), ecut,ecuts)
01274           else
01275             wfac= 1d0
01276           endif
01277 ! rcxq is used as work---> rcxq= Average value of Im chi.
01278 ! Note rcxq is "negative" (
01279           do jpm=1,npm
01280             call dscal(2*nmbas1*nmbas2, -wfac/(his_r(iw)-his_l(iw)),rcxq(1,1,iw,jpm),1)
01281           enddo
01282 c         if(debug) write(6,*) 'dpsion5: RRR 7777 iw wfac=',iw,wfac,ecut,ecuts
01283         enddo
01284         if(debug) write(6,*)'sumchk 122 rcxq=', sum(abs(rcxq))
01285
01286 C... Temporary. maybe, we will have better procedure...
01287 ctakao moved this to x0kv_v4h.F jun2012takao
01288 !! hermitianize.
01289 c        if(nmbas1==nmbas2) then !Is this required??? apr2012takao
01290 c          do jpm=1,npm
01291 c            do iw= 1, nwhis
01292 c              do igb2= 1, nmbas2
01293 c                do igb1= 1, igb2-1
01294 c                  rcxq(igb2,igb1,iw,jpm) = dconjg(rcxq(igb1,igb2,iw,jpm))
01295 c                enddo
01296 c              enddo
01297 c            enddo
01298 c          enddo
01299 c        endif
01300 ccccccccccccccccccc
01301         if(debug) write(6,*)'sumchk 222 rcxq=', sum(abs(rcxq))
01302
01303         if(evaltest().and.nmbas1==nmbas2) then
01304           write(6,"('hhh --- EigenValues for rcxq --------')")
```

```
01305           allocate(ebb(nmbas1))
01306           do jpm= 1,npm
01307             do iw = 1, nwhis
01308               call diagcvh2(rcxq(:,:,iw,jpm),nmbas1,ebb)
01309               do ii=1,nmbas1
01310                 write(6,"('hhh1: xxxxxxxxxxxxxxxx',2i4)") jpm,iw
01311                 if(abs(ebb(ii))>1d-8.and.ebb(ii)>0)
01312      &          write(6,"('hhh1: jpm iw eb=',2i4,d13.5)") jpm,iw,ebb(ii)
01313               enddo
01314             enddo
01315           enddo
01316           deallocate(ebb)
01317         endif
01318
01319 C--- realomega case
01320         if(realomega)then
01321           write(6,*) " --- realomega --- "
01322           if(npm==1) then
01323             allocate( rmat(0:nw_w,-nwhis:nwhis,npm), rrr(-nwhis:nwhis))
01324             rmat  = 0d0
01325             do it =  0,nw_w
01326               zz = freqr(it) !his_C(it)
01327               call hilbertmat(zz, nwhis,his_l,his_c,his_r, rrr)
01328               rmat(it,:,1) = dreal(rrr)
01329             enddo ;   if(debug) write(6,*) 'dpsion5: RRR 55555555555'
01330             allocate( rmatt(0:nw_w,nwhis,npm) )
01331             if(     chipm.and.ispx==1 ) then
01332               rmatt(:,:,1) = rmat(:,1:nwhis,1)
01333             elseif( chipm.and.ispx==2 ) then
01334               do iw= 1,nwhis
01335                 rmatt(:,iw,1) = -rmat(:,-iw,1)
01336               enddo
01337             else
01338               do iw= 1,nwhis
01339                 rmatt(:,iw,1) = rmat(:,iw,1) - rmat(:,-iw,1)
01340               enddo
01341             endif
01342             deallocate(rmat,rrr)
01343           else ! npm==2 case ------------------------------------------------
01344             allocate( rmatt(-nw_w:nw_w,nwhis,npm), rrr(-nwhis:nwhis))
01345             rmatt = 0d0
01346             do it  =  -nw_w,nw_w
01347               if(it<0) then
01348                 zz = -freqr(-it) !his_C(it)
01349               else
01350                 zz = freqr(it) !his_C(it)
01351               endif
01352               call hilbertmat(zz, nwhis,his_l,his_c,his_r, rrr)
01353               rmatt(it,:,1) =  dreal(rrr(1:nwhis))
01354               rmatt(it,:,2) = -dreal(rrr(-1:-nwhis:-1))
01355             enddo ;   if(debug) write(6,*) 'dpsion5: RRR2 55555555555'
01356             deallocate(rrr)
01357           endif
01358           rmatt = rmatt/pi ; if(debug) write(6,*)'dpsion5: RRR 6666'
01359
01360 !! takao remove if(nolfc) block here.
01361 c          write(6,*) " --- realomega dgemm--- "
01362
01363
01364 !! WARN! I think npm==2.and.chipm does not make sense. apr2012.
01365 !!
01366           if(npm==2.and.chipm)
01367 Cstop2rx 2013.08.09 kino     &      stop 'x0kf_v4h:npm==2.and.chipm is not meaningful probably'
01368      &      call rx( .and.'x0kf_v4h:npm==2chipm is not meaningful probably')
01369
01370
01371 !! Note rcxq is negative now (converted at the top of this routine !!!
01372           if(     chipm .and. ispx==2 ) then
01373             !nothing here
01374             !Since the range of zxq is nw_i=0, we have no area to store negative energy part of chipm.
01375           elseif( chipm               ) then
01376             call zaxpy( nmbas1*nmbas2*nw_w, img, rcxq, 1, zxq(1,1,1), 1)
01377           else
01378             zxq = 0d0    ! not accumlating case.
01379             call zaxpy( nmbas1*nmbas2*nw_w, img, rcxq(1,1,1,1), 1, zxq(1,1,1), 1)
01380           endif
01381
01382           if(npm==2) then
01383             do iw=1,nw_w
01384               call zaxpy( nmbas1*nmbas2, img, rcxq(1,1,iw,2),1, zxq(:,:,-iw),1)
01385             enddo
01386           endif
01387
01388           if(npm==1) then
01389             call dgemm('n','t',  2*nmbas1*nmbas2, nw_w+1, nwhis, 1d0,
01390      &               rcxq, 2*nmbas1*nmbas2,  rmatt, nw_w+1,
01391      &                1d0, zxq, 2*nmbas1*nmbas2 )
```

```
01392           elseif(npm==2) then
01393             call dgemm('n','t',  2*nmbas1*nmbas2,   npm*nw_w+1, nwhis, 1d0,
01394      &              rcxq(1,1,1,1), 2*nmbas1*nmbas2, rmatt(:,:,1), npm*nw_w+1,
01395      &               1d0, zxq, 2*nmbas1*nmbas2 )
01396             call dgemm('n','t',  2*nmbas1*nmbas2,   npm*nw_w+1, nwhis, 1d0,
01397      &              rcxq(1,1,1,2), 2*nmbas1*nmbas2, rmatt(:,:,2), npm*nw_w+1,
01398      &               1d0, zxq, 2*nmbas1*nmbas2 )
01399           else
01400 Cstop2rx 2013.08.09 kino            stop 'dpsion5: npm=1 or 2'
01401             call rx( 'dpsion5: npm=1 or 2')
01402           endif
01403           deallocate(rmatt)
01404         endif
01405
01406 !! === imagomega case      imatt(niwt -->niwt,npm may2005 ===
01407         if(imagomega) then
01408           allocate( rrr(-nwhis:nwhis))
01409           if(npm==1) then
01410             allocate( rmati(niwt,-nwhis:nwhis,npm))
01411             rmati= 0d0
01412           else
01413             allocate( rmatic(niwt,-nwhis:nwhis,npm))
01414             rmatic = 0d0
01415           endif ;   if(debug) write(6,*) 'dpsion5: III 111111155555555555'
01416           do it =  1,niwt
01417             zz = img*freqi(it) !his_C(it)
01418             call hilbertmat(zz, nwhis,his_l,his_c,his_r, rrr) !Im(zz)>0
01419             if(npm==1) then
01420               rmati(it,:,1) = dreal(rrr)
01421             else
01422               rmatic(it,:,1) = rrr
01423             endif
01424           enddo ;   if(debug) write(6,*) 'dpsion5: III 55555555555'
01425 !! ==== npm=1 case ====
01426           if(npm==1) then
01427             allocate( imatt(niwt, nwhis,npm) )
01428             do iw= 1,nwhis
01429               imatt(:,iw,1) = rmati(:,iw,1) - rmati(:,-iw,1)
01430             enddo
01431             deallocate(rmati,rrr)
01432             imatt = imatt/pi; if(debug) write(6,*) 'dpsion5: III  '
01433             call dgemm('n','t',  2*nmbas1*nmbas2, niwt, nwhis, 1d0,
01434      &              rcxq, 2*nmbas1*nmbas2, imatt, niwt,
01435      &               0d0, zxqi, 2*nmbas1*nmbas2 )
01436             deallocate(imatt)
01437 !! ==== npm=2 case ====
01438           else
01439             allocate( imattc(niwt, nwhis,npm) )
01440             do iw= 1,nwhis
01441               imattc(:,iw,1) =   rmatic(:, iw,1)
01442               imattc(:,iw,2) = - rmatic(:,-iw,1)
01443             enddo
01444             deallocate(rmatic,rrr)
01445             imattc = imattc/pi; if(debug) write(6,*) 'dpsion5: IIIc '
01446             call zgemm('n','t',  nmbas1*nmbas2, niwt, nwhis, 1d0,
01447      &              rcxq(1,1,1,1), nmbas1*nmbas2, imattc(1,1,1), niwt,
01448      &              0d0, zxqi,    nmbas1*nmbas2 )
01449             call zgemm('n','t',  nmbas1*nmbas2, niwt, nwhis, 1d0,
01450      &              rcxq(1,1,1,2), nmbas1*nmbas2, imattc(1,1,2), niwt,
01451      &              1d0, zxqi,    nmbas1*nmbas2 )
01452             deallocate(imattc)
01453           endif
01454         endif
01455         deallocate(his_l,his_c,his_r)
01456         write(6,'("          end dpsion5 ",$)')
01457         call cputid(0)
01458         end
01459         logical function checkbelong(qin, qall, nq,ieibz) !ieibz is also returned
01460         integer:: nq,ieibz
01461         real(8):: qin(3), qall(3,nq)
01462         checkbelong=.false.
01463         do i=1,nq
01464            if(sum(abs(qin-qall(:,i)))<1d-8) then
01465               ieibz=i
01466               checkbelong=.true.
01467               return
01468            endif
01469         enddo
01470         end
01471
01472
01473 !!-------------------------------------------------------------------------
01474       subroutine hilbertmat (zz,nwhis, his_L,his_C,his_R, rmat)
01475 C- Martix for hilbert transformation, rmat.
01476 Cr  zz is real--->  no img*delta function part
01477 Cr   zz is complex (and Im(zz)>0) : includes all contribution when Im(zz)>eps
01478 Co  rmat(-nwhis:nwhis) : rmat(0) is not meaningful.
```

```
01479 Ci i-th Histgram bin on real axis are given by [his_L, his_R]. center is his_C.
01480 Cr f(zz) = \int_-x(nwhis)^x(nwhis) f(x)/(zz-x)
01481 Cr       = \sum_{i/=0} rmat(i)*f(i)
01482 Cr     ,where f(i) is the average value at i-th bin.
01483 C!!! 23May2006 I think
01484 C!!! rmat is --------------
01485 C!!! f(zz) = - \int_-x(nwhis)^x(nwhis) f(x)/(zz-x)
01486 C!!!       = - \sum_{i/=0} rmat(i)*f(i)
01487 C I forgot minus sign in the previous note.
01488 C-------------------------------
01489       implicit none
01490       integer(4):: iw,nwhis
01491       complex(8) ::zz,imgepsz
01492       real(8)    :: his_l(-nwhis:nwhis),his_c(-nwhis:nwhis),his_r(-nwhis:nwhis)
01493       complex(8) :: rr_fac(-nwhis:nwhis),rl_fac(-nwhis:nwhis),img=(0d0,1d0)
01494       real(8)::  eps=1d-8, epsz=1d-13,delta_r,delta_l,ddr,ddl
01495       complex(8):: domega_c,domega_r,domega_l
01496       complex(8) ::  rmat(-nwhis:nwhis)
01497       imgepsz =img*epsz
01498       do iw = -nwhis, nwhis
01499         if(iw==0) cycle
01500         domega_r = zz - his_r(iw) + imgepsz
01501         domega_c = zz - his_c(iw) + imgepsz
01502         domega_l = zz - his_l(iw) + imgepsz
01503         if( abs(domega_c)<eps .or. abs(domega_r)<eps ) then
01504           rr_fac(iw) = 0d0
01505         else
01506 ! rr_fac(his_C(is)) = \int^{his_R}_{his_C} d omega' /(his_C(is) -omega')
01507 c            rr_fac(iw) = log( abs((domega_r/domega_c)) )
01508           rr_fac(iw) = log( domega_r/domega_c )
01509         endif
01510         if( abs(domega_c)<eps .or. abs(domega_l)<eps ) then
01511           rl_fac(iw) = 0d0
01512         else
01513 ! rl_fac(his_C(is)) = \int^{his_C}^{his_L} d omega' /(his_C(is) -omega')
01514 c            rl_fac(iw) = log( abs((domega_c/domega_l)) )
01515           rl_fac(iw) = log( domega_c/domega_l)
01516         endif
01517       enddo
01518       rmat=0d0
01519       do iw = -nwhis, nwhis !symmetric version. iw=0 is meaningless
01520         if(iw==0) cycle
01521 c          if(debug) print *,' it iw=',it, iw
01522         domega_c = zz - his_c(iw)
01523         if(iw==  nwhis) then
01524           delta_r = his_r(iw)   - his_c(iw)
01525         elseif(iw== -1) then
01526           delta_r =   0d0       - his_c(iw)
01527         else
01528           delta_r = his_c(iw+1) - his_c(iw)
01529         endif
01530 !         if(debug) print *,' it iw RRR1'
01531         if(iw== -nwhis) then
01532           delta_l = his_c(iw)  - his_l(iw)
01533         elseif(iw==  1) then
01534           delta_l = his_c(iw)  - 0d0
01535         else
01536           delta_l = his_c(iw)  - his_c(iw-1)
01537         endif
01538 !         if(debug) print *,' it iw RRR2'
01539 !          ddr = (his_R(iw)-his_C(iw))/delta_r
01540 !          ddl = (his_C(iw)-his_L(iw))/delta_l
01541         rmat(iw)  = rmat(iw  ) + rr_fac(iw)*( 1d0-domega_c/delta_r) !+ ddr
01542         if(iw/=nwhis.and.iw/=-1) then
01543           rmat(iw+1) = rmat(iw+1) + rr_fac(iw)*domega_c/delta_r     !- ddr
01544         endif
01545         rmat(iw)  = rmat(iw) + rl_fac(iw)*( 1d0+domega_c/delta_l)   !- ddl
01546         if(iw/=-nwhis.and. iw/=1) then
01547           rmat(iw-1) = rmat(iw-1) - rl_fac(iw)*domega_c/delta_l     !+ ddl
01548         endif
01549 ccccccccccccccccccccccc
01550 c no-derivarive test
01551 c          rmat(iw)  =  rr_fac(iw) + rl_fac(iw)
01552 ccccccccccccccccccccccc
01553       enddo
01554       end
01555
01556 c$$$      subroutine reducezmel(aold, ngbo,ngb,nx,
01557 c$$$     i       io,   in,   nmat, pmat,
01558 c$$$     i     io_q, in_q, nmat_q, pmat_q,
01559 c$$$     o      anew)
01560 c$$$c   For given q+G basis, we augment the basis within MT.
01561 c$$$c   For given atom and l  prod and prodd at MT boundary (reserved in PPBRD
01562 c$$$      integer(4):: nmat,io(nmat),in(nmat),nmat_q,io_q(nmat),in_q(nmat)
01563 c$$$      complex(8):: aold(ngbo,nx), anew(ngb,nx),pmat(nmat) ,pmat_q(nmat)
01564 c$$$      anew=0d0
01565 c$$$      do ix=1,nmat
```

```
01566 c$$$          anew(in(ix), :)
01567 c$$$     &  = anew(in(ix), :)   +  pmat(ix) * aold(io(ix), :)
01568 c$$$      enddo
01569 c$$$      do ix=1,nmat_q
01570 c$$$          anew(in_q(ix), :)
01571 c$$$     &  = anew(in_q(ix), :)  + dconjg(pmat_q(ix)) * aold(io_q(ix), :)
01572 c$$$      enddo
01573 c$$$      end
01574
01575       real(8) function wcutef(e,ecut,ecuts)
01576       real(8):: e,ecut,ecuts
01577 c      wcutef = 1d0/( exp((e-ecut)/ecuts)+ 1d0)
01578       wcutef = exp( -(e/ecut)**2 ) ! ecuts is not used in this case
01579       end
```

4.29 main/hbasfp0.m.F File Reference

Functions/Subroutines

- program hbasfp0_v2

4.29.1 Function/Subroutine Documentation

4.29.1.1 program hbasfp0_v2 ( )

Definition at line 1 of file hbasfp0.m.F.

4.30 hbasfp0.m.F

```
00001       program hbasfp0_v2
00002 c-- Generates orthonormal optimal product basis and required radial integrals in each MT.
00003 c input files
00004 c  GWinput : input data for GW
00005 c  LMTO   : fundamental data for crystal
00006 c  PHICV  : radial functions Valence and Core
00007 c
00008 c output files
00009 c  BASFP//ibas :: product basis for each ibas
00010 c  PPBRD_V2_//ibas :: radial <ppb> integrals. Note indexing of ppbrd
00011 c
00012 c The main part of this routine is in the subroutine basnfp_v2
00013       use m_rgwinf_v3,only:rgwinf_v3,
00014      & alat,nclass,natom,nspin,nl,nnv,nnc,nrx, cutbase,lcutmx,nindxc,
00015      & nindxv,occv,unoccv,occc,unoccc,iclass
00016       use keyvalue,only: getkeyvalue
00017       use m_anf,only: ibasf,laf,anfcond !may2015takao
00018       implicit none
00019       real(8):: qbas(3,3),ginv(3,3)
00020       integer(4)::
00021      l ifphiv(2),ifphic(2), iphiv(2),iphivd(2),iphic(2),iphi(2),iphidot(2),
00022      l ifev(2),ifevf(2),ibas,ibas1,ic,icx,ifaln,ifinin,iflmto,ifphi,
00023      l ii,ir,irad,isp,ix,lmx,lmx2,n,nbas,ncoremx,l,nn,icore,ifianf,nphi,nradmx,nsp,iopen,maxnn,iclose
00024       integer(4),allocatable:: lcutmxa(:)
00025       character(12) :: aaa
00026       integer(4),allocatable::nrofi(:), nocc(:,:),nunocc(:,:),nindx(:,:)
00027       logical :: ptest=.false. !See ptest in hvccfp0.f
00028       real(8),allocatable :: bb(:),zz(:), phic(:,:)
00029       integer(4) :: ndat
00030       integer(4),allocatable:: ncindx(:,:),lcindx(:,:),
00031      &           nrad(:), nindx_r(:,:), lindx_r(:,:),
00032      &           nc_max(:,:),ncore(:)
00033       real(8),allocatable:: phitoto(:,:,:,:,:), aa(:),rr(:,:),phitotr(:,:,:,:,:)
00034       character*11 :: ffaln
00035       integer(4)::incwfin,ret
00036       integer(4),allocatable:: idid(:)
00037       logical :: checkdid ,anfexist
00038       integer(4):: iread, idummy
00039 c--------------------------------------------------------------------------
00040       ifinin=-99999 !dummy
00041       write(6,'(a)') ' --- Input normal(=0); coremode(=3);'//
00042      & ' ptest(=4); Excore(=5); for core-valence Ex(=6);'//
00043      & ' val-val Ex(7);  normal+<rho_spin|B> (8); version(-9999) ?'
00044       call readin5(ix,iread,idummy)
```

```
00045       call headver('hbasfp0',ix)
00046       if(ix==3) then
00047         write(6,*)'  ### coremode; Product basis for SEXcore ### '
00048         incwfin = -2
00049       elseif(ix==0) then
00050         write(6,*)'  ### usual mode use occ and unocc for core ### '
00051         incwfin = 0
00052       elseif(ix==4) then
00053         write(6,*)
00054      & ' ### ptest mode. now special for Q0P. GWIN_V2 is neglected ### '
00055         write(6,*) '  See basnfp.f of ptest section.'
00056         incwfin = 0
00057       elseif(ix==5) then
00058         write(6,*)
00059      &  '  ### calculate core exchange energy ### ix==5'
00060         incwfin = 0
00061       elseif(ix==6) then
00062         write(6,*)
00063      &  '  ### calculate p-basis for core-valence Ex ix==6'
00064         write(6,*) ' occ=1:unocc=0 for all core'
00065         incwfin = -3
00066       elseif(ix==7) then
00067         write(6,*)
00068      &  '  ### calculate p-basis for val-val Ex ix==7'
00069         write(6,*) ' occ=0:unocc=0 for all core'
00070         incwfin = -4
00071       elseif(ix==8) then !May2005
00072         write(6,"('  ### usual mode use occ and unocc for core',
00073      &            ' and <rho_spin |B(I)> ### ')")
00074         incwfin = 0
00075       else
00076         write(6,*)' hbasfp: input is out of range'
00077         call rx( ' hbasfp: input is out of range')
00078       endif
00079
00080 !! read data in m_rgwinf_v3
00081 !! Output are allocated and data are setted as above.
00082       iflmto = iopen('LMTO',1,0,0)
00083       if (iflmto <= 0) call rx( 'unit file for LMTO <= 0')
00084       call rgwinf_v3(iflmto,ifinin,incwfin) ! readin inputs. See use use m_rgwinf_v3,only: ... at the
     begining.
00085       iflmto= iclose('LMTO')
00086       nsp=nspin
00087       write(6,*)'end of rgwinf'
00088 !! readin lcutmxa ------------
00089       call getkeyvalue("GWinput","<PRODUCT_BASIS>",unit=ifinin,status=ret)
00090       allocate(lcutmxa(1:natom))
00091       do
00092         read(ifinin,*,err=980) aaa
00093         if(aaa=='lcutmx(atom)') then
00094           read(ifinin,*) lcutmxa(1:natom)
00095 c          write(6, '(" lcutmxa=",20i3)' ) lcutmxa(1:natom)
00096           goto 990
00097         endif
00098       enddo
00099   980 continue
00100       lcutmxa=lcutmx
00101   990 continue
00102       close(ifinin)
00103
00104       if(ix==8) then
00105          write(6,*)' Enfoece lcutmx=0 for all atoms'
00106          lcutmxa=0
00107       endif
00108
00109       write(6,"(' lcutmxa=',$)")
00110       write(6,'(20i3)') lcutmxa(1:natom)
00111       lmx        = 2*(nl-1)
00112       lmx2       = (lmx+1)**2
00113       nn         = maxnn(nindxv,nindxc,nl,nclass)
00114       nphi       = nrx*nl*nn*nclass
00115
00116
00117 c -optimal orthonormal product basis
00118 c> reindex nocc,nunocc,nindx
00119 ! For valence  from GWIN_V2
00120 ! occv   : occ    switch
00121 ! unoccv : unocc  switch
00122 ! nindexv: n index
00123 !-----------------------------
00124 ! For core  from GWIN_V2
00125 ! occc   : occ  switch
00126 ! unoccc : unocc switch
00127 ! nindexc: n index
00128 !-----------------------------
00129 ! For valence+core
00130 ! nocc
```

```
00131 ! nunocc
00132 ! nindx
00133       allocate( nocc(nl*nn,nclass), nunocc(nl*nn,nclass), nindx(nl,nclass) )
00134       call reindx(occv,unoccv,nindxv,   occc,unoccc,nindxc,
00135      d              nl,nn,nnv,nnc,nclass,
00136      o              nocc,nunocc,nindx)
00137       write(6,*)' --- end of reindx ---'
00138
00139 c-----------
00140 c read PHIVC  and reserve it to phitot
00141 c----------
00142       ifphi  = iopen('PHIVC', 0,-1,0)     ! PHIV+PHIC augmentation wave and core
00143       read(ifphi) nbas, nradmx, ncoremx
00144       allocate(  ncindx(ncoremx,nbas),
00145      &           lcindx(ncoremx,nbas),
00146      &           nrad(nbas),
00147      &           nindx_r(1:nradmx,1:nbas),
00148      &           lindx_r(1:nradmx,1:nbas),
00149      &        aa(nbas),bb(nbas),zz(nbas), rr(nrx,nbas), nrofi(nbas) ,
00150      &        phitoto(nrx,0:nl-1,nn,nbas,nsp),
00151      &        phitotr(nrx,0:nl-1,nn,nbas,nsp),
00152      &        nc_max(0:nl-1,nbas),ncore(nbas) )
00153       read(ifphi) nrad(1:nbas)
00154       read(ifphi) nindx_r(1:nradmx,1:nbas),lindx_r(1:nradmx,1:nbas)
00155       nc_max=0
00156       do ibas=1,nbas
00157         write(6,*)' --- read PHIVC of ibas=',ibas
00158         ic = ibas
00159         read(ifphi) ncore(ic), ncoremx                              !core
00160         read(ifphi) ncindx(1:ncoremx,ibas),lcindx(1:ncoremx,ibas) !core
00161         read(ifphi) icx,zz(ic),nrofi(ic),aa(ic),bb(ic)
00162         if(ic/=icx) then
00163           write(6,*) 'ic icx=',ic,icx
00164           call rx( 'hbasfp0: ic/=icx')
00165         endif
00166         read(ifphi) rr(1:nrofi(ic),ic)
00167         do isp = 1, nsp
00168           write(6,*)'          ---  isp nrad ncore(ic)=',isp, nrad(ic),ncore(ic)
00169           do icore = 1, ncore(ic)
00170             l =  lcindx(icore,ic)
00171             n =  ncindx(icore,ic)
00172             read(ifphi) phitoto(1:nrofi(ic),l,n, ic,isp)   !core orthogonal
00173             phitotr(1:nrofi(ic),l,n, ic,isp)=              !core raw= core orthgonal
00174      &      phitoto(1:nrofi(ic),l,n, ic,isp)               !
00175             if(n>nc_max(l,ic)) nc_max(l,ic)=n
00176           enddo
00177           do irad = 1, nrad(ic)
00178             l = lindx_r(irad,ic)
00179             n = nindx_r(irad,ic) + nc_max(l,ic)
00180             read(ifphi) phitoto(1:nrofi(ic),l,n, ic,isp) !valence orthogonal
00181             read(ifphi) phitotr(1:nrofi(ic),l,n, ic,isp) !valence raw
00182           enddo
00183         enddo
00184       enddo
00185 c-----------
00186
00187 !! check write
00188       ffaln ='PHIV.chk'
00189       ifaln = iopen(ffaln,1,-1,0)
00190       do ibas = 1,nbas
00191         ic = ibas
00192         do irad = 1, nrad(ic)
00193           l = lindx_r(irad,ic)
00194           n = nindx_r(irad,ic) + nc_max(l,ic)
00195           write(ifaln,"(a,5i5)")'------- ibas l n =',ibas,l,n
00196           do ir=1,nrofi(ic)
00197             write(ifaln,"(3d24.15)")rr(ir,ic), phitotr(ir,l,n,ic,1:nsp)
00198           enddo
00199         enddo
00200       enddo
00201       ifaln = iclose(ffaln)
00202
00203 !!  excore mode ---------
00204       if(ix==5 ) then
00205         call excore(nrx,nl,nnc,nclass,nsp,natom,
00206      &   phitotr(1:nrx,0:nl-1,1:nnc,1:nclass,1:nsp),   !core
00207      &   nindxc,iclass,
00208      &    aa,bb,nrofi,rr)
00209         goto 998
00210       endif
00211
00212 !! antiferro or not.
00213 !! For AF case, we have laf=.true. and we have data set for 'call anfsig', stored in m_anf.
00214       call anfcond()
00215       if(laf) then
00216 !!      Check iclass =ibas ; CLASS file contains true classs information.
00217 c        allocate(idid(natom))
```

```
00218         write(6,*) '--- Antiferro mode --- '
00219         do ibas=1,natom
00220           if(iclass(ibas)/=ibas) call rx( ' iclass(ibas)/=ibas: ')
00221         enddo
00222         ii=0
00223         do ic=1,nclass
00224           ibas=ic
00225           if( ibasf(ibas)>0 ) then
00226             phitotr(:,:,:,ibasf(ibas), :)=phitotr(:,:,:,ibas, :)
00227             write(6,"(a,2i4)")
00228      &       '  radial functions: phi(ibasf)=phi(ibas): ibasf ibas=',ibasf(ibas),ibas
00229           endif
00230         enddo
00231 c        if( sum (idid(1:ii)) /= natom*(natom+1)/2)
00232 c     &    call rx( 'hbasfp0:sum (idid(1:ii)) /= n(n+1)/2')
00233 c        write(6,*)' end of anf section...'
00234       endif
00235
00236 !! override cutbase to make epsPP_lmfh safer. may2013takao
00237       if(ix==4) then
00238          write(6,*)' !!! set tolerance for PB to be 1d-6 ---'
00239          cutbase=1d-6
00240       endif
00241
00242       do ic = 1,nclass
00243         call basnfp_v2(nocc(1,ic),nunocc(1,ic),nindx(1,ic), ! Product Basis functions
00244      &    nl,nn,nrx, nrofi(ic),rr(1,ic),aa(ic),bb(ic),ic,
00245      &    phitoto,phitotr,nsp,nclass,
00246      i    cutbase, lcutmxa(ic),ix,iread,alat
00247      i  ,nc_max(0,ic) )
00248       end do
00249       if(ix==0) call rx0( ' OK! hbasfp0 ix=0 normal mode ')
00250       if(ix==3) call rx0( ' OK! hbasfp0 ix=3 core mode ')
00251       if(ix==4) call rx0( ' OK! hbasfp0 ix=4 ptest mode  ')
00252       if(ix==6) call rx0( ' OK! hbasfp0 ix=6 Exx core-val mode  ')
00253       if(ix==7) call rx0( ' OK! hbasfp0 ix=7 Exx val-val mode  ')
00254       if(ix==8) call rx0( ' OK! hbasfp0 ix=8 normal(ix==0) + <B|spin den>. Enforce lcutmx=0.')
00255  998  if(ix==5) call rx0( ' OK! hbasfp0 ix=5 ex core mode  ')
00256       end
00257
00258
00259 c      logical function checkdid (idid,ii, ibas)
00260 c      integer(4):: idid(ii),ix
00261 c      checkdid=.true.
00262 c      do ix=1,ii
00263 c        if(idid(ix)==ibas) return
00264 c      enddo
00265 c      checkdid=.false.
00266 c      end
00267
00268
00269
00270
00271
00272
00273
00274
```

4.31 main/hsfp0.sc.m.F File Reference

Functions/Subroutines

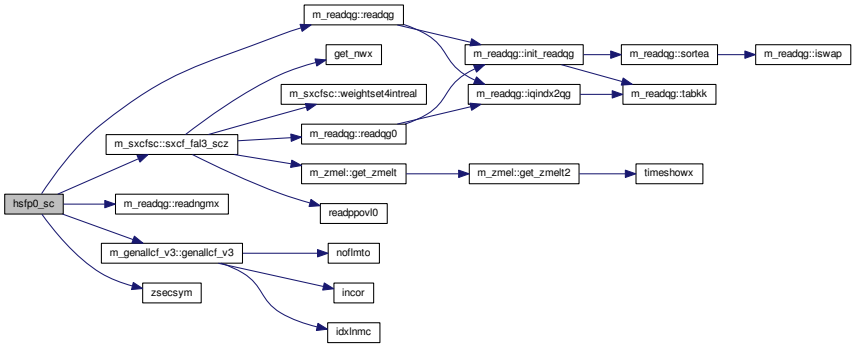
- program hsfp0_sc
- subroutine zsecsym (zsec, ntq, nq, nband, nbandmx, nspinmx, eibzsym, ngrp, tiii, q, is)

4.31.1 Function/Subroutine Documentation

4.31.1.1 program hsfp0_sc ( )

Definition at line 1 of file hsfp0.sc.m.F.

Here is the call graph for this function:



4.31.1.2 subroutine zsecsym (complex(8), dimension(ntq,ntq,nq), intent(inout) *zsec,* integer, intent(in) *ntq,* integer, intent(in) *nq,* integer, intent(in) *nband,* integer, dimension(nq,nspinmx), intent(in) *nbandmx,* integer, intent(in) *nspinmx,* integer, dimension(ngrp,-1:1,nq), intent(in) *eibzsym,* integer, intent(in) *ngrp,* logical, intent(in) *tiii,* real(8), dimension(3,nq), intent(in) *q,* integer, intent(in) *is*)

Definition at line 1219 of file hsfp0.sc.m.F.

Here is the caller graph for this function:

4.32 hsfp0.sc.m.F

```
00001       program hsfp0_sc
00002 !> Calculates the self-energy \Sigma in GW approximation,
00003 !!  including Off-diagonal components.
00004 !!  (hsfp0.F is for diagonal part only).
00005 !! ---------------------------------------
00006 !!    SEx(q,itp,itpp) = <psi(q,itp) |SEx| psi(q,itpp)>
00007 !!    SEc(q,itp,itpp) = <psi(q,itp) |SEc| psi(q,itpp)>
00008 !!    Here SEc(r,r';w) = (i/2pi) < [w'=-inf,inf] G(r,r';w+w') Wc(r,r';w') >
00009 !!
00010 !! ---------------------------------------
00011 !! See papers;
00012 !! [1]T. Kotani and M. van Schilfgaarde, Quasiparticle self-consistent GW method:
00013 !!     A basis for the independent-particle approximation, Phys. Rev. B, vol. 76, no. 16,
00014 !!     p. 165106[24pages], Oct. 2007.
00015 !! [2]T. Kotani, Quasiparticle Self-Consistent GW Method Based on the Augmented Plane-Wave
00016 !!     and Muffin-Tin Orbital Method, J. Phys. Soc. Jpn., vol. 83, no. 9, p. 094711 [11 Pages], Sep. 2014.
00017 !!
00018 !! EIBZ symmetrization;
00019 !! See [3] C. Friedrich, S. Bl?gel, and A. Schindlmayr,
00020 !!   Efficient implementation of the GW approximation within the all-electron FLAPW method,
00021 !!   Physical Review B, vol. 81, no. 12, Mar. 2010.
00022 !!
00023 !! Usage: This routine is called from a script for QSGW, ecalj/fpgw/exec/gwsc.
00024 !! which calls is as "echo 2|../exec/hsfp0_sc >lsc" when mode=2 (three times in the gwsc).
00025 !!
```

```
00026 !! mode= 1: exchange    mode SEx, the exchange part of the self-energy
00027 !! mode= 2: correlation mode SEc, the correlated part of the self-energy
00028 !! mode= 3: core exchange mode SEXcore
00029 !! xxx mode= 4: plot spectrum function ---See manual ---> this is performed by echo 4|hsfp0
00030 !!
00031 !! iSigMode parameter which determines approximation for self-energy is given by GWinput file as iSigMode.
00032 !!    iSigMode==0 SE_nn'(ef)+image integr:delta_nn'(SE_nn(e_n)-SE_nn(ef))
00033 !!    iSigMode==1 SE_nn'(ef)+delta_nn'(SE_nn(e_n)-SE_nn(ef))
00034 !!       xxx not support this mode now ... iSigMode==2 SE_nn'((e_n+e_n')/2)
00035 !!    iSigMode==3 (SE_nn'(e_n)+SE_nn'(e_n'))/2  <--- this is mainly used
00036 !!     iSigMode==5 delta_nn' SE_nn(e_n)
00037 !!     Output file contain hermitean part of SE for energies to be real
00038 !!    (for example, hermitean conjunction of SE_nn'(e_n) means SE_n'n(e_n')^* )
00039 !!
00040 !!     History: We learned so much from LMTO-ASA codeds developed by F.Aryasetiawan.
00041 !! ---------------------------------------
00042       use m_readqg,only: readqg,readngmx
00043       use m_readeigen,only: init_readeigen,init_readeigen2,readeval,lowesteval
00044       use m_read_bzdata,only: nqbz,nqibz,nqbzw,nteti,ntetf
00045      & ,n1,n2,n3,qbas,ginv,qbasmc,qbz,wbz,qibz,wibz,qbzw,idtetf,ib1bz,idteti
00046      & ,nstar,irk,nstbz,ngrp2=>ngrp,qibz_r,nqibz_r,  read_bzdata
00047       use m_genallcf_v3,only: genallcf_v3,
00048      & nclass,natom,nspin,nl,nn, ngrp,
00049      & nlmto,nlnmx, nctot,niw,nw_input=>nw,
00050      & alat,ef, diw,dw,delta,deltaw,esmr,symgrp,clabl,iclass,
00051      & invg, il,in,im,nlnm,
00052      & plat, pos,z,ecore, symgg, konf,nlnx, iantiferro
00053       use keyvalue,only: getkeyvalue
00054
00055 !! Base data to generate matrix elements zmel*. Used in "call get_zmelt".
00056       use m_rdpp,only: rdpp,    !"call rdpp" generate following data.
00057      & nblocha,lx,nx,ppbrd,mdimx,nbloch,cgr
00058 !! Generate matrix element for "call get_zmelt".
00059       use m_zmel,only:  ! folloiwng data set are stored in this module in the main routin,
00060                         ! and used when call get_zmelt, get_zmelt2.
00061      & nband,itq,ngcmx,ngpmx,
00062      & miat,tiat,shtvg, ntq, ppbir
00063 !! antiferro condition. only laf is used, after 'call anfcond()'
00064       use m_anf,only: anfcond,
00065      &   laf
00066 !! subroutine only
00067       use m_sxcfsc,only: sxcf_fal3_scz
00068 !! MPI
00069       use m_mpi,only:
00070      &   mpi__initialize,mpi__real8send,mpi__real8recv,mpi__send_iv,mpi__recv_iv,mpi__sxcf_rankdivider,
00071      &   mpi__finalize,mpi__root,mpi__broadcast,mpi__rank,mpi__size,mpi__allreducesum,
00072      &   mpi__consoleout,
00073      &   mpi__barrier
00074
00075       implicit none
00076       integer:: nw
00077 !! ------------------------------------------------------------------------------------
00078 !!     real(8),parameter :: ua  = 1d0 ! constant in w(0)exp(-ua^2*w'^2) to take care of peak around w'=0
00079 c-----------------------------------
00080 !!!   test switches to calculate the self-energy based on an another separation of \Sigma.
00081 !!!     \Sigma = \Sigma_{sx} + \Sigma_{coh} + \Sigma_{img axis} + \Sigma_{pole} by Hedin PR(1965)A785
00082 !!!     I found COH term has inevitably poor accuracy.
00083       logical ::tetra, tetra_hsfp0,
00084      & screen = .false.,        ! \Sigma_{sx} for mode 1 and
00085 ! \Sigma_{img axis} + \Sigma_{pole} for mode 2
00086      & cohtest= .false.         ! \Sigma_{coh}. mode swich is not required.
00087 c     &  , tetra  = .false.  ! test switch for tetrahedron method test.
00088 c     ! tetra=T is only effective for exchange=T case.
00089 c     ! Tetrahedron mehod for correlation is a bit
00090 ! difficult and I gave up for a while.
00091 ! If you want to calculate with tetra=T for exchange, you
00092 ! have to uncomment tetra related part in
00093 ! sxcf.f, and a part calling sxcf in this routine. Note wtet wtetef!
00094 ! They sometimes cause array destruction if you run tetra=T without comment them.
00095
00096 c      real(8) :: shtw
00097       integer::
00098      & ixc,iopen,ifhbed, nprecb,mrecb,mrece,nlmtot,nqbzt, !nband,
00099      & ibas,ibasx,nxx,ifqpnt,ifwd,
00100      & nprecx,mrecl,nblochpmx2,nwp,niwt, nqnum,nblochpmx, !mdimx,nbloch
00101      & noccxv,maxocc,noccx,ifvcfpout,iqall,iaf, !ntq, !ifrcw,ifrcwi,
00102      & i,k,nspinmx, nq,is,ip,iq,idxk,ifoutsex,iclose,nq0i,ig,
00103      & mxkp,nqibzxx,ntet,nene,iqi, ix,iw,
00104      & nlnx4,invr,ivsum, ifoutsec, !niwx,
00105      & ifsec(2)
00106      & ,ifxc(2),ifsex(2), ifphiv(2),ifphic(2),ifec,ifexsp(2),
00107      & ifsex2(2),ifsec2(2),     !out S_nn'
00108      & ifsecomg(2),ndble=8
00109       real(8) :: pi,tpia,vol,voltot,rs,alpha,
00110      & qfermi,efx,valn,efnew,edummy,efz,qm,xsex,egex,edummyd(1),
00111      & zfac1,zfac2,dscdw1,dscdw2,dscdw,zfac
00112       logical :: lqall,laff,lntq
```

```
00113       real(8),allocatable    :: q(:,:)
00114
00115       integer,allocatable ::
00116      & ngvecp(:,:), ngvecc(:,:),iqib(:),
00117      & kount(:,:)
00118       real(8),allocatable:: vxcfp(:,:,:),
00119      & wqt(:),q0i(:,:),
00120      & eqt(:),
00121      & ppbrdx(:,:,:,:,:,:),
00122      & eq(:),
00123      & eqx(:,:,:),eqx0(:,:,:),ekc(:),coh(:,:)
00124       complex(8),allocatable:: zsec(:,:,:)
00125 c
00126       logical :: legas
00127       real(8) :: rydberg,hartree
00128       real(8):: qreal(3), ntot,nocctotg2,tripl,xxx(3,3)
00129       logical ::nocore
00130
00131 c     space group information
00132       integer,allocatable :: iclasst(:), invgx(:)
00133 c     tetra
00134       real(8),allocatable :: qz(:,:),qbzxx(:),wbzxx(:),wtet(:,:,:,:),
00135      & eband(:,:,:), ene(:)
00136       integer,allocatable ::idtetx(:,:),idtet(:,:),ipq(:)
00137      & ,iene(:,:,:),ibzx(:)
00138       integer ::ib,iqx,igp,iii,ivsumxxx,isx,iflegas, iqpntnum
00139 c
00140       real(8),allocatable   :: eex1(:,:,:),exsp1(:,:,:),qqex1(:,:,:,:)
00141       integer,allocatable:: nspex(:,:),ieord(:),itex1(:,:,:)
00142       real(8)    :: qqex(1:3), eex,exsp,eee, exwgt,deltax0
00143       integer :: itmx,ipex,itpex,itex,nspexmx,nnex,isig,iex,ifexspx
00144      & ,ifexspxx ,ifefsm, nq0ix,ifemesh,nz
00145       character(3)  :: charnum3
00146       character(12) :: filenameex
00147       logical :: exspwrite=.false.
00148       character*8 xt
00149
00150       integer :: isigmode,ifinin ,idummy
00151
00152       real(8),allocatable:: omega(:)
00153       real(8)   :: ebmx(2)
00154       integer:: nbmx(2)
00155
00156       real(8):: volwgt
00157
00158       integer::nwin, incwfin
00159       real(8)::efin
00160       real(8),allocatable::freqx(:),freqw(:),wwx(:)
00161
00162       integer::  ngpn1,mrecg,ngcn1
00163       real(8)   :: wgtq0p,quu(3)
00164
00165       character(2):: soflag
00166       integer:: ifianf
00167
00168       integer:: ifpomat,nkpo,nnmx,nomx,ikpo,no
00169       real(8):: q_r(3)
00170       real(8),allocatable:: qrr(:,:)
00171       integer,allocatable:: nnr(:),nor(:)
00172
00173       logical :: allq0i
00174       integer:: nw_i
00175       logical:: exonly
00176       real(8):: wex
00177 !! newaniso mode
00178 c      logical:: newaniso
00179       real(8),allocatable:: vcousq(:),dmlx(:,:),epinvq0i(:,:),wklm(:),vcoud(:)
00180       complex(8),allocatable:: zcousq(:,:)
00181       integer:: ifvcoud,lxklm,ifidmlx
00182
00183       integer,allocatable:: irkip_all(:,:,:,:),irkip(:,:,:,:)
00184
00185       integer,allocatable:: nrkip_all(:,:,:,:),nrkip(:,:,:,:)
00186       integer,allocatable:: neibz(:),nwgt(:,:),ngrpt(:),igx(:,:,:),igxt(:,:,:),eibzsym(:,:,:)
00187       integer:: iqxend,iqxini
00188       integer:: l2nl,igrp,kx,kr
00189       logical :: iprintx,tiii,timereversal, eibz4sig,tiiiout
00190
00191       logical :: selectqp=.false.,diagonly=.false.
00192       integer:: ret,dest,nnn
00193       character(128) :: ixcc
00194       real(8):: eftrue,esmref   !jan2013
00195       real(4):: time_red1,time_red2
00196       integer:: timevalues(8) ,ibz
00197
00198       integer::irot !,nn_
00199       real(8),allocatable:: wgt0(:,:)
```

```
00200       logical:: exchange
00201       real(8):: exx
00202       real(8),allocatable:: freq_r(:)
00203       integer:: ififr,ifile_handle,nwxx,ifih
00204
00205       integer:: verbose,iband,isp,iqq
00206       integer,allocatable:: nbandmx(:,:)
00207
00208       integer:: ificlass,ifiq0p,ntqxx,nq_r,nband_r
00209       logical:: hermitianw
00210 c--------------------------------------
00211       call mpi__initialize()    ! MIZUHO-IR
00212       call date_and_time(values=timevalues)
00213       write(6,'(a,9i5)')'dateandtime1=',mpi__rank,timevalues(1:8)
00214 !TIME0_0000
00215 !TIME0_0010
00216       hartree=2d0*rydberg()
00217       hermitianw=.true.
00218       if(cohtest) then          !currently not used (may need fixing if necessary)
00219         screen = .true.
00220         ixc = 2; nz=0
00221         open(671,file='COH')
00222       elseif(mpi__root) then
00223         write(6,*) ' --- Choose modes below ------------'
00224         write(6,*) '  Sx(1) Sc(2) ScoreX(3) '
00225         write(6,*) '  [option --- (+ QPNT.{number} ?)] '
00226         write(6,*) ' Add 1000, eg, 1001 is diagonal only mode for one-shot Z=1'
00227         write(6,*) ' --- Put number above ! ------------'
00228         call readin5(ixc,nz,idummy)
00229         write(6,*) ixc
00230       endif
00231       call mpi__broadcast(ixc)
00232       call mpi__broadcast(nz)
00233       if(mpi__root) call headver('hsfp0_sc',ixc)
00234       write(ixcc,"('.mode=',i4.4)")ixc
00235
00236       if(ixc>1000) then         !selected QP
00237         ixc=mod(ixc,1000)
00238         selectqp=.true.
00239         diagonly=.true.
00240         hermitianw=.false.
00241         write(6,*) "--- Diagonal-only mode. jobsw=5; see description at the top of sxcf_fal2.sc.F."
00242         write(6,*) "--- This is the same as one-shot calculaiton with iSigMode5 in GWinput."
00243       endif
00244
00245       call mpi__consoleout('hsfp0_sc'//trim(ixcc))
00246       write(6,*) ' ixc nz=',ixc, nz
00247       if(ixc==0) call rx( ' --- ixc=0 --- Choose computational mode!')
00248
00249 !! ===  readin BZDATA. See gwsrc/rwbzdata.f ===
00250 !! See use m_read_bzdata,only: at the top of this routine
00251       call read_bzdata()
00252       write(6,*)' nqbz =',nqbz
00253       write(6,*)' nqibz ngrp=',nqibz,ngrp2
00254       call pshprt(60)
00255
00256 !! === readin GWIN and LMTO, then allocate and set datas. ===
00257 !! See use m_genallcf_v3,only: at the top of this routine
00258       nwin   = 0                !Readin nw from NW file
00259       efin=-999d0               !not readin EFERMI
00260       if(ixc==3) then; incwfin= -2 !core exchange mode
00261       else           ; incwfin= -1 !use 7th colmn for core at the end section of GWIN
00262       endif
00263       call genallcf_v3(nwin,efin,incwfin) ! module m_genallcf_v3. See use m_genallcf in this
     routine
00264       if(ngrp/= ngrp2) call rx( 'ngrp inconsistent: BZDATA and LMTO GWIN_V2')
00265       esmref=esmr
00266
00267 !! iSigMode
00268       call readd_isigma_en(ifinin,isigmode) !reading self-energy mode parameter from file 'GWinput'
00269       if(diagonly) isigmode=5
00270
00271 !! Get maximums
00272       call getnemx8(nbmx,ebmx)  !Get maximums takao 18June03
00273 !!     nbmx1 ebmx1: to set how many bands of <i|sigma|j>  do you calculate.
00274 !!     nbmx2 ebmx2: to restrict num of bands of G to calculate G \times W
00275 !! ebmx2 nbmx2 are not used. For safe, strange number is supplied here.
00276       nbmx(2)=9999999
00277       ebmx(2)=1d10
00278       write(6,"('  nbmx ebmx from GWinput=',i8,d13.5)") nbmx(1),ebmx(1)
00279
00280 !!Caution!  WE ASSUME iclass(iatom)= iatom (because of historical reason)
00281       if (nclass /= natom ) call rx( ' hsfp0: nclass /= natom ')
00282       write(6,*)' hsfp0_sc: end of genallcf_v3'
00283       call pshprt(30)
00284       pi   = 4d0*datan(1d0)
00285       tpia = 2d0*pi/alat
```

```
00286       call dinv33(plat,1,xxx,vol)
00287       voltot = dabs(vol)*(alat**3)
00288 c      shtw = 0d0
00289       tetra= tetra_hsfp0()
00290 !! if(esmr<1d-5) shtw=0.01d0 ! Ferdi's shift to avoid resonance effect(maybe), I used this until sep2012
00291
00292 c$$$!! ef is taken as rs for the empty-sphere test case of legas=T case
00293 c$$$!! HOMOGENIOUS GAS code. Usually not used. Need fixing if necessary.
00294 c$$$!! Keep this just as a memo.
00295 c$$$      legas = .false.
00296 c$$$      if(.false.) then
00297 c$$$        INQUIRE (FILE = 'LEGAS', EXIST = legas)
00298 c$$$        if(legas) then          !!! test for electron gas case.
00299 c$$$          write(6,*)' find LEGAS. legas =',legas
00300 c$$$          iflegas = 2101
00301 c$$$          open (iflegas,file='LEGAS')
00302 c$$$          read(iflegas,*)rs
00303 c$$$          close(iflegas)
00304 c$$$          alpha = (9*pi/4d0)**(1d0/3d0)
00305 c$$$          qfermi = alpha/rs
00306 c$$$          efx  = qfermi**2
00307 c$$$          valn = efx**1.5d0*voltot/3d0/pi**2
00308 c$$$          write (6,*)'  #### egas test mode  legas=T #### given rs =',rs
00309 c$$$          write (6,*)' egas  Exact Fermi momentum  qf  =', qfermi
00310 c$$$          write (6,*)' egas  Exact Fermi energy    Ef  =', efx
00311 c$$$          if(tetra) call rx( 'legas You have to give ef of  tetrahedron')
00312 c$$$        endif
00313 c$$$      endif
00314 c$$$!!
00315       if(ixc==1) then
00316         exchange=.true.
00317         write(6,*) ' --- Exchange mode --- '
00318         if(mpi__root) then
00319           ifxc(1)  = iopen('XCU'//xt(nz),1,-1,0)
00320           ifsex(1) = iopen('SEXU'//xt(nz),1,-1,0)
00321           ifsex2(1)= iopen('SEX2U',0,-1,0) !out SEX_nn'
00322           if (nspin == 2) then
00323             ifxc(2)  = iopen('XCD'//xt(nz),1,-1,0)
00324             ifsex(2) = iopen('SEXD'//xt(nz),1,-1,0)
00325             ifsex2(2)= iopen('SEX2D',0,-1,0) !out SEX_nn'
00326           endif
00327         endif
00328 c         INQUIRE (FILE = 'EXspTEST', EXIST = exspwrite)
00329 c         if(exspwrite) then
00330 c            write(6,*)'--- Find EXspTEST ExspectrumWrite=',exspwrite
00331 c            write(6,*)'--- esmr is chosen to be 2d0 Ry'
00332 c            esmr= 2d0
00333 c            do is=1,nspin
00334 c               ifexsp(is)  = iopen('EXSP.'//char(48+is),1,-1,0)
00335 c            enddo
00336 c         endif
00337       elseif(ixc==2) then
00338         exchange=.false.
00339         write(6,*) ' --- Correlation mode --- '
00340         if(cohtest) write(6,*) ' COH calculation mode. Results in COH'
00341         if(mpi__root) then
00342           ifsec(1)   = iopen('SECU'//xt(nz),1,-1,0) ! output files
00343           ifsec2(1)= iopen('SEC2U',0,-1,0) !out SEC_nn'
00344           if (nspin == 2)
00345      .     ifsec(2)   = iopen('SECD'//xt(nz),1,-1,0)
00346           ifsec2(2)= iopen('SEC2D',0,-1,0) !out SEC_nn'
00347         endif
00348       elseif(ixc==3) then
00349         exchange=.true.
00350         esmr=0d0
00351         write(6,*) ' --- CORE Exchange mode --- '
00352         if(mpi__root) then
00353           ifsex(1)   = iopen('SEXcoreU'//xt(nz),1,-1,0)
00354           ifsex2(1)= iopen('SEXcore2U',0,-1,0) !out SEXcore_nn'
00355           if (nspin == 2) then
00356             ifsex(2)   = iopen('SEXcoreD'//xt(nz),1,-1,0)
00357             ifsex2(2)= iopen('SEXcore2D',0,-1,0) !out SEXcore_nn'
00358           endif
00359         endif
00360 !! spectrum funciton mode, we do not use ixc==4
00361 c     elseif(ixc==4) then
00362 c     write(6,*) ' --- Spectrum function Sigma(\omega) mode --- '
00363 c     exchange=.false.
00364 c     ifsecomg(1) = iopen('SEComgU'//xt(nz),1,-1,0) ! output files
00365 c     if (nspin == 2)
00366 c     .  ifsecomg(2) = iopen('SEComgD'//xt(nz),1,-1,0)
00367       else
00368         call rx( ' hsfp0: Need input (std input) 1(Sx) 2(Sc) or 3(ScoreX)!')
00369       endif
00370
00371 c---  Neglect core is NoCore exists ------------
00372 c     inquire(file='NoCore',exist=nocore)
```

```
00373 c     if(nocore) nctot=0
00374
00375       write(6, *) ' --- computational conditions --- '
00376       write(6,'("    deltaw  =",f13.6)') deltaw
00377 c     write(6,'("    ua      =",f13.6)') ua
00378       write(6,'("    esmr    =",f13.6)') esmr
00379       write(6,'("    alat voltot =",2f13.6)') alat, voltot
00380
00381 !! read dimensions of wc,b,hb
00382       ifhbed = ifile_handle()   !  ifhbed = iopen('hbe.d',1,0,0)
00383 ! ifile_handle() search unused file handle
00384       open(ifhbed,file='hbe.d',status='old')
00385       read (ifhbed,*) nprecb,mrecb,mrece,nlmtot,nqbzt, nband,mrecg
00386       close(ifhbed)             !isx = iclose ('hbe.d')
00387       if (nprecb == 4) call rx( 'hsfp0: b,hb in single precision')
00388 !!
00389       call init_readeigen(ginv,nspin,nband,mrece) !initialization of readEigen
00390 ! required for readeigen readchpi readgeig.
00391
00392 !!  === Get space group information ===
00393 !! True class information in order to determine the space group,
00394 !! because the class in the generated GW file is dummy. (iclass(ibas)=ibas should be kept).
00395       ificlass=ifile_handle()
00396       open (ificlass,file='CLASS')
00397       allocate(iclasst(natom),invgx(ngrp)
00398      & ,miat(natom,ngrp),tiat(3,natom,ngrp),shtvg(3,ngrp))
00399       write(6,*)'  --- Readingin CLASS info ---'
00400       do ibas = 1,natom
00401         read(ificlass,*) ibasx, iclasst(ibas)
00402         write(6, "(2i10)") ibasx, iclasst(ibas)
00403       enddo
00404       close(ificlass)
00405 !!  Get space-group transformation information. See header of mptaouof.
00406       call mptauof(symgg,ngrp,plat,natom,pos,iclasst
00407      o ,miat,tiat,invgx,shtvg ) !note: miat,tiat,shtvg are defined in m_zmel.
00408       if(verbose()>=40) write (*,*)' hsfp0.sc.m.F: end of mptauof'
00409
00410 !! ====  Get array size to call rdpp can call rdpp to generate base data for get_zmel ====
00411       call getsrdpp2( nclass,nl,nxx)
00412       call readngmx('QGpsi',ngpmx)
00413       call readngmx('QGcou',ngcmx)
00414       write(6,*)' max number of G for QGpsi and QGcou: ngcmx ngpmx=',ngcmx,ngpmx
00415       allocate(ngvecp(3,ngpmx),ngvecc(3,ngcmx))
00416       call readqg('QGpsi',qibz(1:3,1),ginv, quu,ngpn1, ngvecp)
00417       call readqg('QGcou',qibz(1:3,1),ginv, quu,ngcn1, ngvecc)
00418       deallocate(ngvecp,ngvecc)
00419       write(6,*) ' end of read QGcou'
00420 !!  ppbrd = radial integrals
00421 !!  cgr   = rotated cg coeffecients.
00422       call rdpp(nxx, nl, ngrp, nn, nclass, nspin, symgg,qbas)
00423 !  output: nblocha, lx, nx,  ppbrd , mdimx, nbloch, cgr are stored in m_rdpp.
00424       call pshprt(60)
00425
00426 !! Readin WV.d
00427       if(.not.exchange.or.(exchange.and.screen)) then !screen means screened exchange case
00428         ifwd=ifile_handle()     ! ifwd = iopen('WV.d',1,-1,0)
00429 !direct access files WVR and WVI which include W-V.
00430         open(ifwd,file='WV.d')
00431         read (ifwd,*) nprecx,mrecl,nblochpmx,nwp,niwt, nqnum, nw_i
00432         write(6,"(' Readin WV.d =', 10i8)") nprecx,mrecl,nblochpmx,nwp,niwt, nqnum, nw_i
00433         close(ifwd)             !ifwd =iclose('WV.d')
00434         call checkeq(nprecx,ndble)
00435         nw = nwp-1
00436         if(niwt /= niw) call rx( 'hsfp0_sc: wrong niw')
00437
00438 !! Energy mesh; along real axis. Read 'freq_r'
00439 !! NOTE nw_i=nw for non-timereversal case.
00440 !!      nw_i=0 for time-reversal case.
00441 !!  NOTE: We assume freq_r(i) == -freq_r(-i) in this code. feb2006
00442 !!  NOTE: this program assumes freq_r(iw)=freq_r(-iw). freq_r(iw <0) is redundant.
00443         write(6,'("    niw nw dw   =",2i6,f13.6)') niw,nw,dw
00444         ififr=ifile_handle()
00445         open(unit=ififr,file='freq_r')
00446         read(ififr,*)nwxx
00447         if(nwxx/= nw+1) call rx( ' freq_r nw /=nw')
00448         allocate(freq_r(nw_i:nw)) !freq_r(0)=0d0
00449         do iw= nw_i,nw
00450           read(ififr,*) freq_r(iw)
00451         enddo
00452         close(ififr)
00453         if(nw_i/=0) then
00454           if(nw/= -nw_i)        call rx( "sxcf_fal3_scz: nw/=-nw_i")
00455           if(freq_r(0)/=0d0)    call rx( "sxcf_fal3_scz: freq_r(0)/=0")
00456           if( sum(abs( freq_r(1:nw)+freq_r(-1:-nw:-1)))/=0)
00457      &     call rx( "sxcf_fal3_scz: freq_r /= -freq_r")
00458         endif
00459       endif
```

```
00460
00461       if(tetra) goto 201        !tetra is experimental.  usually =F.
00462
00463 !!== Determine Fermi energy ef for given valn (legas case), or corresponding charge given by z and konf.==
00464 !!    When esmr is negative, esmr is geven automatically by efsimplef.
00465       legas=.false.
00466       call efsimplef2a(nspin,wibz,qibz,ginv,
00467      i nband,nqibz
00468      i ,konf,z,nl,natom,iclass,nclass
00469      i ,valn, legas, esmref,    !!! valn is input for legas=T, output otherwise.
00470      i qbz,nqbz                 ! index_qbz, n_index_qbz,
00471      o ,efnew)
00472       if(ixc/=3) ef = efnew
00473       eftrue = efnew
00474
00475 !! ==== check total ele number =====
00476       ntot  = nocctotg2(nspin, ef,esmr, qbz,wbz, nband,nqbz)
00477       write(6,*)' ef    =',ef
00478       write(6,*)' esmr  =',esmr
00479       write(6,*)' valn  =',valn
00480       write(6,*)' ntot  =',ntot
00481
00482 !! == Core-exchange case. ef means just below the valence eigenvalue (to take only core in sxcf).==
00483       if(ixc==3) then
00484         ef = lowesteval() -1d-3 !lowesteigen(nspin,nband,qbz,nqbz) - 1d-3 !lowesteb was
00485         call getkeyvalue("GWinput","EXonly",wex,default=0d0)
00486         if(wex==0d0) then
00487           exonly=.false.
00488         else
00489           exonly=.true.
00490           write(6,*)' exonly=T ecore shift: ecore---> ecore-100'
00491           ecore = ecore-100.0
00492         endif
00493         if(maxval(ecore(:,1:nspin))>ef) then
00494           write(6,*)' ef nspin=',ef,nspin,nctot
00495           do is=1,nspin
00496             write(6,*)' maxval( ecore) nctot=', is,nctot
00497             do ix=1,nctot
00498               write(6,"(i4,d13.5)") ix,ecore(ix, is)
00499             enddo
00500           enddo
00501           call rx( 'hsfp0 ixc=3: ecore>evalence. ')
00502         endif
00503       endif
00504  201  continue
00505
00506       call init_readeigen2(mrecb,nlmto,mrecg) !initialize m_readeigen
00507
00508 !! Read q-points and states
00509       nspinmx = nspin
00510       if(selectqp .and. mpi__root) then
00511         call getkeyvalue("GWinput","<QPNT>",unit=ifqpnt,status=ret)
00512         lqall      = .false.
00513         laff        = .false.
00514         call readx(ifqpnt,10)
00515         read (ifqpnt,*) iqall,iaf
00516         if (iqall == 1) lqall = .true.
00517         if (iaf   == 1)   laff = .true.
00518         call readx(ifqpnt,100)
00519         if (lqall) then         !all q-points case
00520           nq         = nqibz
00521           allocate(q(3,nq))
00522           call dcopy(3*nqibz,qibz,1,q,1)
00523         else
00524           call readx(ifqpnt,100)
00525           read (ifqpnt,*) nq
00526           allocate(q(3,nq))
00527           do       k = 1,nq
00528             read (ifqpnt,*) i,q(1,k),q(2,k),q(3,k)
00529           enddo
00530         endif
00531         nspinmx = nspin
00532         if (laff) nspinmx =1
00533         close(ifqpnt)
00534       else
00535 !     q-points. bzcase()=1
00536         nq = nqibz
00537         allocate(q(3,nq))
00538         q(:,1:nq) = qibz(:,1:nq) !call dcopy   (3*nqibz,qibz,1,q,1)
00539       endif
00540 !!
00541       call mpi__broadcast(nq)
00542       if(mpi__root) then
00543         do dest=1,mpi__size-1
00544           call mpi__real8send(q,3*nq,dest)
00545         enddo
00546       else
```

```
00547         call mpi__real8recv(q,3*nq,0)
00548       endif
00549 !! antiferro case. Only calculate up spin
00550       call anfcond()
00551       if(laf) nspinmx=1
00552       call mpi__broadcast(nspinmx)
00553 
00554 
00555 !! Determine ntq.  See also in sxcf_fal.sc.F ntq should be common for all ixc modes.
00556 !! FIX NTQ during iteration by the file NTQ 15jun2015
00557 !!
00558 !! Determine nbandmx. Moved from sxcf_fal2.sc.F.
00559 !!!! count number of band to calculate.
00560 !! I think it it better to determine nbandmx in a manner within LDA
00561 !! (need to care degeneracy...).
00562       allocate(nbandmx(nq,nspinmx))
00563       if(mpi__root) then
00564         inquire(file='NTQXX',exist=lntq)
00565 
00566         ifih = ifile_handle()
00567         open(ifih,file='NTQXX')
00568 !!   Get ntq
00569         if(lntq) then
00570           read(ifih,*) nband_r,nq_r,ntq
00571           if(nband_r/=nband.or.nq_r/=nq) then
00572             rewind ifih
00573             lntq=.false.
00574           endif
00575         endif
00576         if(.not.lntq) then
00577           ntq=0
00578           allocate(eqt(nband))
00579           do is = 1,nspin
00580             do ip = 1,nq
00581               call readeval(qibz(1,ip),is, eqt)
00582               do iband=1,nband
00583                 ntq = max(iband,ntq)
00584                 if(eqt(iband)-eftrue>ebmx(1)) exit
00585               enddo
00586             enddo
00587           enddo
00588           ntq = min(ntq, nbmx(1))
00589           deallocate(eqt)
00590           write(ifih,"(3i10)") nband,nq,ntq
00591         endif
00592 !!   Get ntqxx(iq,isp) and nbandmx
00593         allocate(eqt(nband))
00594         do is = 1,nspinmx
00595           do ip = 1,nq
00596             call readeval(qibz(1,ip),is, eqt)
00597             if(lntq) then
00598               read(ifih,*) ntqxx   ! ntqxx = ntq !jun2016
00599             else
00600               ntqxx = 0
00601               do i = 1,ntq
00602                 if(eqt(i)-eftrue<ebmx(1)) ntqxx =ntqxx  + 1
00603               enddo
00604               ntqxx = min(ntqxx, nbmx(1))
00605               write(ifih,"(i10)") ntqxx
00606             endif
00607             if(ntqxx<nband) then ! redudce ntqxx when band tops are degenerated.
00608               do i=ntqxx,1,-1
00609                 if(eqt(i+1)-eqt(i)<1d-2) then !1d-2 is a tol to check degeneracy.
00610                   ntqxx=i-1
00611                 else
00612                   exit
00613                 endif
00614               enddo
00615             endif
00616             nbandmx(ip,is) = ntqxx !number of bands to be calculated
00617           enddo
00618         enddo
00619         deallocate(eqt)
00620         close(ifih)
00621       endif
00622       call mpi__broadcast(ntq)
00623 !!
00624       do is=1,nspinmx
00625         if(mpi__root) then
00626           print *,'is nbandmx(:,is)=',is,nbandmx(:,is)
00627           do dest=1,mpi__size-1
00628             call mpi__send_iv(nbandmx(1:nq,is),dest)
00629           enddo
00630         else
00631           call mpi__recv_iv(nbandmx(1:nq,is),0)
00632         endif
00633       enddo
```

```
00634
00635 !! trivial case of itq itq(i)=i
00636       allocate (itq(ntq))
00637       do i = 1, ntq
00638          itq(i) = i !itq is used also in hsfp0.m.F
00639       enddo
00640       do iq=1,nq
00641          write(6,'(" Target iq q=",i6,3f9.4)')iq,q(:,iq)
00642       enddo
00643
00644 !! read LDA eigenvalues
00645 c      allocate(omega(ntq))
00646       allocate(eqx(ntq,nq,nspin),eqx0(ntq,nq,nspin),eqt(nband))
00647       do is = 1,nspin
00648         do ip = 1,nq
00649           call readeval(q(1,ip),is,eqt)
00650           eqx0(1:ntq,ip,is) = eqt(itq(1:ntq))
00651           eqx(1:ntq,ip,is) = rydberg()*(eqt(itq(1:ntq))- eftrue)
00652         enddo
00653       enddo
00654       deallocate(eqt)
00655
00656       write (6,*)' ***'
00657       write (6,6700) nspin,nq,ntq
00658  6700 format (1x,3i4,'  nspin  nq  ntq')
00659       write (6,6501) is,nbloch,ngpn1,ngcn1,nqbz,nqibz,ef,deltaw,alat,ef,esmr
00660  6501 format (' spin =',i2,'   nbloch ngp ngc=',3i4
00661      & ,'  nqbz =',i6,'  nqibz =',i6,'   ef=', f10.4,' Rydberg'
00662      & ,/,d23.16,' <= deltaw(Hartree)'
00663      & ,/,d23.16,' <= alat'
00664      & ,/,d23.16,' <= ef '
00665      & ,/,d23.16,' <= esmr')
00666 c      call winfo(6,nspin,nq,ntq,is,nbloch,ngpn1,ngcn1,nqbz,nqibz,ef,deltaw,alat,esmr)
00667 !!-------------------------
00668 !!     LDA exchange-correlation
00669 !!-------------------------
00670       if(ixc==1) then
00671         allocate(  vxcfp(ntq,nq,nspin) )
00672         call rsexx(nspin,itq,q,ntq,nq, ginv, vxcfp) !add ginv july2011
00673         if(mpi__root) then
00674           do is = 1,nspinmx
00675             write (ifxc(is),*) '==================================='
00676             write (ifxc(is),"(' LDA exchange-correlation : is=',i3)")is
00677             write (ifxc(is),*) '==================================='
00678             call winfo(ifxc(is),nspin,nq,ntq,is,nbloch
00679      &       ,ngpn1,ngcn1,nqbz,nqibz,ef,deltaw,alat,esmr)
00680             write (ifxc(is),*)' ***'
00681             write (ifxc(is),"(a)") ' jband   iq ispin
00682      &qvec
00683      &eigen-Ef (in eV)
00684      &LDA XC (in eV)'
00685             ifoutsex = ifxc(is)
00686             write(6,*)
00687             do ip = 1,nq
00688               do i  = 1,ntq
00689                 write(ifoutsex,"(3i5,3d24.16,3x,d24.16,3x,d24.16)")
00690      &           itq(i),ip,is, q(1:3,ip), eqx(i,ip,is),
00691      &           vxcfp(i,ip,is)
00692                 if(eqx(i,ip,is) <1d20.and.vxcfp(i,ip,is)/=0d0) then !takao june2009. See lmf2gw
     (evl_d=1d20; in Ry.. but eqx is in eV. no problem for inequality).
00693                   write(6,"(' j iq isp=' i3,i4,i2,'  q=',3f8.4,
00694      &'  eig=',f10.4,'  Sxc(LDA)=',f10.4)")
00695      &             itq(i),ip,is, q(1:3,ip), eqx(i,ip,is),
00696      &             vxcfp(i,ip,is)
00697                 endif
00698               end do
00699             end do
00700             if(is==1) isx = iclose('XCU'//xt(nz))
00701             if(is==2) isx = iclose('XCD'//xt(nz))
00702           enddo                 !     end of spin-loop
00703         endif                   !MPI__root
00704         deallocate(vxcfp)
00705       endif
00706
00707 !!  Offset Gamma point Q0P
00708       write(6,*) 'reading QOP'
00709       ifiq0p=ifile_handle()
00710       open (ifiq0p,file='Q0P')
00711       read (ifiq0p,"(i5)") nq0i
00712       if(.not.exchange) call checkeq(nqibz+nq0i-1, nqnum)
00713       write(6,*) ' *** nqibz nq0i_total=', nqibz,nq0i
00714       allocate( wqt(1:nq0i),q0i(1:3,1:nq0i) )
00715 c     read (101,"(d24.16,3x, 3d24.16)" )( wqt(i),q0i(1:3,i),i=1,nq0i)
00716       nq0ix = nq0i
00717       do i=1,nq0i
00718         read (ifiq0p,* ) wqt(i),q0i(1:3,i)
00719         if(wqt(i)==0d0 ) nq0ix = i-1
```

```
00720       enddo
00721       nq0i = nq0ix              ! New nq0i July 2001
00722       write(6,*) ' Used k number in Q0P =', nq0i
00723       write(6,"(i3,f14.6,2x, 3f14.6)" )(i, wqt(i),q0i(1:3,i),i=1,nq0i)
00724       close(ifiq0p)
00725       allocate( wgt0(nq0i,ngrp) )
00726       call getkeyvalue("GWinput","allq0i",allq0i,default=.false.) !S.F.Jan06
00727       call q0iwgt3(allq0i,symgg,ngrp,wqt,q0i,nq0i, !S.F.Jan06
00728      o wgt0)                    ! added allq0i argument
00729       if (nq0i/=0 ) write(6,*) ' *** tot num of q near 0   =', 1/wgt0(1,1)
00730       write(6,"('  sum(wgt0) from Q0P=',d14.6)")sum(wgt0)
00731 c$$$      if(bzcase()==2) then
00732 c$$$         wgt0= wgt0*wgtq0p()/dble(nqbz)
00733 c$$$         write(6,"('bzcase=2:  sum(wgt0_modified )=',d14.6)")sum(wgt0)
00734 c$$$      endif
00735
00736 !! Pointer to optimal product basis
00737 c      allocate(imdim(natom))
00738 c      call indxmdm (nblocha,nclass,iclass,natom,
00739 c     o imdim )                   !in m_zmel
00740       if(niw/=0) then
00741 !! Generate gaussian frequencies x between (0,1) and w=(1-x)/x
00742         allocate(freqx(niw),freqw(niw),wwx(niw)) !,expa(niw))
00743         call freq01x(niw,     !ua,
00744      o    freqx,freqw,wwx)       !,expa)
00745       endif
00746
00747 c$$$!!  ------ write energy mesh for check ----------
00748 c$$$      ifemesh = iopen('emesh.hsfp0'//xt(nz),1,-1,0)
00749 c$$$      deltax0 = 0d0
00750 c$$$      if(MPI__root) then
00751 c$$$        call writeemesh(ifemesh,freqw,niw,freq,nw,deltax0)
00752 c$$$      endif
00753
00754 !! === readin Vcoud and EPSwklm for newaniso()=T ===
00755       ifidmlx = iopen('EPSwklm',0,0,0)
00756       read(ifidmlx) nq0ix,lxklm
00757       if(nq0i/=nq0ix) then
00758         write(6,*)'nq0i from EPSwklm /= nq0i',nq0i,nq0ix
00759         call rx( 'nq0i from EPSwklm /= nq0i')
00760       endif
00761       allocate( dmlx(nq0i,9))
00762       allocate( epinvq0i(nq0i,nq0i) )
00763       allocate( wklm((lxklm+1)**2))
00764       read(ifidmlx) dmlx, epinvq0i
00765       read(ifidmlx) wklm
00766       ifidmlx = iclose('EPSwklm')
00767
00768 c----tetra block is experimental. unused usually. ----------------
00769       if(tetra) then
00770 c     --- get tetrahedron
00771 c     mxkp = n1*n2*n3
00772 c     allocate( qbzxx(3*mxkp),wbzxx(mxkp),ipq(mxkp) )
00773 c     call bzmesh (plat,qbasmc,n1,n2,n3,w(igrp),ngrp,ipq,
00774 c     .                qbzxx,wbzxx,nqibzxx,mxkp)
00775 c     allocate(idtetx(0:4,mxkp*6))
00776 c     call tetirr(qbasmc,n1,n2,n3,ipq,nqibz,ntet,
00777 c     .              idtetx)
00778 c     allocate(idtet(0:4,ntet))
00779 c     idtet(0:4,1:ntet) = idtetx(0:4,1:ntet)
00780 c     deallocate(idtetx,qbzxx,wbzxx,ipq)
00781 c
00782 c     nene = ntq*nq*nspin ! for energy points.
00783 c     if(exchange) nene=0
00784 c     allocate(wtet(nband,nspin,nqibz,0:3*nene),
00785 c     &    eband(nband,nspin,nqibz), qz(3,nqibz),nstar(nqibz),
00786 c     &    iene(3*ntq,nq,nspin), ene(0:3*nene) ) ! pointer for
00787         allocate(wtet(nband,nspin,nqibz,0:0),
00788      &   eband(nband,nspin,nqibz), qz(3,nqibz) ) ! pointer for
00789         call dcopy(3*nqibz,qibz,1,qz,1)
00790         do  is    = 1,nspin     !Readin eband
00791           do  iqi = 1,nqibz
00792 c     iq       = idxk (qz(1:3,iqi),qbz,nqbz)
00793 c     call rwdd1 (ifev(is), iq, nband, eband(:,is,iqi))
00794             call readeval(qz(1:3,iqi),is, eband(:,is,iqi))
00795           enddo
00796         enddo
00797 c     wtet(nband,nsp,nqibz,iene) where
00798 c     the energy pointer as iene(itp,ip,ispin) corresponding its energy value.
00799 c     ene(0) = ef
00800 c     if(.not.exchange) then
00801 c     ix =0
00802 c     do is = 1,nspin
00803 c     do ip = 1,nq
00804 c     do i  = 1,ntq
00805 c     do iw = -1,1
00806 c     ix  = ix+1
```

```
00807 c     iene(3*i+iw-1,ip,is) = ix
00808 c     ene(ix) = eqx0(i,ip,is) + 2.d0*(dble(iw)-shtw)*deltaw
00809 c     enddo
00810 c     enddo
00811 c     enddo
00812 c     enddo
00813 c     endif
00814 c     do ix = 0,3*nene
00815 c     ene(ix) = ene(ix)-1d-15  ! to avoid coincidence
00816 c     call bzints2(n1,n2,n3,eband,wtet(:,:,:,ix),nqibz,nband,nband,
00817 c     .               nspin,edummy,edummy,edummy,1,ene(ix),2,ntet,idtet)
00818 c     enddo
00819         volwgt = (3d0 - nspin) / ntetf ! ntetf was =6*n1*n2*n3
00820         call bzints2x(volwgt,eband,wtet(:,:,:,0),nqibz,nband,nband,
00821      .   nspin,edummy,edummy,edummyd,1,ef,2,nteti,idteti)
00822         ntot= sum(wtet)
00823 c        if(legas) then
00824 c          write(6,"(' tetra=T ef ntot nexact ratio=',15f12.6)") ef,ntot
00825 c     &     , ef**1.5d0/3d0/pi**2*voltot, ef**1.5d0 /3d0/pi**2*voltot/ntot
00826 c        else
00827 c          write(6,"(' tetra=T ef nvalence)=',15f12.6)") ef,ntot
00828 c        endif
00829         write(6,"(' tetra=T ef nvalence)=',15f12.6)") ef,ntot
00830         if(nspin==1) wtet = wtet/2d0
00831         do iqi = 1,nqibz
00832           wtet(:,:,iqi,:) = wtet(:,:,iqi,:)/nstar(iqi)
00833         enddo
00834         deallocate( eband, qz, ene ) ! pointer for
00835 c -- ibzx denote the index of k{FBZ for given k{1BZ.
00836         allocate(ibzx(nqbz))
00837         call invkibzx(irk,nqibz,ngrp,nqbz,
00838      o   ibzx)
00839       else
00840         allocate(wtet(1,1,1,1), iene(1,1,1)) !dummy
00841       endif
00842 c ---- end of tetra section -------------------------------------------
00843 c      iii=ivsumxxx(irk,nqibz*ngrp)
00844 c      write(6,*) " sum of nonzero iirk=",iii, nqbz
00845
00846
00847 !!-------------------------------------------------------------
00848 !!     calculate the the self-energy SEx(ip) or SEc(ip)
00849 !!-------------------------------------------------------------
00850 !! eibz4sig() is EIBZ symmetrization or not...
00851       if(eibz4sig()) then
00852         allocate(nwgt(nqbz,1:nq),igx(ngrp*2,nqbz,nq))
00853         allocate(igxt(ngrp*2,nqbz,nq), eibzsym(ngrp,-1:1,nq))
00854         iqxini=1
00855         iqxend=nq
00856 c         write(6,"('TimeRevesal switch = ',l1)") timereversal()
00857 c         call eibzgen(nq,symgg,ngrp,q(:,iqxini:iqxend),
00858 c     &        iqxini,iqxend,qbz,nqbz,timereversal(),ginv,iprintx,
00859 c     o        nwgt,igx,igxt,eibzsym,tiii)
00860 !! Check timereversal is required for symmetrization operation or not. If tiii=timereversal=F is enforced,
00861 !! the symmetrization procedure in x0kf_v4h becomes a little time-consuming.
00862         tiii=.false.            !Enforce no time reversal. time reversal not yet...
00863         write(6,*)'NOTE:TimeReversal not yet implemented in hsfp0.sc.m.F'
00864         write(6,"('=== goto eibzgen === used timereversal=',l1)")tiii
00865         iprintx=.false.
00866         if(mpi__root) iprintx=.true.
00867         call eibzgen(nq,symgg,ngrp,q(:,iqxini:iqxend),
00868      &   iqxini,iqxend,qbz,nqbz,tiii,ginv,iprintx,
00869      o   nwgt,igx,igxt,eibzsym,tiiiout)
00870 c     call PBindex(natom,lx,l2nl,nx) !all input. this returns requied index stored in arrays in m_pbindex.
00871 ! PBindex: index for product basis.  We will unify this system; still similar is used in ppbafp_v2.
00872 c     call readqgcou() !no input. Read QGcou and store date into variables.
00873 c     call Spacegrouprot(symgg,ngrp,plat,natom,pos) ! all inputs.
00874 C         do iq=iqxini,iqxini
00875 C         do ibz=1,200
00876 C           if(nwgt(ibz,iq)/=0) then
00877 C             write(6,"('yyy1: ',i8,2x,25(i3,i2))") ibz,(igx(i,ibz,iq),igxt(i,ibz,iq),i=1,nwgt(ibz,iq))
00878 C           endif
00879 C         enddo
00880 C         enddo
00881       endif
00882
00883 !! == irkip control paralellization  ==
00884 !! We have to distribute non-zero irkip into processes (nrank).
00885 !! When irkip(nqibz,ngrp,nq,nspinmx)/=0, we expect grain-size
00886 !! for each job of (iqibz,igrp,iq,isp) is almost the same.
00887 !! Our pupose is to calculate zsec(itp,itpp,iq).
00888 !! Thus we need to set up communicator (grouping) MPI__COMM_iqisp(iq,isp) to do all_reduce.
00889 !! (for given zsec(iq,isp), we take sum on zsec for (iqibz,igrp) by all_reduce.)
00890 !! ---
00891 !! NOTE: in future, we will further extend irkip for itp and itpp
00892       allocate(irkip_all(nspinmx,nqibz,ngrp,nq)) !this is global
00893       allocate(nrkip_all(nspinmx,nqibz,ngrp,nq)) !this is global
```

```
00894       allocate(nrkip(nspinmx,nqibz,ngrp,nq)) !this is global
00895       if(eibz4sig()) then
00896         nrkip_all=0
00897         irkip_all=0
00898         is=1                    ! not spin dependent
00899         do iqq=1,nq
00900 c     irkip_all(is,:,:,iqq)=irk
00901           do kx=1,nqibz
00902             do igrp=1,ngrp
00903               kr = irk(kx,igrp) !ip_all(is,kx,igrp,iqq) !kr is index for qbz (for example, nonzero # of kr
      is 64 for 4x4x4)
00904               if(kr==0) cycle
00905               if(nwgt(kr,iqq)/=0) then
00906                 irkip_all(is,kx,igrp,iqq)= irk(kx,igrp)
00907                 nrkip_all(is,kx,igrp,iqq)= nwgt(kr,iqq)
00908               endif
00909 c     write(6,*)' iqq kr irk =',iqq,kr,irkip_all(is,kx,igrp,iqq),nrkip_all(is,kx,igrp,iqq)
00910             enddo
00911           enddo
00912         enddo
00913 C          do iqq=1,nq
00914 C            write(6,"('iq=',i4,' # of EIBZ: Used(TimeR 1 or -1)=',i3,'=',i3,'+',i3)")
00915 C     &           iqq,sum(eibzsym(:,:,iqq)),sum(eibzsym(:,1,iqq)),sum(eibzsym(:,-1,iqq))
00916 C            write(6,"('eibz: iqq sum(nrkip_all)=nqbz  ',i3,3f11.5,3i8)")
00917 C     &           iqq,q(:,iqq),sum(nrkip_all(is,:,:,iqq)),nqbz
00918 C            do kx=1,nqibz
00919 C              do igrp=1,ngrp
00920 C                kr = irkip_all(is,kx,igrp,iqq) !kr is index for qbz
00921 C                if(kr/=0) write(6,"('      ',i8,3f11.5,i8,2x,25(i4,i2))")
00922 C     &               kr,qbz(:,kr),nrkip_all(is,kx,igrp,iqq)
00923 C     &               ,(igx(i,kr,iqq),igxt(i,kr,iqq),i=1,nwgt(kr,iqq))
00924 C              enddo
00925 C            enddo
00926 C!     !   Probably partial group symmetrization is enough. But it may not reduce computational time so
      much.
00927 C         enddo
00928         if(nspinmx==2) then
00929           irkip_all(2,:,:,:)=irkip_all(1,:,:,:)
00930           nrkip_all(2,:,:,:)=nrkip_all(1,:,:,:)
00931         endif
00932       else                      ! not eibz4sig
00933         do is = 1,nspinmx
00934           do iqq=1,nq
00935             irkip_all(is,:,:,iqq)=irk
00936           enddo
00937         enddo
00938       endif
00939
00940 !! -- ppb= <Phi(SLn,r) Phi(SL'n',r) B(S,i,Rr)>
00941       allocate( ppbir(nlnmx*nlnmx*mdimx*nclass,ngrp,nspin))
00942       do irot = 1,ngrp
00943         do isp = 1,nspin
00944           call ppbafp_v2(irot,ngrp,isp,nspin,
00945      i     il,in,im,nlnm,       !w(i_mnl),
00946      i     nl,nn,nclass,nlnmx,
00947      i     mdimx,lx,nx,nxx,     !Bloch wave
00948      i     cgr, nl-1,           !rotated CG
00949      i     ppbrd,               !radial integrals
00950      o     ppbir(:,irot,isp))   !this is in m_zmel
00951         enddo
00952       enddo
00953
00954 !! MPI RankDivider for iqibz and irot cycle in sxcf.
00955 !!     nrkip is weight correspoinding to irkip for a node.
00956       allocate(irkip(nspinmx,nqibz,ngrp,nq)) !local
00957       call mpi__sxcf_rankdivider(irkip,irkip_all,nspinmx,nqibz,ngrp,nq) ! MIZUHO-IR
00958       nrkip = nrkip_all         ! we don't need to change this for MPI case.
00959 ! It just need to distribute non-zero irkip.
00960 !!
00961       nlnx4    = nlnx**4        ! niwx     = max0 (nw+1,niw) !nw --->nw+1 feb2006
00962       allocate( kount(nqibz,nq),zsec(ntq,ntq,nq), coh(ntq,nq) )
00963 !TIME1_0010 "main:before2000loop"
00964       do 2000 is = 1,nspinmx
00965 !TIME0_0020
00966         if(mpi__root) then
00967           if(exchange) then
00968             write(ifsex2(is)) nspin, nq, ntq,nqbz,nqibz, n1,n2,n3
00969             write(ifsex(is),*) '======================================'
00970             write(ifsex(is),"('Self-energy exchange SEx(q,t): is=',i3)") is
00971             write(ifsex(is),*) '======================================'
00972             call winfo(ifsex(is),nspin,nq,ntq,is,nbloch,ngpn1,
00973      &       ngcn1,nqbz,nqibz,ef,deltaw,alat,esmr)
00974             write (ifsex(is),*)' *** '
00975             write (ifsex(is),"(a)") ' jband   iq ispin                 '//
00976      &       '           qvec            eigen-Ef (in eV)           exchange (in eV)'
00977           elseif(ixc==2) then
00978             write(ifsec2(is)) nspin, nq, ntq ,nqbz,nqibz  ,n1,n2,n3
```

```
00979             write(ifsec(is),*) '========================================='
00980             write(ifsec(is),"('Self-energy correlated SEc(qt,w): is=',i3)") is
00981             write(ifsec(is),*) '========================================='
00982             call winfo(ifsec(is),nspin,nq,ntq,is,nbloch,ngpn1,
00983      &       ngcn1,nqbz,nqibz,ef,deltaw,alat,esmr)
00984             write (ifsec(is),*)' *** '
00985             write (ifsec(is),"(a)") ' jband   iq ispin                 '//
00986      &       '           qvec            eigen-Ef (in eV)          '//
00987      &       'Re(Sc) 3-points (in eV)                        '//
00988      &       '           In(Sc) 3-points (in eV)               Zfactor(=1)'
00989           endif
00990         endif
00991         zsec  = 0d0
00992         coh   = 0d0
00993         kount = 0
00994         if(ixc==3.and.nctot==0) goto 2001 !make dummy SEXcore
00995 !! dummy to overlaid -check bounds sep2014
00996         if(size(ecore)==0) then
00997           deallocate(ecore)
00998           allocate(ecore(1,2))
00999         endif
01000
01001 !!== ip loop to spedify external q ==
01002 c        do 1001 ip = 1,nq
01003 c          if(sum(irkip(is,:,:,ip))==0) cycle
01004         call sxcf_fal3_scz(kount,q,itq,ntq,ef,esmr,
01005      i   nspin,is,
01006      i   qbas,ginv,qibz,qbz ,wbz, nstbz,
01007      i   irkip(is,:,:,:),nrkip(is,:,:,:),
01008      i   freq_r,nw_i,nw, freqx,wwx,
01009      i   dw,
01010      i   ecore(:,is),
01011      d   nlmto,nqibz,nqbz,nctot,
01012      d   nbloch,ngrp,niw,nq,
01013      i   nblochpmx, ngpmx,ngcmx,
01014      i   wgt0,nq0i,q0i,symgg,alat,
01015      i   nband,                  !shtvg,
01016      i   ifvcfpout,
01017      i   exchange, screen, cohtest, ifexsp(is),
01018      i   nbmx,ebmx,
01019      i   wklm,lxklm,
01020      i   eftrue,
01021      i   jobsw = isigmode, nbandmx=nbandmx(1:nq,is), !nbandmx is input mar2015
01022      i   hermitianw=hermitianw,
01023      o   zsec=zsec)
01024 c 1001   continue
01025 !TIME1_0020 "main:endofsxcf_fal3_scz"
01026 c        call date_and_time(values=timevalues)
01027 c        write(6,'(a,9i5)')'dateandtime2=',MPI__rank,timevalues(1:8)
01028 c        call cpu_time(time_red1)
01029
01030 !! CAUITION! Allreduce wait all cpu jobs done here.
01031 !! Before nov2013, MPI__sxcf_rankdivider was stpid---> half of cores assigned for isp=2
01032 !! was just waiting here!
01033 c      call MPI__AllreduceMax( nbandmx(:,is), nq ) ! MIZUHO-IR
01034 c      call cpu_time(time_red2)
01035 c      write(6,*) MPI__rank,'time(MPI__AllreduceMax)=',time_red2-time_red1
01036
01037 c$$$!!  electron gas bare exchange (exact)
01038 c$$$        if (legas.and.exchange) then
01039 c$$$          efz=(ntot*3*pi**2/voltot)**(2d0/3d0) ! ef is calculated from ntot.
01040 c$$$          pi         = 4.d0*datan(1.d0)
01041 c$$$          tpia       = 2.d0*pi/alat
01042 c$$$          qfermi= dsqrt(efz)
01043 c$$$          alpha = (9*pi/4d0)**(1d0/3d0)
01044 c$$$          write (6,*)' --- exact electron gas bare exchange --- '
01045 c$$$          write (6,*)' density parameter rs= ', alpha/qfermi
01046 c$$$          write (6,*)' kf= ',qfermi
01047 c$$$          do      ip = 1,nq
01048 c$$$            qreal =  tpia*q(1:3,ip)
01049 c$$$            qm    = dsqrt ( sum(qreal**2) )
01050 c$$$            xsex  = hartree * egex (qm,efz)
01051 c$$$            write (6,*)
01052 c$$$            write (6,"(' True qm-ef Sx=',2f14.6,' q/qf=',f14.6)")
01053 c$$$     &       rydberg()*(qm**2-efz), xsex, qm/qfermi
01054 c$$$            write (6,"(' Num  qm-ef Sx=',2f14.6)")
01055 c$$$     &       eqx(1,ip,is),        hartree*dreal(zsec(1,1,ip)) !sf 21May02
01056 c$$$            write (6,"(' === diff     =',2f14.6)")
01057 c$$$     &       rydberg()*(qm**2-efz)-eqx(1,ip,is)
01058 c$$$     &       , xsex - hartree*dreal(zsec(1,1,ip)) !sf 21May02
01059 c$$$            write (661,"(' qm True qm-ef Sx=',3f14.6)")
01060 c$$$     &       qm,rydberg()*(qm**2-efz), xsex
01061 c$$$            write (662,"(' qm Num  qm-ef Sx=',3f14.6)")
01062 c$$$     &       qm,eqx(1,ip,is),     hartree*dreal(zsec(1,1,ip)) !sf 21May02
01063 c$$$ccc   write (ifsex(is),6600) qreal(1),qreal(2),qreal(3),xsex
01064 c$$$ccc   write (6,6600) qreal(1),qreal(2),qreal(3),xsex
01065 c$$$ccc   6600   format (' qreal =',3f8.4,'   SEx(q) =',d13.5)
```

```
01066 c$$$            write (663,"(2f14.6)") qm/qfermi, qfermi
01067 c$$$          end do
01068 c$$$        endif
01069  2001   continue
01070
01071 !! eibz4sig symmetrization. MPI__AllreduceSum in zsecsym.
01072        if(eibz4sig()) then
01073 !TIME0_0030
01074          call zsecsym(zsec,ntq,nq,nband,nbandmx,nspinmx, eibzsym,ngrp,tiii,q,is)
01075 !TIME1_0030 'zsecsym'
01076        endif
01077 !TIME0_0040
01078        call mpi__allreducesum( zsec,ntq*ntq*nq )
01079 !TIME1_0040 'MPI__AllreduceSumzsec'
01080
01081        if(mpi__root) then
01082          if(exchange) then
01083            ifoutsex=ifsex(is)
01084            write(6,*)
01085            do ip = 1,nq
01086              do i  = 1,ntq
01087                write(ifoutsex,"(3i5,3d24.16,3x,d24.16,3x,d24.16)")
01088     &           itq(i),ip,is, q(1:3,ip), eqx(i,ip,is),
01089     &           hartree*dreal(zsec(i,i,ip)) !sf 21May02
01090                if( eqx(i,ip,is)<1d20.and.abs(zsec(i,i,ip))/=0d0 ) then !takao june2009
01091                  write(6,"(' j iq isp=' i3,i4,i2,'  q=',3f8.4,' eig=',f10.4,'  Sx=',f10.4)")
01092     &             itq(i),ip,is, q(1:3,ip), eqx(i,ip,is),
01093     &             hartree*dreal(zsec(i,i,ip)) !sf 21May02
01094                endif
01095              end do
01096              write(ifsex2(is)) is, q(1:3,ip), zsec(1:ntq,1:ntq,ip) !SEC_nn' out
01097            end do
01098          elseif(ixc==2) then
01099            ifoutsec=ifsec(is)
01100            do ip = 1,nq
01101              do i  = 1,ntq
01102                if( eqx(i,ip,is)<1d20.and.abs(zsec(i,i,ip))/=0d0 ) then !takao june2009
01103                  write(6,"(' j iq isp=' i3,i4,i2,'  q=',3f8.4,'  eig=',f8.4,'  Re(Sc) =',f8.4,'  Img(Sc)
       =',f8.4 )")
01104     &             itq(i),ip,is, q(1:3,ip), eqx(i,ip,is),
01105     &             hartree*dreal(zsec(i,i,ip)),
01106     &             hartree*dimag(zsec(i,i,ip))
01107                endif
01108                write(ifoutsec,"(3i5,3d24.16,3x,d24.16,3x,d24.16, 3x,d24.16)")
01109     &           itq(i),ip,is, q(1:3,ip), eqx(i,ip,is),
01110     &           hartree*dreal(zsec(i,i,ip)),
01111     &           hartree*dimag(zsec(i,i,ip))
01112              end do
01113              write(ifsec2(is)) is, q(1:3,ip), zsec(1:ntq,1:ntq,ip) !SEC_nn' out
01114            end do
01115          endif                 !ixc
01116        endif                   !MPI__root
01117
01118  2000 continue                 !end of spin-loop
01119
01120 c$$$!!   --- EXspectrum ----------------------------------------------------
01121 c$$$c     This section is similar with efsimplef.f
01122 c$$$      if(sum(ifexsp(1:nspin))/=0) then
01123 c$$$         do is = 1,nspin
01124 c$$$            write(6,*)' --- Goto ExSpectrum section --- is=',is
01125 c$$$            rewind (ifexsp(is))
01126 c$$$            itmx = 0
01127 c$$$            do
01128 c$$$               read(ifexsp(is),*,end=1215)ipex,itpex,itex,qqex(1:3), eex,exsp
01129 c$$$               if(itex>itmx) itmx=itex
01130 c$$$            enddo
01131 c$$$ 1215       continue
01132 c$$$            nspexmx = itmx*(nqbz+nq0i*ngrp) !Get marimum value of the number of the ex spectrum
01133 c$$$c
01134 c$$$            allocate( eex1(nspexmx,ntq,nq), exsp1(nspexmx,ntq,nq),
01135 c$$$     &           nspex(ntq,nq) ,
01136 c$$$     &           itex1(nspexmx,ntq,nq),
01137 c$$$     &           qqex1(3,nspexmx,ntq,nq) )
01138 c$$$            write(6,*)' nspexmx =',nspexmx
01139 c$$$c
01140 c$$$            rewind (ifexsp(is))
01141 c$$$            nspex = 0
01142 c$$$            do
01143 c$$$               read(ifexsp(is),*,end=1216) ipex,itpex,itex,qqex(1:3),eex,exsp
01144 c$$$               nspex(itpex,ipex) = nspex(itpex,ipex)+1
01145 c$$$               iex  = nspex(itpex,ipex)
01146 c$$$               eex1  (iex,itpex,ipex) = eex
01147 c$$$               exsp1 (iex,itpex,ipex) = exsp
01148 c$$$               itex1 (iex,itpex,ipex) = itex
01149 c$$$               qqex1(:,iex,itpex,ipex)= qqex
01150 c$$$            enddo
01151 c$$$ 1216       continue            !Get eex1(1:nspex) exsp1(1:nspex) for itp ip.
```

```
01152 c$$$            write(6,*)' nspex(1 1)=',nspex(1,1)
01153 c$$$c
01154 c$$$            do ipex = 1,nq
01155 c$$$               do itpex=1,ntq
01156 c$$$                  write(6,*)' is itq ip =',is,itq,ip
01157 c$$$                  nnex = nspex(itpex,ipex)
01158 c$$$                  allocate( ieord(1:nnex) )
01159 c$$$                  call sortea( eex1(1:nnex,itpex,ipex),ieord, nnex,isig)
01160 c$$$                  eex1 (1:nnex,itpex,ipex)  = eex1  (ieord(1:nnex),itpex,ipex)
01161 c$$$                  exsp1 (1:nnex,itpex,ipex) = exsp1 (ieord(1:nnex),itpex,ipex)
01162 c$$$                  itex1 (1:nnex,itpex,ipex) = itex1 (ieord(1:nnex),itpex,ipex)
01163 c$$$                  qqex1(:,1:nnex,itpex,ipex)= qqex1 (:,ieord(1:nnex),itpex,ipex)
01164 c$$$
01165 c$$$                  filenameex = 'EXSP'//charnum3(ipex)//charnum3(itpex)
01166 c$$$     &                 //'.'//char(48+is)
01167 c$$$                  ifexspx=4111
01168 c$$$                  open(ifexspx,file=filenameex)
01169 c$$$
01170 c$$$                  filenameex = 'EXSS'//charnum3(ipex)//charnum3(itpex)
01171 c$$$     &                 //'.'//char(48+is)
01172 c$$$                  ifexspxx=4112
01173 c$$$                  open(ifexspxx,file=filenameex)
01174 c$$$
01175 c$$$                  do i=1,nnex
01176 c$$$                     write(ifexspx, "(2d14.6, i4, 3f14.6)")
01177 c$$$     &                    eex1  (i,itpex,ipex), exsp1 (i,itpex,ipex),
01178 c$$$     &                    itex1 (i,itpex,ipex), qqex1 (1:3,i,itpex,ipex)
01179 c$$$                  enddo
01180 c$$$c
01181 c$$$                  eee  =-1d99
01182 c$$$                  exwgt= 0d0
01183 c$$$                  do i=1,nnex
01184 c$$$                     if(eex1(i,itpex,ipex) > eee+1d-4 .or. i==nnex) then
01185 c$$$                        if(i/=1) write(ifexspxx, "(2d23.15)")
01186 c$$$     &                       eee, exwgt*hartree
01187 c$$$                        eee  = eex1(i,itpex,ipex)
01188 c$$$                        exwgt= exsp1 (i,itpex,ipex)
01189 c$$$                     else
01190 c$$$                        exwgt= exwgt + exsp1 (i,itpex,ipex)
01191 c$$$                     endif
01192 c$$$                  enddo
01193 c$$$c
01194 c$$$                  deallocate( ieord )
01195 c$$$                  close(ifexspx)
01196 c$$$                  close(ifexspxx)
01197 c$$$               enddo
01198 c$$$            enddo
01199 c$$$            deallocate( eex1, exsp1, nspex, itex1, qqex1 )
01200 c$$$         enddo
01201 c$$$         write(6,*)' End of ExSpectrum section ---'
01202 c$$$      endif
01203 c      isx = iclose ('wc.d')
01204 c      isx = iclose ('wci.d')
01205 c      isx = iclose ('hbe.d')
01206       call cputid(0)
01207       write(6,*) '--- end of hsfp0_sc --- irank=',mpi__rank
01208       call flush(6)
01209       call mpi__finalize
01210 !TIME1_0000 "main:totalofhsfp0_sc"
01211 !TIMESHOW
01212       if(ixc==1 ) call rx0( ' OK! hsfp0_sc: Exchange mode')
01213       if(ixc==2 ) call rx0( ' OK! hsfp0_sc: Correlation mode')
01214       if(ixc==3 ) call rx0( ' OK! hsfp0_sc: Core-exchange mode')
01215       end program hsfp0_sc
01216
01217
01218
01219       subroutine zsecsym(zsec,ntq,nq,nband,nbandmx,nspinmx, eibzsym,ngrp,tiii,q,is)
01220 !! --- symmetrize zsec for eibz4sig mode. -----------------
01221 !! Read a file lmfgw_kdivider, which contains info for vxc and evec (they are in separated files in MPI)
01222 !!
01223 c      use m_mpi,only: MPI__AllreduceSum
01224       use m_readeigen,only: readeval
01225       implicit none
01226       complex(8),intent(inout)::zsec(ntq,ntq,nq)
01227       integer,intent(in)::ntq,nq,nspinmx,nband,nbandmx(nq,nspinmx),is
01228       integer,intent(in):: ngrp,eibzsym(ngrp,-1:1,nq)
01229       logical,intent(in):: tiii !time reversal switch
01230       real(8),intent(in):: q(3,nq)
01231
01232       complex(8),allocatable::zsect(:,:)
01233       integer:: ifile_handle,iqq
01234       integer:: procid,nrankv,ifvxc_,ifevec_,ifiproc,iqqxx,
01235      & isp,ixx,ixxx,nqixx,nspxx,ispxx,iqbz,i,igrp,iq
01236       character*256:: extn,ext
01237       character*256,allocatable:: extp(:)
01238       integer,allocatable:: ifevec__(:),ifvxc__(:),iprocq(:,:)
```

```
01239
01240       integer:: nsym,nhdim,it,nblk,iband,napw,ldim,ierr,ispx,nbsize,nbsizemx
01241      & ,iblk1,iblk2,ii1,ii2,ie1,ie2,ne1,ne2,iqxx, ndimhx, nspx,nnnx
01242       integer,allocatable::iblki(:),iblke(:)
01243       complex(8),allocatable:: evec(:,:),evec_inv(:,:),evecrot(:,:),rmatjj(:,:,:)
01244       real(8),allocatable::evaliq(:)
01245       real(8)::tolry=1d-4,qqqx(3),qtarget(3)
01246       complex(8),allocatable:: ovl(:,:)
01247       integer::nev,j
01248 !TIME0_0100
01249       write(6,*)'zsecsym:'
01250       allocate( zsect(ntq,ntq))
01251 !! === readin lmfgw_kdivider, and get extensions === apr2013
01252       ifiproc=ifile_handle()
01253       open(unit=ifiproc,file='lmfgw_kdivider',status='old')
01254       read(ifiproc,*) ext
01255       read(ifiproc,*) nqixx, nspxx, nrankv
01256       if(allocated(iprocq)) deallocate(iprocq)
01257       allocate(iprocq(nqixx,nspxx))
01258       do isp=1,nspxx
01259         do iqq=1,nqixx
01260           read(ifiproc,*) iqqxx, ispxx, ixxx
01261           if(iqqxx/=iqq) call rx( 'iqqxx/=iqq')
01262           if(ispxx/=isp) call rx( 'ispxx/=isp')
01263           iprocq(iqq,isp) = ixxx
01264           write(6,"('iqq isp irank=',i8,i2,i6)") iqq,isp, iprocq(iqq,isp)
01265         enddo
01266       enddo
01267       close(ifiproc)
01268 !! for multiple files.
01269 c      if(allocated(extp)) deallocate(extp,ifvxc__,ifevec__)
01270       allocate(extp(0:nrankv-1),ifvxc__(0:nrankv-1),ifevec__(0:nrankv-1))
01271       extp(0) = trim(ext)
01272       write(6,"('  0 ext= ',a,a)") trim(extp(0)),' ----------'
01273       do procid=1,nrankv-1
01274         write(extn,"(i10)") procid
01275         extp(procid)=trim(adjustl(ext))//'_'//trim(adjustl(extn))
01276         write(6,"(i3,' ext= ',a,a)") procid,trim(extp(procid)),' ----------'
01277       enddo
01278       do procid=0,nrankv-1
01279         ifvxc__(procid) = ifile_handle()
01280         open( ifvxc__(procid), file='vxc'//extp(procid),form='unformatted')
01281         ifevec__(procid)= ifile_handle()
01282         open( ifevec__(procid), file='evec'//extp(procid),form='unformatted')
01283       enddo
01284       ifvxc_ = ifvxc__(0)       !0 is root
01285       ifevec_= ifevec__(0)
01286       read(ifevec_)  ndimhx, nspx,nnnx
01287       read(ifvxc_)              !skip ndimh, nsp,nnn
01288       allocate(evaliq(nband),iblki(nband),iblke(nband))
01289 !TIME1_0100 "zsecsym:endof_allocate_zsect"
01290 !TIME0_0110
01291       iqq=0                     !iqq is to read multiple vxc.* evec.*
01292       do 3020 iq=1,nq           !nq means iq for which we will calculate sigma
01293         iqq=iqq+1
01294         do 3030 ispx=1,nspinmx  !ispx loop is to find isx=is
01295           ifvxc_  = ifvxc__(iprocq(iqq,ispx))
01296           ifevec_ = ifevec__(iprocq(iqq,ispx))
01297           if(ispx==is) then
01298 !this if-block is due to evec and v_xc file-->they shall be divieded into spin files.
01299             read(ifvxc_)  nhdim,ldim
01300             read(ifvxc_)
01301             allocate( evec(nhdim,nhdim),evecrot(nhdim,nhdim))
01302             read(ifevec_) qqqx(1:3), evec(1:nhdim,1:nhdim),nev !nev number of true bands nov2015
01303             zsect = 0d0
01304           else                  !skip isx/=is. Need to get access sequential files evec and v_xc.
01305             read(ifvxc_)
01306             read(ifvxc_)
01307             read(ifevec_)
01308             cycle
01309           endif
01310           do i=1,nnnx           !nq     !qqqx from evec v_xc.
01311             if(sum(abs(qqqx-q(:,i)))<1d-6) then
01312               iqxx=i
01313               goto 3011
01314             endif
01315           enddo
01316           deallocate(evec,evecrot)
01317           call rx( 'hsfp0_sc: bug:qqqx can not find ...')
01318  3011     continue
01319           if(tiii) call rx( 'timereversal is not yet implemented')
01320
01321 !! evec_inv(ib1,iww)= \sum_ib2  ovlinv(ib1,ib2)*dconjg(evec(iww,ib2))  nov2015, we introduce nev. iww is
       for PMT basis. ib for band index.
01322 !! This is for converting rotated evec (=evecrot(ib)) in the representation of original evec(ib).
01323           allocate(ovl(nev,nev))
01324 c          print *,'nnnnnnnnn zsecsym: nband=',nhdim,nband,nev
```

```
01325           do i=1,nev
01326           do j=1,nev
01327 c            write(6,*)'evec orth=',i,j,sum(dconjg(evec(:,i)*evec(:,j)))
01328             ovl(i,j)=sum(dconjg(evec(:,i))*evec(:,j))
01329           enddo
01330           enddo
01331           call matcinv(nev,ovl) !ovl --> ovlinv
01332           allocate(evec_inv(nev,nhdim))
01333           evec_inv = matmul(ovl(1:nev,1:nev),dconjg(transpose(evec(:,1:nev)))) !note ovl means ovlinv
01334           deallocate(ovl)
01335 c          evec_inv = evec
01336 c          call matcinv(nhdim,evec_inv)
01337           call readeval(q(:,iqxx), is, evaliq)
01338           nsym = sum(eibzsym(:,:,iqxx))
01339           do it=1,1             !no-time reversal yet !it=1,-1,-2 !c.f. x0kf_v4h
01340             do igrp=1,ngrp      !A-rotator
01341               if( eibzsym(igrp,it,iqxx)==0) cycle
01342               nblk=0
01343               iblki=0
01344               iblke=0
01345               iblki(1)=1
01346 !!  degeneracy divider for evaliq. See How to apply EIBZ to
01347 !! Is this procedure really make speed up so much?
01348               tolry= 0.2d0      !Degeneracy tol. if tolry is large,
01349 !! larger tolry is safer, although a little inefficient.
01350 !! If tolry is too small to divide degenerated values to different blocks --> then we have wrong results.
01351 !(NOTE that Hamiltonian can be not so symmetric in some reasons)
01352               nbsizemx=0
01353               do iband=2,nbandmx(iqxx,is)
01354 ! nbandmx is the number of bands for which we calculate self-energy.
01355 ! We assume nbandmx(iqxx,is) is well separated for degeneracy.
01356                 if(evaliq(iband) > evaliq(iband-1)+tolry
01357      &            .or.iband==nbandmx(iqxx,is)) then
01358                   nblk=nblk+1
01359                   if(nblk>=2) iblki(nblk)=iblke(nblk-1)+1
01360                   if(iband==nbandmx(iqxx,is)) then
01361                     iblke(nblk)=iband
01362                   else
01363                     iblke(nblk)=iband-1
01364                   endif
01365                   nbsize = iblke(nblk)- iblki(nblk)+1
01366                   if( nbsize>nbsizemx ) nbsizemx = nbsize
01367                 endif
01368               enddo             ! iband
01369 !! rotation of evec. Generate evecrot. (Within degenerated block, evec are mapped).e
01370               allocate(rmatjj(nbsizemx,nbsizemx,nblk))
01371               napw=nhdim-ldim
01372               do iblk1=1,nblk
01373                 ii1=iblki(iblk1)
01374                 ie1=iblke(iblk1)
01375                 ne1=ie1-ii1+1
01376                 call rotwvigg(igrp,q(:,iqxx),q(:,iqxx),nhdim,
01377      &            napw,ne1,evec(:,ii1:ie1),evecrot(:,ii1:ie1),ierr )
01378                 rmatjj(1:ne1,1:ne1,iblk1) =
01379      &            matmul(evec_inv(ii1:ie1,:),evecrot(:,ii1:ie1))
01380               enddo             ! iblk1
01381               do iblk1=1,nblk
01382                 do iblk2=1,nblk
01383                   ii1=iblki(iblk1)
01384                   ie1=iblke(iblk1)
01385                   ne1=ie1-ii1+1
01386                   ii2=iblki(iblk2)
01387                   ie2=iblke(iblk2)
01388                   ne2=ie2-ii2+1
01389                   zsect(ii1:ie1,ii2:ie2)= zsect(ii1:ie1,ii2:ie2)
01390      &             + matmul( dconjg(transpose(rmatjj(1:ne1,1:ne1,iblk1))),
01391      &             matmul(zsec(ii1:ie1,ii2:ie2,iqxx),
01392      &             rmatjj(1:ne2,1:ne2,iblk2)) )
01393                 enddo           ! iblk2
01394               enddo             ! iblk1
01395               deallocate(rmatjj)
01396             enddo               ! igrp
01397           enddo                 ! it
01398           deallocate(evec, evec_inv, evecrot)
01399           zsec(:,:,iqxx) = zsect(:,:)/dble(nsym)
01400 c          call MPI__AllreduceSum( zsec(:,:,iqxx),ntq*ntq ) ! MIZUHO-IR
01401  3030   continue               ! ispx
01402  3020 continue                 ! iq
01403       do procid=0,nrankv-1
01404         close(ifvxc__(procid) )
01405         close(ifevec__(procid))
01406       enddo
01407       deallocate(iblki,iblke,evaliq)
01408       deallocate(zsect,extp,ifevec__,ifvxc__,iprocq)
01409 !TIME1_0110 "sub_zsecsym"
01410       end subroutine zsecsym
```

4.33 main/hvccfp0.m.F File Reference

Functions/Subroutines

- program hvccfp0
- subroutine checkagree (a, b, char)
- subroutine mkradmatch (p, nxdim, rdmatch)
- subroutine phimatch (p, pd, p1, p1d, p2, p2d, s, t)
- subroutine pmatorth (oo, oon, pmat, no, nn, pomat)
- subroutine diagcvh (hh, ngb, eb, zz)
- subroutine zgesvdnn2 (no, nn, nnmx, epsmx, pmat, nnn)
- subroutine mkb0 (q, lxx, lx, nxx, nx, aa, bb, nrr, nrx, rprodx, alat, bas, nbas, nbloch, b0mat)

4.33.1 Function/Subroutine Documentation

4.33.1.1 subroutine checkagree (real(8), dimension(3) *a,* real(8), dimension(3) *b,* character∗(∗) *char*)

Definition at line 1294 of file hvccfp0.m.F.

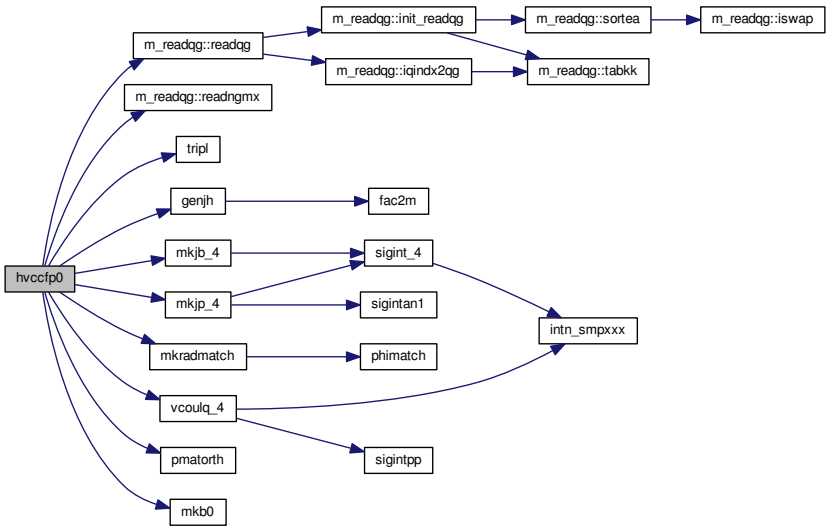
4.33.1.2 subroutine diagcvh (complex(8), dimension(ngb,ngb) *hh,* integer(4) *ngb,* real(8), dimension(ngb) *eb,* complex(8), dimension(ngb,ngb) *zz*)

Definition at line 1420 of file hvccfp0.m.F.

4.33.1.3 program hvccfp0 ( )

Definition at line 1 of file hvccfp0.m.F.

Here is the call graph for this function:



4.33.1.4 subroutine mkb0 (real(8), dimension(3) *q,* integer(4) *lxx,* integer(4), dimension(nbas) *lx,* integer(4) *nxx,* integer(4), dimension(0:lxx,nbas) *nx,* real(8), dimension(nbas) *aa,* real(8), dimension(nbas) *bb,* integer(4), dimension(nbas) *nrr,* integer(4) *nrx,* real(8), dimension(nrx,nxx,0:lxx,nbas) *rprodx,* real(8) *alat,* real(8), dimension(3,nbas) *bas,* integer(4) *nbas,* integer(4) *nbloch,* complex(8), dimension(nbloch) *b0mat*)

Definition at line 1475 of file hvccfp0.m.F.

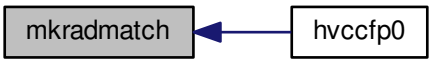
Here is the caller graph for this function:

4.33.1.5 subroutine mkradmatch (real(8), dimension(1:2, 1:nxdim) *p,* integer(4) *nxdim,* real(8), dimension(1:nxdim,1:nxdim) *rdmatch*)

Definition at line 1304 of file hvccfp0.m.F.

Here is the call graph for this function:

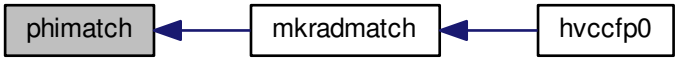
Here is the caller graph for this function:



4.33.1.6 subroutine phimatch (real(8) *p,* real(8) *pd,* real(8) *p1,* real(8) *p1d,* real(8) *p2,* real(8) *p2d,* real(8) *s,* real(8) *t*)

Definition at line 1373 of file hvccfp0.m.F.

Here is the caller graph for this function:



4.33.1.7 subroutine pmatorth (complex(8), dimension(no,no) *oo,* complex(8), dimension(nn,nn) *oon,* complex(8), dimension(no,nn) *pmat,* integer(4) *no,* integer(4) *nn,* complex(8), dimension(nn,no) *pomat*)

Definition at line 1396 of file hvccfp0.m.F.

Here is the caller graph for this function:

4.33.1.8 subroutine zgesvdnn2 (integer(4) *no,* integer(4) *nn,* integer(4) *nnmx,* real(8) *epsmx,* complex(8), dimension(no,nn) *pmat,* integer(4) *nnn*)

Definition at line 1437 of file hvccfp0.m.F.

4.34 hvccfp0.m.F

```
00001       program hvccfp0
00002 c- Coulomb matrix. <f_i | v| f_j>_q. ------------------------
00003 c input files
00004 c    HVCCIN      : some inputs by hbg0.
00005 c    PLN         : plane wave expansion data by nbg0.
00006 c    BASFP//atom : product basis by hbasfp0. ic=iatom should be kept!
00007 c output
00008 c    VCCFP : the coulomb matrix vcoul(nblochpmx,nblochpmx) for all qibz.
00009 c-------------------------------------------------------------
00010 c int
00011 c    strx: structure constant for e=0 (means 1/|r-r'| )
00012 c
00013       use m_readqg,only: readqg,readngmx
00014       use keyvalue,only: getkeyvalue
00015       use m_mpi,only: mpi__hx0fp0_rankdivider2,mpi__task,mpi__initialize,mpi__finalize,mpi__root,
00016      &   mpi__broadcast,mpi__dblecomplexsend,mpi__dblecomplexrecv,mpi__rank,mpi__size,
00017      &   mpi__ranktab,mpi__consoleout,mpi__iend,mpi__iini,mpi__getrange
00018
00019       implicit none
00020       integer(4) :: ifvcfpout,iopen,ifhvccfp,is, nqbz, nbas,lmxcg,
00021      &  nband, ifplane,ngpmx, ngcmx, nblochpmx, nbloch,
00022      &  ibas,ic,lxx,nxx,nrx,l,n,k,isx,kdummy, iclose,
00023      &  nkdmx,nkqmx,lmax,nkdest,nkrest,ngp,ngc,nlxx,i,lnjcg,lnxcg,
00024      &  nkd,nkq ,ibas1,ibas2,nlx1,nlx2, nqibz,iqibz
00025       real(8) :: alat, plat(3,3),qlat(3,3),q(3),p(3),voltot,
```

```
00026      &  pi,fpi,tripl,alat0,epsx,
00027      &  tol,as,tpiba,qb0(3,3),vol0,rdist0,qdist0,radd,qadd,
00028      &  a0,awald,alat1,tol1,r0,q0,awald0,qg(3),   absqg2,aaa,aaa12
00029       integer(4),allocatable :: jcg(:),indxcg(:),
00030      &  lx(:),kmx(:),nblocha(:),nr(:),ificrb(:),
00031      &  nx(:,:),ngvecp(:,:),ngvecc(:,:),ngvecci(:,:,:),iqibzx(:)
00032       real(8),allocatable :: qbz(:,:),qibz(:,:),bas(:,:),rmax(:),
00033      &  cg(:),rprodx(:,:,:,:),dlv(:,:),qlv(:,:),work(:),ngcn(:),
00034      &  rojb(:,:,:), sgbb(:,:,:,:),aa(:),bb(:),rofit(:),phi(:),psi(:),
00035      &  wqt(:), q0i(:,:)
00036       complex(8) ,allocatable :: vcoul(:,:),geig(:,:),strx(:,:,:,:),
00037      &  sgpb(:,:,:,:),sgpp(:,:,:,:),
00038      &  fouvb(:,:,:,:),fouvp(:,:,:,:),vcoul0(:,:),
00039      &  s(:,:),sd(:,:),rojp(:,:,:) , vcoulnn(:,:)
00040       character*7,allocatable :: filename(:)
00041       character(20) :: xxt
00042
00043       complex(8):: phasep,img=(0d0,1d0)
00044       integer(4)::ir,ig1,n1,n2
00045
00046       complex(8),allocatable :: hh(:,:),oox(:,:),ooxi(:,:),oo(:,:),zz(:,:),zzr(:)
00047       real(8),allocatable    :: eb(:)
00048
00049       complex(8),allocatable :: matp(:),matp2(:)
00050       complex(8) :: xxx,trwv
00051       integer(4) :: ngb,nev,nmx,iqx,ip11,ip12,nq0i,igx1,igx2
00052       logical checkeig
00053       logical:: besseltest=.false. !test
00054       real(8) :: sss1,sss2,dnorm
00055 c
00056
00057       complex(8),allocatable:: gbvec(:), ppovl(:,:), b0mat(:)
00058
00059       integer(4) ::igc,igc0,ifgb0vec,ifgb0vec1,ix, iy
00060
00061       integer(4) :: iqxini, iqxend,imode
00062       logical :: allochk=.false. !paralellx0=.true.,
00063
00064       complex(8),allocatable:: hh1(:,:),oo1(:,:)
00065       integer(4)::  nqnumc,ifiqgc !bzcase,
00066 c      character(5)  :: charnum5
00067 c      integer(4),allocatable:: iqok(:)
00068       real(8):: qqq(3),qpgcut_cou       !,qq(3)
00069
00070       integer(4),allocatable:: ngvecc0(:,:)
00071       integer(4):: ngc0
00072
00073       real(8):: ginv(3,3),quu(3),det
00074
00075 c---
00076       real(8),allocatable :: rkpr(:,:,:),rkmr(:,:,:),rofi(:,:)
00077       real(8):: eee,eees, q_org(3),screenfac
00078       integer(4):: ifvcfporg,nqbz_in,nblochpmx_in
00079       complex(8),allocatable:: vcoul_org(:,:)
00080
00081       logical :: smbasis,debug=.false.,smbb
00082       integer(4)    :: ifprodmt,nl_r,lx_,nxx_r,nxdim,ibl1,nn,no,ngbnew,
00083      &  nmatch,ifpmatch,nmatch_q,ifpmatch_q,m,ifpomat,nbln,ibln,ngb_in,nnr,igc2
00084       character(3) :: charnum3
00085       character(5) :: charnum5
00086       character(11):: filenamep
00087       integer(4),allocatable:: nx_r(:), ibl(:,:,:,:)
00088      &  ,imatcho(:),imatchn(:),imatcho_q(:),imatchn_q(:)
00089       real(8),allocatable:: prodmt(:,:,:,:),rdmatch(:,:,:,:)
00090       complex(8),allocatable:: ppmt(:,:,:,:),pmat(:,:),pomat(:,:),oon(:,:)
00091       complex(8):: pval,pslo,phasex
00092       real(8)::absqq,qqx(3), epsmx,aaaa
00093       integer(4):: nnmx ,ngcnn,ngbo
00094 cki      integer(4):: is_mix0vec ,ifgb0vec_a,ifgb0vec_b
00095       integer(4):: ifgb0vec_a,ifgb0vec_b , ifvcoud,idummy
00096       logical:: is_mix0vec,wvcc !,newaniso
00097       character(128):: vcoudfile
00098       real(8),allocatable:: wqfac(:),qbzwww(:,:)
00099       integer:: ifiwqfac,iqbz,iqbzx,nnn
00100       character(128) :: ixcc
00101 !!-------------------------------------------------------------
00102       call mpi__initialize()
00103       pi  = 4d0*datan(1d0)
00104       fpi = 4d0*pi
00105       if(mpi__root) write(6,"(' mode=0,3,202 (0 and 3 give the same results for given bas)' )")
00106 c      call readin5(imode,iqxini,iqxend)
00107       if( mpi__root ) then
00108          read(5,*) imode
00109       end if
00110       call mpi__broadcast(imode)
00111       write(ixcc,"('.mode=',i4.4)")imode
00112       call mpi__consoleout('hvccfp0'//trim(ixcc))
```

```
00113       call headver('hvccfp0: start',imode)
00114       call cputid(0)
00115       if(imode==202 ) then
00116         write(6,*)' hvccfp0: imode=',imode
00117 c      elseif(imode==101) then
00118 c        write(6,*)' hvccfp0: imode=',imode
00119 c        write(6,*)' remove_r0c is effective'
00120 c        write(6,*)' Generate VCCFP = VCCFP.ORG - new_VCCFP'
00121 c        ifvcfporg = iopen( "VCCFP.ORG",0,-1,0)
00122 c      elseif(imode==102) then
00123 c        write(6,*)' hvccfp0: imode=',imode
00124 c        write(6,*)' remove_r0c is effective'
00125       elseif(imode==0) then
00126       elseif(imode==3) then
00127       else
00128 Cstop2rx 2013.08.09 kino        stop 'hvccfp0: now hvccfp0 support just normal mode=0 3 202 101'
00129         call rx( 'hvccfp0: now hvccfp0 support just normal mode=0 3 202 101')
00130       endif
00131 c      if(iqxini< 2) paralellx0=.false.
00132 c      if(paralellx0) then
00133 c        write(6,"(' PARALELL.X0 mode: iqxini iqxend=',2i3)")
00134 c     &  iqxini, iqxend
00135 c      endif
00136
00137 C --- q, nqbz, alat, qlat, nbas, bas
00138       ifhvccfp = iopen('HVCCIN',0,-1,0)
00139       read (ifhvccfp) alat, plat,qlat, nqbz, nbas, nband
00140       if(allochk)
00141      & write(*,*) 'allocate(qbz(3,nqbz),bas(3,nbas),rmax(nbas))'
00142       allocate(qbz(3,nqbz),bas(3,nbas),rmax(nbas))
00143       read(ifhvccfp) qbz, bas,rmax
00144       read(ifhvccfp) nqibz
00145       if(allochk)
00146      &  write(*,*)'allocate(qibz(3,nqibz),iqibzx(nqbz))'
00147       allocate(qibz(3,nqibz),iqibzx(nqbz))
00148       read(ifhvccfp) qibz(1:3,1:nqibz)
00149       voltot = abs(alat**3*tripl(plat,plat(1,2),plat(1,3)))
00150       write(6,*)' voltot=',voltot
00151       write(6,*)
00152       write(6,"(i4,3f13.6)")(i,qibz(1:3,i),i=1,nqibz)
00153 c$$$!! Use instead of HVCCIN
00154 c$$$      call read_bzdata()
00155 c$$$      nwin   = 0                !Readin nw from NW file
00156 c$$$      incwfin= 0                !use ForX0 for core in GWIN
00157 c$$$      efin =  0d0               !readin EFERMI
00158 c$$$      call genallcf_v3(nwin,efin,incwfin) !in module m_genallcf_v3
00159 c$$$      call dinv33x (plat,qlat)
00160 c$$$      allocate(rmax(nbas))
00161 c$$$      voltot = abs(alat**3*tripl(plat,plat(1,2),plat(1,3)))
00162 c$$$      write(6,*)' voltot=',voltot
00163 c$$$      write(6,*)
00164 c$$$      write(6,"(i4,3f13.6)")(i,qibz(1:3,i),i=1,nqibz)
00165       is = iclose('HVCCIN')
00166
00167 c$$$!!! 9dec2012
00168 c$$$      ifiwqfac = iopen('WQFAC',0,-1,0)
00169 c$$$      read(ifiwqfac) nnn
00170 c$$$      write(6,*)'nnn nqbz=',nnn,nqbz
00171 c$$$c      if(nnn/=nqbz) stop 'hvccfp0_sc: readin nnn WQFAC/= nqbz'
00172 c$$$      allocate( wqfac(nnn),qbzwww(3,nnn) )
00173 c$$$      read(ifiwqfac) wqfac,qbzwww
00174 c$$$      ifiwqfac = iclose('WQFAC')
00175
00176 c$$$c-------------------------------------------------------
00177 c$$$      iqibzx =0
00178 c$$$      do iqibz=1,nqibz
00179 c$$$        do iq =1,nqbz
00180 c$$$          if( sum(abs(qibz(:,iqibz)-qbz(:,iq)))<1d-8 ) then
00181 c$$$            iqibzx(iq)=iqibz !iqibzx is the index for iqx in bz
00182 c$$$            goto 119
00183 c$$$          endif
00184 c$$$        enddo
00185 c$$$        stop " hvccfp: cannot find iqx"
00186 c$$$  119 enddo
00187 c$$$c --- Readin PLN. plane wave contributions 2000 May
00188 c$$$      ifplane = iopen('PLN',0,-1,0)
00189 c$$$      read (ifplane) ngpmx, ngcmx
00190 c$$$c q+G vector
00191 c$$$      if(allochk)
00192 c$$$     & write(*,*) 'allocate( ngcn(nqbz), ngvecci(3,ngcmx,nqibz))'
00193 c$$$      allocate( ngcn(nqbz), ngvecci(3,ngcmx,nqibz))
00194 c$$$      do iq=1, nqbz
00195 c$$$        read(ifplane) ngp, ngc
00196 c$$$        if(allochk)write(*,*)'allocate( geig, ngvecp, ngvecc)'
00197 c$$$        allocate( geig(ngp,nband), ngvecp(3,ngp), ngvecc(3,ngc))
00198 c$$$        read(ifplane) ngvecp, ngvecc, geig
00199 c$$$        if(iqibzx(iq) /=0) then
```

```
00200 c$$$          iqibz = iqibzx(iq)
00201 c$$$          ngcn(iqibz) = ngc
00202 c$$$          ngvecci(1:3,1:ngc,iqibz) = ngvecc(1:3,1:ngc)
00203 c$$$        endif
00204 c$$$        if(allochk)write(*,*) 'deallocate( geig, ngvecp, ngvecc)'
00205 c$$$        deallocate( geig, ngvecp, ngvecc)
00206 c$$$      enddo
00207 c-------------------------------------------------------
00208
00209 c q+G vector
00210 c$$$      ifiqgc = 1302
00211 c$$$      open(ifiqgc, file='QGcou',form='unformatted')
00212 c$$$      read(ifiqgc ) nqnumc, ngcmx, QpGcut_Cou
00213 c$$$      allocate( ngcn(nqbz), ngvecci(3,ngcmx,nqibz), ngvecc(3,ngcmx),iqok(nqibz))
00214 c$$$      iqok=1
00215 c$$$      do iq=1, nqnumc
00216 c$$$        read (ifiqgc) qqq, ngc
00217 c$$$        read (ifiqgc) ngvecc(1:3,1:ngc)
00218 c$$$        do iqibz=1,nqibz
00219 c$$$          if( sum(abs(qibz(:,iqibz)-qqq))<1d-8 ) then
00220 c$$$            ngcn(iqibz) = ngc
00221 c$$$            ngvecci(1:3,1:ngc,iqibz) = ngvecc(1:3,1:ngc)
00222 c$$$            iqok(iqibz)=0
00223 c$$$            exit
00224 c$$$          endif
00225 c$$$        enddo
00226 c$$$        if(sum(iqok)==0) exit
00227 c$$$      enddo
00228 c$$$      if(sum(iqok)/=0) stop 'hvccfp0: iqok/=0;wrong QGcou?'
00229 c$$$      deallocate(ngvecc,iqok)
00230 c$$$      close(ifiqgc)
00231
00232
00233       call readngmx('QGcou',ngcmx)
00234       allocate(ngvecc(3,ngcmx))
00235 c        allocate( ngcn(nqibz), ngvecci(3,ngcmx,nqibz),ngveccc0(3,ngcmx))
00236 c        do iqibz = 1,nqibz
00237 c          call readqg('QGcou',qibz(:,iqibz), ngcn(iqibz),ngvecci(1,1,iqibz))
00238 c        enddo
00239 c        call readqg('QGcou',(/0d0,0d0,0d0/), ngc0, ngvecc0(1,1))
00240 c        call releaseqg('QGcou')
00241
00242
00243
00244 c --- Readin BASFP//atom. The product basis functions.
00245       if(allochk)
00246      &  write(*,*)'allocte(lx,kmx,nblocha,nr,aa,bb,filename,ificrb'
00247       allocate(lx(nbas),kmx(nbas),nblocha(nbas),
00248      &         nr(nbas),aa(nbas),bb(nbas),filename(nbas),
00249      &         ificrb(nbas) )
00250
00251       do ibas = 1,nbas
00252         ic = ibas !
00253         filename(ibas)= 'BASFP'//char( 48+ic/10 )//char( 48+mod(ic,10))
00254         ificrb(ibas)  = iopen( filename(ibas),1,3,0)
00255         read(ificrb(ibas),"(4i6,2d24.16)")
00256      &    lx(ibas), kmx(ibas), nblocha(ibas), nr(ibas),aa(ibas),bb(ibas)
00257       enddo
00258       lxx = maxval(lx)
00259       if(allochk) write(*,*) 'allocate( nx(0:lxx,nbas) )'
00260       allocate( nx(0:lxx,nbas) )
00261       do ibas = 1,nbas
00262         read(ificrb(ibas),"(i5)") nx(0:lx(ibas),ibas)
00263       enddo
00264       nxx = maxval(nx)
00265       nrx = maxval(nr)
00266       if(allochk) write(*,*) 'allocate( rprodx(nrx,nxx,0:lxx,nbas) )'
00267       allocate( rprodx(nrx,nxx,0:lxx,nbas) )
00268
00269       do ibas = 1,nbas
00270         do l = 0, lx(ibas)
00271           do n = 1, nx(l,ibas)
00272             read(ificrb(ibas),"(3i5)"   ) k, kdummy,kdummy
00273             read(ificrb(ibas),"(d23.15)") (rprodx(i,n,l,ibas),i=1,nr(ibas))
00274 ccccccccccccc
00275 c       write(660+ibas,*)
00276 c       write(660+ibas,'(" *** nlibas=",3i3)')  n,l,ibas
00277 c       do i=1,nr(ibas)
00278 c        write(660+ibas,'(2d16.8)') bb(ibas)*( exp(aa(ibas)*(i-1))- 1d0),
00279 c      & rprodx(i,n,l,ibas)
00280 c       enddo
00281 ccccccccccccc
00282           enddo
00283         enddo
00284 c       isx = iclose(filename(ibas))
00285       enddo
00286
```

```
00287
00288 ccccccccccccccccccccccccccccccccccccccccccccc
00289 cccc TEST ccccccccccccccc
00290 c      open(117, file='xin')
00291 c      do i=1,nr(1)
00292 c        read(117,"(d24.16)") rprodx(i,1,0,1)
00293 c      enddo
00294 ccccccccccccccccccccccccccccccccccccccccccccc
00295
00296
00297
00298
00299
00300 ccccccccccccccccccccccccccccccccccccccccccccc
00301 c TEST ccccccccccccccccccccccccccccccccccccc
00302       if(besseltest) then
00303         write(6,*)
00304         write(6,*)
00305         write(6,*) ' *** TEST case ***  rprodx is given by Bessel.'
00306 ccc test G, corresponding <q+G|v|q+G> should be exact. e.g. ig1=1 and  ig1=35 for iqx=2
00307 ccc You can change these values for tests. ccccccccccc
00308         iqx  = 2
00309         igx1 = 1
00310         igx2 = 35
00311 c
00312         write(6,"(' iqx=',i3,' ig1 ig2=',2i3)") iqx,igx1,igx2
00313         write(6,"(a)")
00314      & ' <q+G|v|q+G> for the corresponding iqx ig1 ig2 should be exact!'
00315         write(6,"(a)") ' See fort.196'
00316         write(6,"(a)")
00317      & ' Errors will be from the radial function integrals !!!'
00318         write(6,"(a)") ' You can slso so similar test from hbasfp0.'
00319         write(6,"(a)") ' See test1 in basnfp0.'
00320 c
00321         if(allochk) write(*,*) 'deallocate(rprodx,nx)'
00322         deallocate(rprodx,nx)
00323         tpiba=8.d0*datan(1.d0)/alat
00324         lx = 4
00325         nr = nr(1)
00326         aa = aa(1)
00327         bb = bb(1)
00328         lxx = maxval(lx)
00329         if(allochk) write(*,*)'allocate( nx(0:lxx,nbas) )'
00330         allocate( nx(0:lxx,nbas) )
00331         kmx= 1
00332         nx = 2
00333         nxx = maxval(nx)
00334         nblocha= nxx *(lxx+1)**2
00335         nrx = maxval(nr)
00336         if(allochk) write(*,*)'allocate(rprodx,rofi ,phi,psi) '
00337         allocate(rprodx(nrx,nxx,0:lxx,nbas),rofit(nrx)
00338      &  ,phi(0:lxx),psi(0:lxx))
00339         rofit(1) = 0d0
00340         do ir   = 1, nrx
00341           rofit(ir) = bb(1)*( exp(aa(1)*(ir-1)) - 1d0)
00342         enddo
00343         do n = 1, nxx
00344           if(n==1) ig1 = igx1
00345           if(n==2) ig1 = igx2
00346           qg(1:3) =
00347      &    tpiba * (qibz(1:3,iqx)+ matmul(qlat, ngvecci(1:3,ig1,iqx)))
00348           absqg2  = sum(qg(1:3)**2)
00349 c
00350           do ir =1,nrx
00351             call bessl(absqg2*rofit(ir)**2,lxx,phi,psi)
00352             do ibas=1,nbas
00353               do l = 0, lx(ibas)
00354                 rprodx(ir,n,l,ibas) = phi(l)* rofit(ir) **(l +1 )
00355               enddo
00356             enddo
00357           enddo
00358         enddo
00359 c --- orthogonalized rprodx.
00360         do ibas=1,nbas
00361           do l = 0, lx(ibas)
00362             rprodx(1:nr(ibas),l,l,ibas)=
00363      &     rprodx(1:nr(ibas),1,l,ibas)
00364      &   + rprodx(1:nr(ibas),2,l,ibas)
00365             n = 1
00366             call gintxx(rprodx(1,n,l,ibas),rprodx(1,n,l,ibas)
00367      &    ,aa(ibas),bb(ibas),nr(ibas), aaa )
00368             aaa = 1d0/sqrt(aaa)
00369             rprodx(1:nr(ibas),n,l,ibas)= aaa*rprodx(1:nr(ibas),n,l,ibas)
00370             if(nxx==1) cycle
00371             n1=1
00372             n2=2
00373             call gintxx(rprodx(1,n1,l,ibas),rprodx(1,n2,l,ibas)
```

```
00374      &    ,aa(ibas),bb(ibas),nr(ibas), aaa12 )
00375             rprodx(1:nr(ibas),n2,l,ibas) = rprodx(1:nr(ibas),n2,l,ibas)
00376      &      - aaa12*rprodx(1:nr(ibas),n1,l,ibas)
00377             n = 2
00378             call gintxx(rprodx(1,n,l,ibas),rprodx(1,n,l,ibas)
00379      &    ,aa(ibas),bb(ibas),nr(ibas), aaa )
00380             aaa = 1d0/sqrt(aaa)
00381             rprodx(1:nr(ibas),n,l,ibas)= aaa*rprodx(1:nr(ibas),n,l,ibas)
00382           enddo
00383         enddo
00384       endif
00385 cccc TEST end ccccccccccccccccccccccccccccc
00386 ccccccccccccccccccccccccccccccccccccccccccccc
00387
00388
00389       nbloch    = sum(nblocha)
00390       nblochpmx = nbloch + ngcmx
00391
00392 c --- CG coefficienets. <LM3|lm1 lm2>
00393 c inxcg = lm1(lm1-1)/2 + lm2 (lm1>lm2)
00394 c Injcg = indxcg(inxcg) to indxcg(inxcg)-1
00395 c cg(inxcg)  : = <lm3|lm1 lm2>
00396 c jcg(lnjcg) : = lm3
00397       lmxcg = lxx
00398
00399       call scg_sizechk(lmxcg,lnjcg,lnxcg) !(lmax,c,cindx,js)
00400       write(6,*)'scg_sizechk= ',lnjcg,lnxcg
00401 c      if (lmxcg .le. 6) then
00402 c        lnjcg = 6500
00403 c        lnxcg = 1300
00404 c      else if (lmxcg .le. 8) then
00405 c        lnjcg = 22700
00406 c        lnxcg = 3400
00407 c      else if (lmxcg .le. 10) then
00408 c        lnjcg = 62200
00409 c        lnxcg = 7400
00410 c      else
00411 c        call rxi('setcg: cannot handle lmxcg=',lmxcg)
00412 c      endif
00413 c      if(allochk)
00414 c     &  write(*,*) 'allocate(cg(lnjcg),jcg(lnjcg),indxcg(lnxcg))'
00415       allocate(cg(lnjcg),jcg(lnjcg),indxcg(lnxcg))
00416       call scg(lmxcg,cg,indxcg,jcg)
00417       if(allochk) write(6,*)' end of scg: cg coefficients generated.'
00418
00419
00420       call dinv33(qlat,0,ginv,det)
00421
00422 c --- Get real-space vectors and reciprocal-space vectors for Ewald sum.
00423 C defaults values for ewald sum
00424 c      call lattc(awald0,tol,alat,alat,plat0,gx,gy,gz,gam,plat,qlat,
00425 c     .   lmxst,vol,awald,w(odlv),nkd,w(oqlv),nkq,nkdmx,nkqmx,w(owork))
00426 c- taken from lattc.f
00427
00428 c default values ok?
00429       awald0 = 2d0   !See p_lat_0
00430       tol    = 1d-9
00431       nkdmx  = 800
00432       nkqmx  = 800
00433       lmax   = 2*lxx  !lxx or lmax=6 ???
00434
00435       vol0= abs(tripl(plat,plat(1,2),plat(1,3)))
00436       as   = awald0
00437 c      alat0= alat
00438       alat1= alat
00439 c      if(alat1.le.0.5d0) alat1=alat
00440       tpiba=8.d0*datan(1.d0)/alat
00441       call cross_x(plat(1,2),plat(1,3),qb0)
00442       call cross_x(plat(1,3),plat(1,1),qb0(1,2))
00443       call cross_x(plat(1,1),plat(1,2),qb0(1,3))
00444       qb0(1:3,1:3) = qb0(1:3,1:3)/vol0
00445
00446       rdist0=vol0**(1.d0/3.d0)
00447       qdist0=1.d0/rdist0
00448       radd=.7*rdist0
00449       qadd=.7*qdist0
00450       a0=as/rdist0
00451       awald=a0/alat
00452 ccccccccccccccccccccccccccccccccc
00453 c takao
00454 c      tol1= tol*alat**(lmax+1) *0.01
00455 ccccccccccccccccccccccccccccccccc
00456       tol1= tol*alat**(lmax+1)
00457       if(allochk) write(*,*) 'allocate(dlv, qlv, work) '
00458       allocate(dlv(3,nkdmx), qlv(3,nkqmx), work(max0(nkdmx,nkqmx)) )
00459       call lctoff(a0,vol0,lmax,tol1,r0,q0)
00460       nkdest =4.18879*(r0+radd)**3/vol0+.5
```

```
00461       nkrest =4.18879*(q0+qadd)**3*vol0+.5
00462       write(6,340) as,tol,lmax,awald,vol0,alat1,nkdest,nkrest
00463   340 format(/' lattc:  as=',f6.3,'   tol=',1p,e8.2,'   lmax=',i2,
00464      .  '   awald=',0p,f7.4,'   v0=',f10.3/' alat1=',f9.5,
00465      .  '   estimates:   nkd',i6,'   nkr',i6)
00466       call lgen(plat,r0+radd,nkd,nkdmx,dlv,work)
00467       write(6,342) r0,r0*alat,radd,nkd
00468   342 format('  r0=',f9.4,'   rc=',f9.4,'   radd=',f9.4,'   nkd=', i7)
00469       call lgen(qb0,q0+qadd,nkq,nkqmx,qlv,work)
00470       write(6,341) q0,q0*tpiba,qadd,nkq
00471   341 format('  q0=',f9.4,'   qc=',f9.4,'   qadd=',f9.4,'   nkr=', i7)
00472       if(allochk) write(*,*) 'deallocate(work)'
00473       deallocate(work)
00474
00475 C... readin r0c
00476 c      if(newaniso()) then
00477         eee=screenfac() !takao feb2012
00478 c      elseif(imode==101.or.imode==102) then
00479 c        eee = eees()
00480 c      else
00481 c        eee=0d0
00482 c      endif
00483
00484 !! for eps_lmf and epsPP_lmf mode,
00485 !! even the small eee=1d-4 can affect to dielectric function near q=0 when its values is large as
     one-hundred or more.
00486 !! Thus we set eee=0d0 to avoid this.
00487       if(imode==202) then !
00488          eee=0d0
00489       endif
00490
00491       write(6,"(' Coulomb is exp(sqrt(-eee)*r)/r. eee=',d13.6,d13.6)")eee
00492
00493 C--- bessel and hankel for the expansion of exp(-r/r_0)/r.
00494 c bessel and hankel is renomarized so that its behaves as r^l and r^{-l-1} near r=0.
00495 c  rkpr means r^l*r for e=0 (r0c =infinity) case
00496       allocate(rkpr(nrx,0:lxx,nbas),rkmr(nrx,0:lxx,nbas),rofi(nrx,nbas))
00497       do ibas=1,nbas
00498         call genjh(eee,nr(ibas),aa(ibas),bb(ibas),lx(ibas), nrx,lxx,
00499      o     rofi(1,ibas), rkpr(1,0,ibas), rkmr(1,0,ibas))
00500       enddo
00501
00502 C--- onsite integrals <j(e=0)|B> and <B|v(onsite)|B>
00503 cc      if(allochkw) write(*,*) ' allocate  rojb, sgbb '
00504 c      allocate( rojb(nxx, 0:lxx, nbas), sgbb(nxx,  nxx,  0:lxx, nbas))
00505 c      do ibas = 1,nbas
00506 c        call mkjb( lxx, lx(ibas),nxx, nx(0:lxx,ibas),
00507 c     i                 aa(ibas),bb(ibas), nr(ibas), nrx,
00508 c     i                 rprodx(1,1,0,ibas),
00509 c     o        rojb(1,0,ibas), sgbb(1,1,0,ibas))
00510 c      enddo
00511       allocate( rojb(nxx, 0:lxx, nbas), sgbb(nxx,  nxx,  0:lxx, nbas))
00512       do ibas = 1,nbas
00513         call mkjb_4( lxx, lx(ibas),nxx, nx(0:lxx,ibas),
00514      i                aa(ibas),bb(ibas), nr(ibas), nrx,
00515      i                rprodx(1,1,0,ibas),
00516      i     rofi(1,ibas), rkpr(1,0,ibas), rkmr(1,0,ibas),
00517      o        rojb(1,0,ibas), sgbb(1,1,0,ibas))
00518       enddo
00519
00520 c----------------
00521 C--- coulomb matrix for each q = qibz
00522 c----------------
00523       nlxx= (lxx+1)**2
00524 c      ngb = nbloch + ngcn(1)
00525       allocate(ngvecc0(3,ngcmx))
00526       call readqg('QGcou',(/0d0,0d0,0d0/),ginv,  quu,ngc0, ngvecc0)
00527       deallocate(ngvecc0)
00528       ngb = nbloch + ngc0
00529       if(allochk) write(*,*) 'allocate( vcoul)'
00530       allocate( vcoul(nblochpmx,nblochpmx) )
00531 c      if(imode==101) allocate( vcoul_org(nblochpmx,nblochpmx) )
00532
00533       vcoul  = 0d0
00534
00535 C... q near zero
00536       write(6,*) '--- readin Q0P -------'
00537       open (101,file='Q0P')
00538       read (101,"(i5)") nq0i
00539       if(allochk) write(*,*)'allocate( wqt(1:nq0i),q0i(1:3,1:nq0i) )'
00540       allocate( wqt(1:nq0i),q0i(1:3,1:nq0i) )
00541       read (101,"(d24.16,3x, 3d24.16)" )( wqt(i),q0i(1:3,i),i=1,nq0i)
00542       write (6,"(d13.5,3x, 3d13.5)" )( wqt(i),q0i(1:3,i),i=1,nq0i)
00543       close(101)
00544
00545       write(6,*) ' *** goto do iq nqibz nq0i=',nqibz,nq0i
00546
```

```
00547 C --- Check PARALELL.X0
00548 c      INQUIRE (FILE = 'PARALELL.X0', EXIST = paralellx0)
00549 c$$$      wvcc=.true.
00550 c$$$      if(newaniso()) wvcc=.false.
00551       wvcc=.false.
00552       write(6,'(a)') " Mix0vec.XXX is not empty only when"
00553      &   //" the corresponding q is in Q0P with zero weight."
00554 c      if(paralellx0) then
00555 c        if(wvcc) ifvcfpout = iopen( "VCCFP." //xxt(iqxini,iqxend),0,-1,0)
00556 c        ifgb0vec = iopen ( "Mix0vec."//xxt(iqxini,iqxend),1,3,0)
00557 c        ifgb0vec1 = iopen ( "Mix0vec1."//xxt(iqxini,iqxend),1,3,0)
00558 c      else
00559         iqxend = nqibz + nq0i
00560         if(wvcc) ifvcfpout = iopen('VCCFP',0,-1,0)
00561         ifgb0vec = iopen( "Mix0vec",1,3,0)
00562         ifgb0vec1 = iopen( "Mix0vec1",1,3,0)
00563 c      endif
00564
00565       if(imode==202) then
00566         iqxini= nqibz + 1
00567 c      elseif(paralellx0) then
00568 c     &  !skip
00569 c$$$      elseif(bzcase()==1) then
00570 c$$$!        iqxini = 2
00571 c$$$        iqxini = 1 !oct2005
00572       else
00573         iqxini = 1
00574       endif
00575 !!
00576 c      if(newaniso().and.imode==0) then
00577       if(imode==0) then
00578         iqxini=1
00579 c        iqxend=nqibz ! comment out at 18nov2012
00580       endif
00581       write(6,*)'iqxini iqxend=',iqxini,iqxend
00582 c qibz loop
00583 c      epsx =  0.01d0
00584 c      if(bzcase()==1) then
00585         if(abs(sum(qibz(:,1)**2))/=0d0) call rx( 'hvccfp0: sum(q**2)==0d0')
00586 c      endif
00587       if(wvcc) write(ifvcfpout) nqbz, nblochpmx
00588 c      if(imode==101) then
00589 c        read(ifvcfporg) nqbz_in, nblochpmx_in
00590 c        if(nqbz /= nqbz_in) stop 'nqbz /= nqbz_in VCCFP.ORG'
00591 c        if(nblochpmx /= nblochpmx_in)
00592 c     &     stop 'nblochpmx /= nblochpmx_in VCCFP.ORG'
00593 c      endif
00594
00595
00596
00597 C... Readin PRODMT into prodmt. oct2005
00598       smbb = smbasis()
00599       write(6,*) ' smooth mixed basis=',smbb
00600       if(smbasis()) then
00601         allocate( prodmt(2,nxx,0:lxx,nbas))
00602         allocate( nx_r(0:lxx))
00603         do ibas =1,nbas
00604           filenamep = 'PRODMT_'//charnum3(ibas)
00605           ifprodmt  = iopen(filenamep,0,-1,0)
00606           read(ifprodmt) nl_r
00607           if( 2*(nl_r-1) /= lxx ) then
00608             write(6,*) 2*(nl_r-1),lxx
00609 Cstop2rx 2013.08.09 kino            stop '2*nl_r-1 /= lxx '
00610             call rx( '2*nl_r-1 /= lxx ')
00611           endif
00612           read(ifprodmt) nxx_r
00613           write(6,"(' nxx =',100i3)")nxx_r
00614 Cstop2rx 2013.08.09 kino          if(nxx_r>nxx) stop 'nxx_r>nxx'
00615           if(nxx_r>nxx) call rx( 'nxx_r>nxx')
00616           read(ifprodmt) nx_r(0:lxx)
00617           write(6,"(' nx_r=',100i3)") nx_r(0:lxx)
00618           lx_ = lx(ibas)
00619           if(sum(abs(nx(0:lx_,ibas)-nx_r(0:lx_))) /=0) then
00620             write(6,*)' debug: nx  =',nx(0:lx_,ibas)
00621             write(6,*)' debug: nx_r=',nx_r(0:lx_)
00622 Cstop2rx 2013.08.09 kino            stop 'nx /=nx_r'
00623             call rx( 'nx /=nx_r')
00624           endif
00625           read(ifprodmt) prodmt(1:2, 1:nxx_r, 0:lxx, ibas)
00626           write(6,*)' sumcheck prodmt=',sum(abs(prodmt(:,:,:,ibas)))
00627           isx = iclose(filenamep)
00628         enddo
00629
00630
00631 ccccccccccccccccccccccccccccccccccccc
00632 C... Check write for radial part of the product basis
00633         if(.false.) then
```

```
00634           do ibas= 1,1 !1,nbas
00635             do l   =  0,lx(ibas)
00636               open(1011,file='ProdOld_ibas'//charnum3(ibas)//'_l'//charnum3(l))
00637 c         open(2011,file='ProdNew_ibas'//charnum3(ibas)//'_l'//charnum3(l))
00638               nxdim = nx(l,ibas)
00639               do ix=1,nxdim
00640                 write(1011,"(' -- -- -- ',3i3,' --- ' )") ix,l,ibas
00641 c         write(2011,"(' -- -- -- ',3i3,' --- ' )") ix,l,ibas
00642                 do ir =1,nr(ibas)
00643                   write(1011,"(d13.5,2x,2d18.8)")
00644      &    rofi(ir,ibas), rprodx(ir,ix,l,ibas)
00645      &    , rprodx(ir,ix,l,ibas) /rofi(ir,ibas)
00646 c          write(2011,"(d13.5,2x,2d18.8)")
00647 c     &    rofi(ir,ibas), sum(rprodx(ir,1:nxdim,l,ibas)*rdmatch(1:nxdim,ix,l,ibas))
00648 c     &    , sum(rprodx(ir,1:nxdim,l,ibas)*rdmatch(1:nxdim,ix,l,ibas))/rofi(ir,ibas)
00649                 enddo
00650               enddo
00651               close(1011)
00652 c         close(2011)
00653             enddo
00654           enddo
00655 c       stop 'text end'
00656         endif
00657 ccccccccccccccccccccccccccccccccccccccccccccccccc
00658 !
00659         allocate( rdmatch(nxx,nxx,0:lxx,nbas) )
00660         do ibas= 1, nbas
00661           do l   = 0, lx(ibas)
00662             nxdim = nx(l,ibas)
00663             if(nxdim<=1)write(6,*)'hvccfp0:smbasis case error nxdim <=1'
00664 !       pval  = prodmt(1, 1:nxdim, l,ibas)
00665 !       pslo  = prodmt(2, 1:nxdim, l,ibas)
00666 !       prod(r, inew) = \sum_iold rrmat(inew,iold) * prod(r,iold)
00667             write(6,"('goto mkradmatch ibas lnxdim =',3i4)")ibas,l,nxdim
00668             call mkradmatch(prodmt(1:2, 1:nxdim, l,ibas), nxdim,
00669      o           rdmatch(1:nxdim,1:nxdim,l,ibas) )
00670           enddo
00671         enddo
00672
00673
00674
00675 ! index (mx,nx,lx,ibas) ordering: taken from voul_4
00676         allocate(ibl(-lxx:lxx,nxx,0:lxx,nbas))
00677         ibl1 = 0
00678         ibl=999999
00679         do ibas= 1, nbas
00680           do l   = 0, lx(ibas)
00681             do n   = 1, nx(l,ibas)
00682               do m   = -l, l
00683                 ibl1  = ibl1 + 1
00684                 ibl(m,n,l,ibas) = ibl1
00685 !       write(6,*)ibl1,n,l,m,lmbl(ibl1)
00686               enddo
00687             enddo
00688           enddo
00689         enddo
00690         if(ibl1/= nbloch) then
00691           write(6,*)' ibl1 nbloch',ibl1, nbloch
00692 Cstop2rx 2013.08.09 kino          stop ' hvccfp0:smbasis mode  error ibl1/= nbloch'
00693           call rx( ' hvccfp0:smbasis mode  error ibl1/= nbloch')
00694         endif
00695 ! index (mx,nx,lx,ibas) ordering
00696 ctttt
00697         nnr = 2 ! =2 new
00698         ! =0 equivalence with original mixed basis
00699         write(6,*)' sss:nbas lx=',nbas,lx(1:nbas)
00700
00701         nbln=0
00702         do ibas= 1, nbas
00703           do l   = 0, lx(ibas)
00704             write(6,"('sss: nx=',3i4)") ibas,l,nx(l,ibas)
00705             if(nx(l,ibas)<=0) cycle
00706 Cstop2rx 2013.08.09 kino            if(nx(l,ibas)==1) stop 'nx(l,ibas) =1'
00707             if(nx(l,ibas)==1) call rx( 'nx(l,ibas) =1')
00708 ccccccccccccccd
00709 ctttt
00710 c         nnr = 2 ! =2 new
00711 c             ! =0 equivalence with original mixed basis
00712 c         if(l<=3) nnr=0
00713 ccccccccccccccccc
00714             nbln = nbln + (2*l+1)*(nx(l,ibas)-nnr)
00715           enddo
00716         enddo
00717         allocate( pmat(nbloch+ngcmx, nbln+ngcmx) )
00718         pmat=0d0
00719         ibln = 0
00720         do ibas= 1, nbas
```

```
00721           do l    = 0, lx(ibas)
00722 ccccccccccccccccccc
00723 ctttt
00724 c          nnr = 2 ! =2 new
00725 c                  ! =0 equivalence with original mixed basis
00726 c          if(l<=3) nnr=0
00727 ccccccccccccccccc
00728             do nn  = nnr+1, nx(l,ibas) !nn=1 and nn=2 corresponds to non-zero val sol
00729               do m   = -l, l
00730                 ibln = ibln +1
00731                 nxdim = nx(l,ibas)
00732                 pmat( ibl(m,1:nxdim,l,ibas), ibln)
00733      &     =  rdmatch(1:nxdim, nn, l,ibas)
00734 ctttt
00735 c           pmat( ibl(m,nn,l,ibas), ibln)
00736 c     &     =  1d0
00737 ccccccccccccccccccccccccccccccccccccccccccccccccc
00738               enddo
00739             enddo
00740           enddo
00741         enddo
00742 C... Store matting matrix (imatchn,imatcho,pmatch)
00743         ifpomat   = iopen('POmat',0,-1,0)
00744 c       write(6,*)'ttt= sumchk pmat(b)=',sum(abs(pmat(1:nbloch, 1:nbln)))
00745       endif
00746 !! === open file Vcoud ===
00747 !! This contains E(\nu,I), given in PRB81,125102
00748
00749 !! == main loop for iqx ==
00750       call mpi__getrange( mpi__iini, mpi__iend, iqxini, iqxend )
00751       do 1001 iqx = mpi__iini, mpi__iend ! q=(0,0,0) is omitted!
00752 c$$$      do 1001 iqx = iqxini, iqxend ! q in IBZ. avoid q=0 case for iqx=1
00753         write(6,"('#### do 1001 start iqx=',i5)")iqx
00754         vcoudfile='Vcoud.'//charnum5(iqx)  !this is closed at the end of do 1001
00755         ifvcoud = iopen(trim(vcoudfile),0,-1,0)
00756         if(iqx > nqibz) then  !       iq = 1
00757           q  = q0i(:,iqx-nqibz)
00758 c          qq = 0d0
00759         else                  !       iq = iqx
00760           q  = qibz(:,iqx)
00761 c          qq = q
00762         endif
00763 ccccccccccccccccccccc
00764 c        if(imode==202) then !for iqx>nqibz
00765 c          qq=q
00766 c        endif
00767 ccccccccccccccccccccc
00768
00769 c$$$        if(.not. newaniso() ) then !this is for fe_epsPP_lmfh_chipm feb2012
00770 c$$$          if(sum(q**2)<1d-12) q=(/1d-4,0d0,0d0/) !takao oct2006
00771 c$$$        endif
00772 !! ==== q+G vector ====
00773         call readqg('QGcou',q,ginv,  quu,ngc, ngvecc ) !qq-->q
00774         ngb = nbloch + ngc  !it was ngcnn(iq)
00775         write(6,'(" iqx q ngc =",i5,3f10.4,i5)') iqx,q,ngc
00776
00777 c$$$        if(newaniso()) then
00778 c$$$          continue
00779 c$$$        elseif(bzcase()==1.and.iqx==1) then
00780 c$$$          goto 1101
00781 c$$$        endif
00782
00783 c        ngc = ngcn(iq)
00784 c        ngvecc(1:3,1:ngc) = ngvecci(1:3,1:ngc,iq)
00785 c        write(6,*)' iq ngc=',iq, ngc
00786 ccccccccccccccccccccccccccccccccccccccccccccccccc
00787 c  q test
00788 c       q=(/ 0.09d0,0.09d0,0.09d0/)
00789 c      q = q+(/ 0.01d0,0.01d0,0.01d0/)
00790 c        q=q/4
00791 ccccccccccccccccccccccccccccccccccccccccccccccccc
00792
00793 C--- strxq structure factor.
00794         if(allochk) write(6,*)' goto strxq'
00795         if(allochk) write(*,*) 'allocate( strx(nlxx,nbas,nlxx,nbas))'
00796         allocate( strx(nlxx,nbas,nlxx,nbas))
00797         do ibas1 =1,nbas
00798           do ibas2 =1,nbas
00799             p = bas(:,ibas2)-bas(:,ibas1)
00800             phasep =exp(img*2*pi*sum(q*p))
00801             nlx1 = (lx(ibas1)+1)**2
00802             nlx2 = (lx(ibas2)+1)**2
00803             if(allochk) write(*,*) 'allocate( s(nlx1,nlx2))'
00804             allocate( s(nlx1,nlx2),sd(nlx1,nlx2)) !kino add sd----but sd is dummy
00805 c           call strxq(1,0d0,q,p,nlx1,nlx2,nlx1,alat,voltot,
00806             call strxq(1,eee,q,p,nlx1,nlx2,nlx1,alat,voltot,
00807      i        awald,nkd,nkq,dlv,qlv,
```

```
00808      i        cg,indxcg,jcg,
00809      o        s,sd)
00810             strx(1:nlx1,ibas1,1:nlx2,ibas2) = fpi*s      !!! *phasep
00811             if(allochk) write(*,*)'deallocate( s )'
00812             deallocate( s,sd )
00813           enddo
00814         enddo
00815 ccccccccccccccccccccccccc
00816 c        strx=0d0
00817 ccccccccccccccccccccccccc
00818
00819 C--- onsite integrals <j(e=0)|P^(q+G)_L> and <B|v(onsite)|B>
00820 c$$$      if(.true.) then !==New version without sgpp and fouvp allocation June2004=====
00821         if(allochk) write(*,*)'allocate(rojp,sgpb,fouvb)'
00822         allocate(  rojp(ngc,      nlxx, nbas),
00823      &            sgpb(ngc, nxx, nlxx, nbas),
00824      &           fouvb(ngc, nxx, nlxx, nbas))
00825 c     &            sgpp(ngc, ngc, nlxx, nbas),
00826 c     &           fouvp(ngc, ngc, nlxx, nbas) )
00827         do ibas = 1,nbas
00828           if(allochk) write(6,*)' --- goto mkjp_4',ibas
00829           call mkjp_4(q,ngc, ngvecc, alat, qlat,
00830      i      lxx, lx(ibas),nxx, nx(0:lxx,ibas),
00831      i      bas(1,ibas),aa(ibas),bb(ibas),rmax(ibas),
00832      i      nr(ibas), nrx, rprodx(1,1,0,ibas),
00833      i   eee, rofi(1,ibas), rkpr(1,0,ibas), rkmr(1,0,ibas),
00834      o      rojp(1,1,ibas),  sgpb(1,1,1,ibas),
00835      o      fouvb(1,1,1,ibas))
00836 c          call mkjp3(q,ngc, ngvecc, alat, qlat,
00837 c     i      lxx, lx(ibas),nxx, nx(0:lxx,ibas),
00838 c     i      bas(1,ibas),aa(ibas),bb(ibas),rmax(ibas),
00839 c     i      nr(ibas), nrx, rprodx(1,1,0,ibas),
00840 c     o      rojp(1,1,ibas),  sgpb(1,1,1,ibas),
00841 c     o      fouvb(1,1,1,ibas))
00842         enddo
00843
00844 C--- the Coulomb matrix
00845         if(allochk) write(6,*)' goto vcoulq_4'
00846         call vcoulq_4(q, nbloch, ngc,
00847      i                nbas, lx,lxx, nx,nxx,
00848      i                alat, qlat, voltot, ngvecc,
00849      i       strx, rojp,rojb, sgbb,sgpb, fouvb, !sgpp,fouvp,
00850      i       nblochpmx, bas,rmax,
00851      i    eee, aa,bb,nr,nrx,rkpr,rkmr,rofi,
00852      o         vcoul)
00853 c       call vcoulq2(q, nbloch, ngc,
00854 c     i                 nbas, lx,lxx, nx,nxx,
00855 c     i                 alat, qlat, voltot, ngvecc,
00856 c     i       strx, rojp,rojb, sgbb,sgpb, fouvb, !sgpp,fouvp,
00857 c     i       nblochpmx, bas,rmax,
00858 c     o          vcoul)
00859         if(allochk) write(6,*)' end of vcoulq_4'
00860         deallocate( strx, rojp,sgpb,fouvb)
00861
00862 c$$$      else !===old version (allocation of sgpp and fouvp are required) ====
00863 c$$$
00864 c$$$       if(allochk) write(*,*) 'allocate(rojp,sgpb,sgpp,fouvb,fouvp)'
00865 c$$$       allocate( rojp(ngc,      nlxx, nbas),
00866 c$$$     &            sgpb(ngc, nxx, nlxx, nbas),
00867 c$$$     &           fouvb(ngc, nxx, nlxx, nbas),
00868 c$$$     &            sgpp(ngc, ngc, nlxx, nbas),
00869 c$$$     &           fouvp(ngc, ngc, nlxx, nbas) )
00870 c$$$       do ibas = 1,nbas
00871 c$$$          write(6,*)' xxx goto mkjp',ibas
00872 c$$$          call mkjp2(q,ngc, ngvecc, alat, qlat,
00873 c$$$     i      lxx, lx(ibas),nxx, nx(0:lxx,ibas),
00874 c$$$     i      bas(1,ibas),aa(ibas),bb(ibas),rmax(ibas),
00875 c$$$     i      nr(ibas), nrx, rprodx(1,1,0,ibas),
00876 c$$$     o      rojp(1,1,ibas),  sgpb(1,1,1,ibas),
00877 c$$$     o      fouvb(1,1,1,ibas),
00878 c$$$     o      sgpp(1,1,1,ibas),fouvp(1,1,1,ibas) )
00879 c$$$       enddo
00880 c$$$c--- the Coulomb matrix
00881 c$$$       write(6,*)' goto vcoulq'
00882 c$$$       call vcoulq(q, nbloch, ngc,
00883 c$$$     i                 nbas, lx,lxx, nx,nxx,
00884 c$$$     i                 alat, qlat, voltot, ngvecc,
00885 c$$$     i          strx, rojp,rojb, sgbb,sgpb,sgpp, fouvb,fouvp, nblochpmx,
00886 c$$$     o          vcoul)
00887 c$$$       if(allochk)
00888 c$$$     &   write(*,*)'deallocate(strx, rojp,sgpb,sgpp, fouvb,fouvp)'
00889 c$$$       deallocate( strx, rojp,sgpb,sgpp, fouvb,fouvp)
00890 c$$$
00891 c$$$      endif !============================================================
00892
00893 c----check write
00894         trwv = 0d0
```

```
00895         do i = 1,nbloch
00896           trwv = trwv + vcoul(i,i)
00897         enddo
00898         write(6,'(" vcoul trwi=",i6,2d22.14)') iqx,trwv
00899         write(6,'("### sum vcoul(1:ngb,      1:ngb) ",2d22.14,2x,d22.14)')
00900      & sum(vcoul(1:ngb,1:ngb)), sum(abs(vcoul(1:ngb,1:ngb)))
00901         write(6,'("### sum vcoul(1:nbloch,1:nbloch) ",2d22.14,2x,d22.14)')
00902      & sum(vcoul(1:nbloch,1:nbloch)),sum(abs(vcoul(1:nbloch,1:nbloch)))
00903         write(6,*)
00904 ccccccccccccccccccccccccccccc
00905 c      vcoul(:, nbloch+1:ngb)=0d0
00906 c      vcoul(nbloch+1:ngb,:)=0d0
00907 ccccccccccccccccccccccccccccc
00908
00909  1101   continue
00910         ngbo=ngb
00911 C... Generate ppmt mattix oct2005 ......................
00912         if(smbasis()) then
00913           allocate( ppmt(2,(lxx+1)**2,nbas,ngc) )
00914           ppmt = 0d0
00915           call mkppmt(alat,plat,qlat, q,
00916      i    ngc, ngvecc,
00917      i    rmax, nbas,  bas, lx, lxx,
00918      o    ppmt) ! ppmt contains value and slove of e(i q+G r) at MT boundaries.
00919           ! ppmt(2,lmxaa,nbas)
00920 ccccccccccccccccccccccccccccccccccccccccccccccccccc
00921 c        write(6,*) 'lxx ppmtsum=',lxx, sum(abs(ppmt))
00922           write(6,*) 'nbln ngc',nbln,ngc
00923 ccccccccccccccccccccccccccccccccccccccccccccccccccc
00924
00925 C... Matching matrix pmtch. ppmt and prodmt
00926           pmat(:, nbln+1:nbln+ngc)=0d0
00927 c        write(6,*) 'sss nbln ngc',nbln,ngc
00928           do igc=1,ngc
00929 c          write(6,*) 'igc=',igc
00930             pmat(nbloch+igc, nbln+igc) = 1d0
00931             do ibas= 1, nbas
00932               do l  =  0, lx(ibas)
00933                 do m  = -l, l
00934 c               write(6,*) 'ibas l m=',ibas,l,m
00935                   pval= ppmt(1, l**2 + l+1 +m, ibas,igc)
00936                   pslo= ppmt(2, l**2 + l+1 +m, ibas,igc)
00937                   do n = 1,nx(l,ibas)
00938                     if(n==1.and.debug) write(6,"('ttt2: ')")
00939                     pmat(ibl(m,n,l,ibas), nbln+igc)
00940      &          =  rdmatch(n,l,l,ibas) * pval
00941      &          +  rdmatch(n,2,l,ibas) * pslo
00942                     if(debug.and.abs(pmat(ibl(m,n,l,ibas), nbln+igc))/=0d0)
00943      &         write(6,"('ttt2: i1 i2 pmat=',2i5,2d13.5)")
00944      &         ibl(m,n,l,ibas), nbln+igc, pmat(ibl(m,n,l,ibas), nbln+igc)
00945                   enddo
00946                 enddo
00947               enddo
00948             enddo
00949           enddo
00950           deallocate(ppmt)
00951           nn = nbln  +ngc ! number for new smooth mixed basis.
00952           no = nbloch+ngc ! number for original size of mixed basis.
00953           if(debug) write(6,*) 'end of pmat'
00954
00955 C... oo(no,no). The original overlap matrix.
00956           allocate( pomat(nn,no) )
00957           allocate( ppovl(ngc,ngc),oo(no,no))
00958           call mkppovl2(alat,plat,qlat,
00959      i          ngc,  ngvecc,
00960      i          ngc,  ngvecc,
00961      i          nbas, rmax, bas,
00962      o          ppovl)
00963           oo = 0d0
00964           do ipl1 = 1,nbloch
00965             oo(ipl1,ipl1) = 1d0
00966           enddo
00967           do ix= 1,ngc
00968             do iy= 1,ngc
00969               oo(nbloch+ix, nbloch+iy) = ppovl(ix,iy)
00970             enddo
00971           enddo
00972           if(debug) write(6,*) 'end of oo'
00973
00974
00975 C... oon(nn,nn) is the overlap matrix with new basis
00976           allocate(oon(nn,nn))
00977           oon = matmul( dconjg(transpose(pmat(1:no,1:nn)))
00978      &               ,matmul(oo,pmat(1:no,1:nn)) )
00979
00980
00981 ccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
00982 c Reduction of pmat by SVD ... not meaningful
00983 c      nnmx = 1000000
00984 c      epsmx= 1D20
00985 cc      write(6,*)' sumchk pmat=',sum(abs(pmat(1:no,1:nn)))
00986 c      call  zgesvdnn2(nn,nn, nnmx,epsmx,
00987 c     o   oon, ! pmat is reduced to pmat(1:no,1:nnn) by SVD.
00988 c     o   ngcnn)
00989 cc      call  zgesvdnn2(no,ngc, nnmx,epsmx,
00990 cc     i   pmat(1:no,nbln+1:nbln+ngc), ! pmat is reduced to pmat(1:no,1:nnn) by SVD.
00991 cc     o   ngcnn)
00992 cc      nn= nbln+ngcnn
00993 c      write(6,*)' svd ngc ngcnn=',ngc, ngcnn
00994 c      stop 'test end xxxxxx'
00995 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00996
00997 cccccccccccccccccccccccc
00998           if(.false.) then
00999             open(3011,file='oontest'//charnum5(iqx))
01000             do ix=nbln+1,nn
01001               igc=ix-nbln
01002               qqx(1:3) = (q(1:3)+ matmul(qlat, ngvecc(1:3,igc)))
01003               absqq  =  sqrt(sum(qqx(1:3)**2))
01004 c            absqg2x(ix) =sum( (2*pi/alat *q0i(1:3,nq0i))**2)
01005               do iy=nbln+ 1,nn
01006                 igc2=iy-nbln
01007                 if(ix==iy) then
01008                   write(3011,"('on : ',2i8,3i3,2x,3i3,f13.5,3x,2f20.10)")
01009      &        ix,iy, ngvecc(1:3,igc),ngvecc(1:3,igc2),absqq, oon(ix,iy)
01010                 else
01011                   write(3011,"('off:', 2i8,3i3, 2f20.10)")ix,iy,
01012      &        ngvecc(1:3,igc)-ngvecc(1:3,igc2),  oon(ix,iy)
01013                 endif
01014               enddo
01015             enddo
01016             close(3011)
01017           endif
01018 ccccccccccccccccccccccccccccc
01019
01020
01021 C... Generat pomat
01022 !    zmelt_new(K, ij) = \sum_I pomat(K,I)* zmelt(I, ij)
01023 !    means <psi_i psi_j | K> where |K> denote new mixed basis.
01024 !  See sxcf_fal2 and x0kf.
01025 !   Be carefull its transpose procedure---it is a little confusing...
01026           call pmatorth(oo,oon, pmat(1:no,1:nn), no, nn,
01027      o    pomat)
01028
01029 ccccccccccccccccccccccccccccccccccccccccccccccc
01030 ctttt
01031 c        pomat=0d0
01032 c        do ix= 1,ngb
01033 c          pomat(ix,ix)=1d0
01034 c        enddo
01035 c        do ix= 1,ngb
01036 c        do iy= 1,ngb
01037 c          if(pmat(ix,iy)/=0d0 )
01038 c     &   write(6,"(' ttt: pmat=',2i3,2d13.6)")
01039 c     &   ix,iy,pmat(ix,iy)
01040 c        enddo
01041 c        enddo
01042 c        write(6,"(' ttt:sumchk=',2d13.6,2i4)")
01043 c     &     sum(pomat(:,:)), no,nn
01044 ccccccccccccccccccccccccccccccccccccccccccccc
01045
01046           if( iqx <= nqibz ) deallocate(oon)
01047           deallocate(ppovl,oo)
01048 C... Store matching matrix
01049           write(ifpomat) q,nn,no,iqx
01050           write(ifpomat) pomat
01051           deallocate(pomat)
01052         endif
01053
01054 c$$$        if(newaniso()) then
01055 c$$$          continue
01056 c$$$        elseif(bzcase()==1.and.iqx==1)then
01057 c$$$          cycle
01058 c$$$        endif
01059
01060 !! == Write out VCCFP ==
01061         if(debug) write(6,*) 'write out vcoul'
01062         if(smbasis()) then
01063           ngb= nn
01064           allocate(vcoulnn(ngb,ngb))
01065           vcoulnn= matmul(transpose(dconjg(pmat(1:no,1:nn)))
01066      &              ,matmul(vcoul(1:no,1:no),pmat(1:no,1:nn)))
01067           vcoul(1:ngb,1:ngb)= vcoulnn
01068           deallocate(vcoulnn)
```

```
01069         endif
01070         if(wvcc) then
01071           write(ifvcfpout) ngb
01072           write(ifvcfpout) vcoul(1:ngb,1:ngb),q
01073         endif
01074         write(6,"(' ngc ngb/ngbo=',6i6)") ngc,ngb,ngbo
01075
01076 c Mix0vec --------------------------------
01077 !! diagonalize the Coulomb matrix
01078         if(.true.) then
01079 c        if( iqx > nqibz .or. iqx==1) then !feb2012 add iqx==1 for newansio()=T
01080           if(allochk) write(*,*) 'allocate( ppovl(ngc,ngc))'
01081           allocate( oo(ngb,ngb) )
01082           allocate( ppovl(ngc,ngc) )
01083           call mkppovl2(alat,plat,qlat,
01084      &          ngc,  ngvecc,
01085      &          ngc,  ngvecc,
01086      &          nbas, rmax, bas,
01087      o          ppovl)
01088           if(smbasis()) then
01089             oo = oon
01090             deallocate(oon)
01091           else
01092             oo = 0d0
01093             do ipl1=1,nbloch
01094               oo(ipl1,ipl1) = 1d0
01095             enddo
01096             do ix=1,ngc
01097               do iy=1,ngc
01098                 oo(nbloch+ix, nbloch+iy) = ppovl(ix,iy)
01099               enddo
01100             enddo
01101           endif
01102
01103           allocate( oox(ngb,ngb) )
01104           oox = oo
01105           write(6,*)' --- goto eigen check1 --- '
01106           allocate(  vcoul0(ngb,ngb) )
01107           vcoul0 = vcoul(1:ngb,1:ngb)
01108           if(allochk)
01109      &     write(*,*) 'allocate(hh(ngb,ngb),oo(ngb,ngb),oox,zz,eb,zzr)'
01110           allocate(hh(ngb,ngb),zz(ngb,ngb),eb(ngb),zzr(ngb))
01111           hh  = - vcoul0
01112 c          nmx = 15
01113           nmx = ngb
01114           call diagcv(oo,hh,zz,ngb, eb,nmx,1d99,nev)
01115           do ipl1=1,nev
01116             if(ipl1==11) write(6,*)' ... '
01117             if(ipl1>10.and.ipl1<nev-5) cycle
01118             write(6,'(i4,d23.16)')ipl1,-eb(ipl1)
01119           enddo
01120           write(6,"(' nev ngv q=',2i5,3f10.6)")nev,ngb,q
01121
01122 c$$$!! Modify -eb
01123 c$$$          if(iqx<=nqibz) then
01124 c$$$             do iqbz=1,nqbz     !! check
01125 c$$$                if(sum(abs(qbzwww(:,iqbz)-q))<1d-6) then
01126 c$$$                   iqbzx=iqbz
01127 c$$$                   goto 888
01128 c$$$                endif
01129 c$$$             enddo
01130 c$$$             stop 'hvccfp0:sum(abs(qbzwww(:,iq)-qbz(:,iq)))>1d-6'
01131 c$$$ 888         continue
01132 c$$$             if(abs((-eb(1)+eb(2))/eb(2))<1d-2) then
01133 c$$$!! Center. touching case. Respect smoothness when we change n1n2n3 division.
01134 c$$$                eb(1)=eb(1)*wqfac(iqbzx)
01135 c$$$                eb(2)=eb(2)*wqfac(iqbzx)
01136 c$$$             else
01137 c$$$                eb(1)=eb(1)*wqfac(iqbzx)
01138 c$$$             endif
01139 c$$$          endif
01140
01141 !     ! === save zz === apr2012takao
01142 c          if( newaniso().and.iqx==1 ) then
01143 c            if(sum(q**2)>1d-10) then
01144 c               stop ' hvccfp0: sanity check. |q(iqx)| /= 0'
01145 c            endif
01146           write(ifvcoud) ngb
01147           write(ifvcoud) q
01148           write(ifvcoud) -eb
01149           write(ifvcoud) zz
01150
01151 c$$$ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01152 c$$$        write(6,*)' ddddddddddddddd q=',q
01153 c$$$        do ix=1,ngb
01154 c$$$        do iy=1,ngb
01155 c$$$          aaaa=  sum(  dconjg(zz(1:ngb,ix))*matmul( oox,zz(1:ngb,iy))  )
```

```
01156 c$$$           if(ix==iy .and.  abs(aaaa-1d0) >1d-8 ) then
01157 c$$$            write(*,*)' dddd zcousum check',ix,iy,aaaa
01158 c$$$           endif
01159 c$$$           if(ix/=iy .and.  abs(aaaa) >1d-8 ) then
01160 c$$$            write(*,*)' dddd zcousum check',ix,iy,aaaa
01161 c$$$           endif
01162 c$$$        enddo
01163 c$$$        enddo
01164 c$$$ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01165
01166
01167
01168           write(6,*)
01169           write(6,'(" eig0 must be equal to the largest =", 2d24.16)')
01170      &       sum(  dconjg(zz(1:ngb,1))*matmul( vcoul0,zz(1:ngb,1))  )
01171           write(6,'(" zz norm check=",d24.16)')
01172      &    sum( dconjg(zz(1:ngb,1))*matmul(oox,zz(1:ngb,1)) )
01173           write(6,*)
01174 c          write(6,'(" --- vcoul(exact  no eee)=",d14.6," absq2=",d24.16)')
01175 c     &    fpi*voltot/(sum(tpiba**2*q(1:3)**2))
01176 c     &             , (sum(tpiba**2*q(1:3)**2))
01177           write(6,'(" --- vcoul(exact)=",d14.6," absq2=",d24.16)')
01178      &    fpi*voltot/(sum(tpiba**2*q(1:3)**2)-eee)
01179      &             , (sum(tpiba**2*q(1:3)**2)-eee)
01180           write(6,'(" --- vcoul(cal ) =",2d14.6)')
01181      &    sum( dconjg(zz(1:ngb,1))*matmul( vcoul0,zz(1:ngb,1)) )*voltot
01182 ccccccccccccccccccccccccccccccccccc
01183 c          do igc=1,ngb
01184 c          qqx(1:3) = (q(1:3)+ matmul(qlat, ngvecc(1:3,igc)))
01185 c          write(6,'(" --- vcoul(exact) xxx =",d14.6," absq2=",d24.16)')
01186 c     &    fpi*voltot/(sum(tpiba**2*(qqx(1:3)**2)-eee))
01187 c     &             , (sum(tpiba**2*(qqx(1:3)**2)-eee))
01188 c          write(6,'(" --- vcoul(cal ) xxx =",2d14.6)')
01189 c     &    sum( dconjg(zz(1:ngb,igc))*matmul( vcoul0,zz(1:ngb,igc)) )*voltot
01190 c          enddo
01191 ccccccccccccccccccccccccccccccccccccc
01192           deallocate(vcoul0)
01193
01194           if( iqx-nqibz>=1 ) then
01195              if( wqt(iqx-nqibz)==0d0) then ! MIZUHO-IR
01196
01197 C --- To get the vector <Mixed basis| q=0> --------------
01198 cki             if(is_mix0vec()==0) then     !used original befor oct2006
01199            if(.not.is_mix0vec()) then     !used original befor oct2006
01200 ! See switch.F ---> this is not used now.
01201              ifgb0vec_a =ifgb0vec1
01202              ifgb0vec_b =ifgb0vec
01203 cki           elseif(is_mix0vec()==1) then !oct2006 new case
01204            else
01205 ! ismix0vec=1 is to avoid problem at BZ boundary when is_mix0vec()=0.
01206              ifgb0vec_a =ifgb0vec
01207              ifgb0vec_b =ifgb0vec1
01208            endif
01209 c1... Case1 to write ifgb0vec -----------------------------------
01210            write(6,*)' voltot=',voltot
01211            if(ngc==0) then
01212              continue
01213            else
01214              do igc=1,ngc
01215                if( sum(abs( ngvecc(1:3,igc) ))==0 ) then
01216                  igc0=igc
01217                  exit
01218                endif
01219              enddo
01220              write(6,*)' igc0=',igc0,ngvecc(1:3,igc0)
01221              zzr(nbloch+1:nbloch+ngc) = ppovl(1:ngc,igc0)
01222            endif
01223
01224            allocate( gbvec(ngb), b0mat(nbloch) )
01225            write(6,*)' goto mkb0'
01226
01227 C ... get a vector <Product Basis| q+0>
01228            call mkb0( q, lxx,lx,nxx,nx, aa,bb,nr,nrx,rprodx,
01229      i        alat,bas,nbas,nbloch,
01230      o        b0mat)
01231            zzr(1:nbloch) = b0mat(1:nbloch)
01232 ccccccccccccccccccccccccc
01233 c          do igc=1,ngb
01234 c            write(6,"('ssss: ',i5,2d14.6)") igc, zzr(igc)
01235 c          enddo
01236 ccccccccccccccccccccccccc
01237            allocate(ooxi(ngb,ngb))
01238            ooxi=oox
01239            call matcinv(ngb,ooxi)
01240            gbvec = matmul(ooxi, zzr)
01241
01242 ccccccccccccc
```

```
01243 c           do igc=1,ngb
01244 c             write(6,"('ssss: ',i5,2d14.6)") igc, gbvec(igc)
01245 c           enddo
01246 ccccccccccccc
01247             deallocate(ooxi)
01248             dnorm = sqrt( sum(dconjg(gbvec)*zzr) )
01249 ! remove /dnorm at 14June2008. See main/hx0fp0.
01250 ! dnorm corresponds to volume (or sum of MT volume if no IPW).
01251 c            gbvec = gbvec /dnorm
01252 c            zzr   = zzr   /dnorm
01253 ! Not dnorm=1 at 14June2008. See main/hx0fp0.
01254 c            dnorm=1
01255             write(ifgb0vec_a,"(3d24.16,2i10,d24.16)") q, ngb,igc0,dnorm
01256             write(ifgb0vec_a,"(4d24.16)") (gbvec(i),zzr(i),i=1,ngb)
01257             deallocate( gbvec, b0mat)
01258 c1-----------------------------------------------------
01259
01260 c2... --- Case2 to write ifgb0vec c2 is problematic at BZ boundary...------
01261             dnorm  = 1d0
01262             zzr(:) = matmul(oox, zz(:,1))
01263             igc0 = 999999 !dummy now
01264 c phasex ---just to clean. this is irrelevant
01265             phasex =1d0
01266             do i=1,ngb
01267               if(abs(zz(i,1)) > 1d-3) phasex = abs(zz(i,1))/zz(i,1)
01268             enddo
01269             do i=1,ngb
01270               zz(i,1)= phasex * zz(i,1)
01271               zzr(i) = phasex * zzr(i)
01272             enddo
01273             write (ifgb0vec_b,"(3d24.16,2i10,d24.16)") q, ngb,igc0,dnorm
01274             write (ifgb0vec_b,"(4d24.16)") (zz(i,1),zzr(i),i=1,ngb)
01275           endif
01276           endif ! MIZUHO-IR
01277           if(allochk) !bugfix ---this was in inside or above if 7Feb2006
01278      &    write(*,*)'deallocate(hh,oo,zz,eb,oox,zzr)'
01279           deallocate(hh,oo,zz,eb,oox,zzr)
01280           deallocate(ppovl)
01281 c2--------------------
01282         endif
01283         idummy=iclose(trim(vcoudfile))
01284  1001 continue
01285       deallocate(ngvecc)
01286       call cputid(0)
01287       call flush(6)
01288       call mpi__finalize
01289       if(imode==202) call rx0( ' OK! hvccfp0 imode=202 only for Q0P')
01290       if(imode==0) call rx0( ' OK! hvccfp0 imode=0')
01291       if(imode==3) call rx0( ' OK! hvccfp0 imode=3')
01292       end
01293
01294       subroutine checkagree(a,b,char)
01295       real(8):: a(3),b(3)
01296       character*(*) :: char
01297       if(sum(abs(a-b))>1d-6) then
01298         write(6,*)' Error in checkagree:',char
01299 Cstop2rx 2013.08.09 kino        stop ' Error in checkagree:'
01300         call rx( ' Error in checkagree:')
01301       endif
01302       end
01303
01304       subroutine mkradmatch( p, nxdim,
01305      o          rdmatch)
01306 C- make rdmatch
01307 C-----------------------------------------------------
01308 Ci   p(1,i): phi     at mt for i-th basis
01309 Ci   p(2,i): dphi/dr at mt for i-th basis
01310 Co rdmatch(nxdim,nxdim)
01311 C-------
01312 Cr    phinew_j(r) =sum_i phi_i(r)* rdmatch (i,j)
01313 Cr     phinew_1(rmt)    =1      phinew_2(rmt)    =0
01314 Cr   d phinew_1(rmt)/dr =0    d phinew_2(rmt)/dr=1
01315 Cr for k >=3
01316 Cr     phinew_k(rmt)    =0
01317 Cr   d phinew_k(rmt)/dr =0
01318 C-----------------------------------------------------
01319       implicit none
01320       integer(4):: nxdim,lbas,i,i1,i2,ix
01321       real(8):: p(1:2, 1:nxdim), rdmatch(1:nxdim,1:nxdim)
01322       real(8):: pd,p1,p1d,p2,p2d,s,t, eps=1d-3,delta
01323 Cr                                        old     new
01324 c      write(6,"('mkradmatch: nxdim=',i4)") nxdim
01325       if(nxdim <=0) return
01326 Cstop2rx 2013.08.09 kino      if(nxdim ==1) stop 'mkradmatch err nxdim==1'
01327       if(nxdim ==1) call rx( 'mkradmatch err nxdim==1')
01328       rdmatch=0d0
01329 C... pivot--- get better set of phi for augmentation
```

```
01330       do
01331         i1= nxdim
01332         i2= nxdim-1
01333         p1 = p(1, i1)
01334         p2 = p(1, i2)
01335         p1d= p(2, i1)
01336         p2d= p(2, i2)
01337         write(6,"('mkradmatch: i1 p1 p1d=',i3,2d13.6)") i1,p1,p1d
01338         write(6,"('mkradmatch: i2 p2 p2d=',i3,2d13.6)") i2,p2,p2d
01339         delta = p1*p2d-p2*p1d
01340         if(abs(delta) <eps*p1*p2) then
01341           if(i2==1) then
01342             write(6,"(' i1 i2=',2i5,2d13.6)") i1,i2,p1d/p1,p2d/p2
01343 Cstop2rx 2013.08.09 kino            stop'mkradmatch: err poor linear dep'
01344             call rx( 'mkradmatch: err poor linear dep')
01345           endif
01346           i2=i2-1
01347         endif
01348         exit
01349       enddo
01350 C...
01351       call phimatch(1d0,0d0,  p1,p1d,p2,p2d, s,t)
01352       rdmatch(i1, 1)=  s
01353       rdmatch(i2, 1)=  t
01354       write(6,"('mkradmatch: 1 0    st=',2d13.5)") s,t
01355       call phimatch(0d0,1d0,  p1,p1d,p2,p2d, s,t)
01356       rdmatch(i1, 2)=  s
01357       rdmatch(i2, 2)=  t
01358       write(6,"('mkradmatch: 0 1    st=',2d13.5)") s,t
01359
01360       ix=2
01361       do i= 1,nxdim
01362         if(i==i1.or.i==i2) cycle
01363         ix=ix+1
01364 c        write(6,"('mkradmatch: i p pd=',i3,2d13.5)") i,p(1,i),p(2,i)
01365         call phimatch(p(1,i),p(2,i),  p1,p1d,p2,p2d, s,t)
01366         rdmatch(i,  ix)=  1d0
01367         rdmatch(i1, ix)=  -s
01368         rdmatch(i2, ix)=  -t
01369         write(6,"('mkradmatch: ix st=',i3,2d13.5)") ix,s,t
01370       enddo
01371       end
01372
01373       subroutine phimatch(p,pd, p1,p1d,p2,p2d, s,t)
01374 C --- match for given p and pd
01375 c   phi = s phi1 + t phi2 !slope and value are at MT
01376 c     p  = s p1  + t p2
01377 c     pd = s pd1 + t pd2
01378       implicit none
01379       real(8):: matinv(2,2),p,pd,p1,p1d,p2,p2d,s,t,delta,ddd1,ddd2
01380       delta = p1*p2d-p2*p1d
01381       matinv(1,1) = 1/delta *  p2d
01382       matinv(1,2) = 1/delta * (-p2)
01383       matinv(2,1) = 1/delta * (-p1d)
01384       matinv(2,2) = 1/delta *  p1
01385       s = matinv(1,1) *p  + matinv(1,2) *pd
01386       t = matinv(2,1) *p  + matinv(2,2) *pd
01387 C... check
01388       ddd1 = abs(s*p1  + t*p2   -  p )
01389 Cstop2rx 2013.08.09 kino      if(  ddd1 >1d-8 ) stop 'phimatch: ddd1 err'
01390       if(  ddd1 >1d-8 ) call rx( 'phimatch: ddd1 err')
01391       ddd2 = abs(s*p1d + t*p2d  -  pd)
01392 Cstop2rx 2013.08.09 kino      if(  ddd2 >1d-8 ) stop 'phimatch: ddd2 err'
01393       if(  ddd2 >1d-8 ) call rx( 'phimatch: ddd2 err')
01394       end
01395
01396       subroutine pmatorth(oo,oon,pmat,no,nn, pomat)
01397 C get conversion matrix from old mixed basis(no) to augmented mixed basis(nn).
01398 C pmatorth contains
01399 c   oo^{-1}_IJ
01400       implicit none
01401       integer(4):: no,nn,io,in,i
01402       complex(8):: pmat(no,nn),pomat(nn,no),oo(no,no),oon(nn,nn)
01403       complex(8),allocatable:: ooninv(:,:)
01404       real(8),allocatable:: eb(:)
01405       allocate(ooninv(nn,nn))
01406       ooninv = oon
01407       call matcinv(nn,ooninv) !generate ooninv
01408 c      pomat = matmul(ooninv, matmul(dconjg(transpose(pmat)),oo))
01409       pomat = transpose(matmul( oo, matmul(pmat,ooninv)))
01410       deallocate(ooninv)
01411       end
01412 c      allocate(pp(nn,nn),ppin(nn,nn),eb(nn),zz(nn,nn),zze(nn,nn))
01413 c      ppin = pp
01414 c      call diagcvh(ppin,nn,eb,zz)
01415 c      do i=1,nn
01416 c        zze(:,i) =  zz(:,i)* sqrt(eb(i))
```

```
01417 c      enddo
01418 c      pomat = matmul(pmat, matmul(zze,dconjg(transpose(zz))))
01419
01420       subroutine diagcvh(hh,ngb,eb,zz)
01421       implicit none
01422       integer(4):: nmx,nev,i,ngb
01423       complex(8):: hh(ngb,ngb),oo(ngb,ngb),zz(ngb,ngb)
01424       real(8):: eb(ngb)
01425       nmx=ngb
01426       oo = 0d0
01427       do i=1,ngb
01428         oo(i,i) = 1d0
01429       enddo
01430       call diagcv(oo,hh,zz,ngb, eb,nmx,1d99,nev)
01431       write(6,*)' diagcvv: ngb,nev=',ngb,nev
01432       do i=1,nev
01433         write(6,'(i4,d23.16)')i, eb(i)
01434       enddo
01435       end
01436 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01437       subroutine zgesvdnn2(no,nn, nnmx,epsmx,
01438      i   pmat,
01439      o   nnn)
01440 c pmat(no,nn) ---> pmat(no,nnn)
01441 Cio input          pmat(no,nn)
01442 Cio output reduced pmat(no,nnn)
01443       implicit none
01444       integer(4):: lwork,info,nn,no,nnn,nnmx,i
01445       complex(8)::  pmat(no,nn),uu(no,no),vt(nn,nn)
01446       real(8):: ss(nn),epsmx
01447       real(8),allocatable:: rwork(:)
01448       complex(8),allocatable:: work(:),vtt(:,:),pmatx(:,:)
01449 c      write(6,*)' sumchk pmat=',sum(abs(pmat(1:no,1:nn)))
01450       lwork=4*no
01451       allocate(work(lwork),rwork(5*no),pmatx(no,nn))
01452       pmatx =pmat
01453       call zgesvd('A','A',no,nn,pmat,no,ss,uu,no,vt,nn,work,lwork,rwork,info)
01454       nnn=-999
01455       do i=1,nn
01456         write(6,"(' i ss=',i4,' ', d13.5 )")i,ss(i) !    write(6,"(' i ss=',i4,'  ', d13.5,' ss0*ss=',d13.5
       )")i,SS(i),ss(i)*ss0(ngb-i+1)
01457 !         vtt(i,:)=ss(i)*vt(i,:)
01458         if(nnn==-999.and.ss(i)<epsmx) nnn = i-1
01459       enddo
01460 c      write(6,*) 'nnn=',nnn
01461 Cstop2rx 2013.08.09 kino      if(nnn==0) stop 'strange: nnn=0'
01462       if(nnn==0) call rx( 'strange: nnn=0')
01463       if(nnn>nnmx) nnn=nnmx
01464       pmat=pmatx
01465 c      pmat(:,1:nnn) = uu(:,1:nnn)
01466 !      write(6,"('sumcheck zzz  zzz-uu*s*vt=',d13.5,d13.5)")
01467 !     &  sum(abs(zw0bk)), sum(abs(zw0bk - matmul(uu,vtt)))
01468 !      if(abs(sum(abs(zw0bk - matmul(uu,vtt))))>1d-8*sum(abs(zw0bk)))
01469 !     &  stop 'sumcheck zzz  zzz-uu*s*vt= error'
01470 !      deallocate(vtt)
01471       end
01472
01473
01474 c-----------------------------------------------------------------------
01475       subroutine mkb0( q, lxx,lx,nxx,nx, aa,bb, nrr,nrx,rprodx,
01476      i        alat,bas,nbas,nbloch,
01477      o        b0mat)
01478 C--make the matrix elementes < B_q | exp(iq r)>
01479       implicit none
01480       integer(4) :: nlx,l,n,m,nr,ir,lm,ibl1,ibas,nrx,nbloch
01481
01482       integer(4) :: nbas,lxx, lx(nbas), nxx, nx(0:lxx,nbas),nrr(nbas)
01483       real(8)    :: rprodx(nrx,nxx,0:lxx,nbas),aa(nbas),bb(nbas),
01484      &   phi(0:lxx),psi(0:lxx), bas(3,nbas),
01485      &   alat,
01486      &   pi,fpi,tpiba,qg1(3),q(3),absqg,r2s,a,b
01487 c
01488       complex(8) :: b0mat(nbloch),img=(0d0,1d0) ,phase
01489 c
01490       integer(4),allocatable:: ibasbl(:), nbl(:), lbl(:), lmbl(:)
01491       real(8),allocatable :: ajr(:,:),rofi(:),rob0(:,:,:)
01492       real(8),allocatable::cy(:),yl(:)
01493       complex(8),allocatable :: pjyl(:,:)
01494 #ifdef COMMONLL
01495       integer(4) ll(51**2)
01496       common/llblock/ll
01497 #else
01498       integer(4) ll
01499 #endif
01500
01501 c-----
01502       write(6,*)'mkb0:'
```

```
01503       pi   = 4d0*datan(1d0)
01504       fpi  = 4*pi
01505       nlx  = (lxx+1)**2
01506 c
01507       tpiba = 2*pi/alat
01508       qg1(1:3) = tpiba * q(1:3)
01509       absqg    = sqrt(sum(qg1(1:3)**2))
01510 c
01511       allocate(ajr(1:nrx,0:lxx), pjyl(nlx,nbas),rofi(nrx),
01512      &  ibasbl(nbloch), nbl(nbloch), lbl(nbloch), lmbl(nbloch),
01513      &  cy(nlx),yl(nlx),rob0(nxx,0:lxx,nbas))
01514 c
01515       call sylmnc(cy,lxx)
01516       call sylm( qg1/absqg,yl,lxx,r2s) !spherical factor Y( q+G )
01517 c
01518       do ibas = 1,nbas
01519         a = aa(ibas)
01520         b = bb(ibas)
01521         nr= nrr(ibas)
01522         rofi(1)    = 0d0
01523         do ir      = 1, nr
01524           rofi(ir) = b*( exp(a*(ir-1)) - 1d0)
01525           call bessl(absqg**2*rofi(ir)**2,lx(ibas),phi,psi)
01526           do l  = 0,lx(ibas)
01527 c ... bessel function
01528             ajr(ir,l) = phi(l)* rofi(ir) **(l +1 )
01529             ! ajr = j_l(sqrt(e) r) * r / (sqrt(e))**l
01530           enddo
01531         enddo
01532
01533 c ... Coefficients for j_l yl  on MT  in the expantion of of exp(i q r).
01534         phase = exp( img*sum(qg1(1:3)*bas(1:3,ibas))*alat  )
01535         do lm = 1,(lx(ibas)+1)**2
01536           l = ll(lm)
01537           pjyl(lm,ibas) = fpi *img**l *cy(lm)*yl(lm) *phase  *absqg**l
01538         enddo
01539 c ... rob0
01540         do l = 0,lx(ibas)
01541           do n = 1,nx(l,ibas)
01542             call gintxx( ajr(1,l), rprodx(1,n,l,ibas), a,b,nr,
01543      o                   rob0(n,l,ibas) )
01544           enddo
01545         enddo
01546       enddo
01547
01548 c ... index (mx,nx,lx,ibas) order.
01549       ibl1 = 0
01550       do ibas= 1, nbas
01551         do l   = 0, lx(ibas) ! write(6,'(" l ibas nx =",3i5)') l,nx(l,ibas),ibas
01552           do n   = 1, nx(l,ibas)
01553             do m   = -l, l
01554               ibl1  = ibl1 + 1
01555               ibasbl(ibl1) = ibas
01556               nbl(ibl1) = n
01557               lbl(ibl1) = l
01558               lmbl(ibl1) = l**2 + l+1 +m ! write(6,*)ibl1,n,l,m,lmbl(ibl1)
01559             enddo
01560           enddo
01561         enddo
01562       enddo
01563 c ... pjyl * rob0
01564       do ibl1= 1, nbloch
01565         ibas= ibasbl(ibl1)
01566         n   = nbl(ibl1)
01567         l   = lbl(ibl1)
01568         lm  = lmbl(ibl1)
01569         b0mat(ibl1) = pjyl(lm,ibas) * rob0(n,l,ibas)
01570       enddo
01571       deallocate(ajr, pjyl,rofi,
01572      &  ibasbl, nbl, lbl, lmbl,
01573      &  cy,yl,rob0)
01574       end
```

4.35 main/hx0fp0.sc.m.F File Reference

Functions/Subroutines

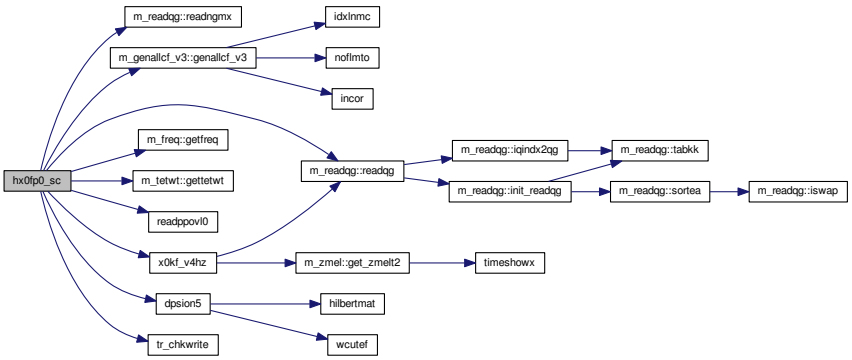
- program hx0fp0_sc
- subroutine tr_chkwrite (tagname, zw, iw, freqq, nblochpmx, nbloch, ngb, iq)

4.35.1 Function/Subroutine Documentation

4.35.1.1 program hx0fp0_sc ( )

Definition at line 1 of file hx0fp0.sc.m.F.

Here is the call graph for this function:



4.35.1.2 subroutine tr_chkwrite (character∗(∗) *tagname,* complex(8), dimension(nblochpmx,nblochpmx) *zw,* integer *iw,* real(8) *freqq,* integer *nblochpmx,* integer *nbloch,* integer *ngb,* integer *iq*)

Definition at line 1268 of file hx0fp0.sc.m.F.

Here is the caller graph for this function:

4.36 hx0fp0.sc.m.F

```
00001       program hx0fp0_sc
00002 !!  Calculate W-V for QSGW mode.
00003 !! We calculate chi0 by the follwoing three steps.
00004 !!  tetwt5: tetrahedron weights
00005 !!  x0kf_v4h: Accumlate Im part of the Lindhard function. Im(chi0) or Im(chi0^+-)
00006 !!  dpsion5: calculate real part by the Hilbert transformation from the Im part
00007       use m_readqg,only: readngmx,readqg
00008       use m_readeigen,only: init_readeigen,init_readeigen2,readeval
00009       use m_read_bzdata,only: read_bzdata,
00010      &     ngrp2=>ngrp,nqbz,nqibz,n1,n2,n3,qbas,ginv,qbasmc,
00011      &     dq_,qbz,wbz,qibz,wibz
00012 c     &     idteti, nstar,irk,nstbz
00013       use m_genallcf_v3,only: genallcf_v3,
00014      &     nclass,natom,nspin,nl,nn,ngrp,
00015      &     nlmto,nlnmx, nctot,niw,nw_input=>nw,
00016      &     alat,ef, diw,dw,delta,deltaw,esmr,symgrp,clabl,iclass,
00017      &     invg, il, in, im, nlnm,
00018      &     plat, pos, ecore, symgg
00019
```

```
00020       use keyvalue,only: getkeyvalue
00021       use m_pbindex,only: pbindex !,norbt,l_tbl,k_tbl,ibas_tbl,offset_tbl,offset_rev_tbl
00022       use m_readqgcou,only: readqgcou
00023
00024 !! Base data to generate matrix elements zmel*. Used in "call get_zmelt".
00025       use m_rdpp,only: rdpp,    !"call rdpp" generate following data.
00026      & nblocha,lx,nx,ppbrd,mdimx,nbloch,cgr
00027 !! Generate matrix element for "call get_zmelt".
00028       use m_zmel,only:    !these data set are stored in this module, and used when
00029      & nband,itq,ngcmx,ngpmx,    ppovlz,
00030      & ppbir,shtvg, miat,tiat , ntq
00031 !! frequency
00032       use m_freq,only: getfreq,
00033      &   frhis,freq_r,freq_i, nwhis,nw_i,nw,npm !output of getfreq
00034 !! antiferro
00035 c      use m_anf,only: anfcond,
00036 c     & laf,ibasf !,ldima,pos,natom
00037 !! tetwt
00038       use m_tetwt,only: gettetwt, !followings are output of 'L871:call gettetwt')
00039      &  whw,ihw,nhw,jhw,ibjb,nbnbx,nhwtot,n1b,n2b,nbnb
00040 !! MPI
00041       use m_mpi,only: mpi__hx0fp0_rankdivider2q,mpi__hx0fp0_rankdivider2s,
00042      &   mpi__qtask,mpi__initializeqspbm,mpi__finalize,mpi__root,
00043      &   mpi__broadcast,mpi__dblecomplexsendq,mpi__dblecomplexrecvq,mpi__rank,mpi__size,
00044      &   mpi__qranktab,mpi__consoleout,mpi__ss,mpi__se, mpi__allreducesums,
00045      &   mpi__barrier, mpi__rankq,mpi__rootq,mpi__roots
00046
00047       implicit none
00048       integer,allocatable:: nwgt(:,:)
00049       integer::iopen,maxocc2,iclose, ixc,iqxini,iqxend,
00050      &     ifhbe,  nprecb,mrecb,mrece,nlmtot,nqbzt,!nband,
00051      &     nq0i,i,nq0ix,neps,ngrpmx,mxx,nqbze,nqibze,ini,ix,ngrpx !ngcmx,
00052      &     ,nblochpmx,ndummy1,ndummy2,ifcphi,is,nwp, !ifvcfpout,,mdimx,nbloch
00053      &     ifepscond,nxx,ifvxcpout,ifgb0vec
00054      &     ,nw0,iw,ifinin,iw0,noccxv,noccx
00055      &     ,nprecx,mrecl,ifwd,ifrcwi,ifrcw,nspinmx,ifianf,ibas
00056      &     ,ibas1,irot,iq,ngb,iqixc2,ifepsdatnolfc,ifepsdat,ngbin,igc0
00057      &     ,kx,isf,kqxx,kp,job,nwmax !,ifev1,ifev2 !,nhwtot
00058      &     ,ihis,ik,ibib,ib1,ib2,ichkhis,ihww,j,imode
00059      &     ,  ifchipmlog ,   nw_w,nwmin  ! ,ngpmx
00060       real(8):: dum1,dum2,dum3,wqtsum,epsrng,dnorm, dwry,dwh,omg2, q(3),  qgbin(3),qx(3)
00061       real(8):: ua=1d0          ! this is a dummy.
00062       integer:: ifrb(2),ifcb(2),ifrhb(2),ifchb(2), ndble=8, nword
00063       real(8),allocatable:: vxcfp(:,:),  wqt(:), wgt0(:,:),q0i(:,:) !,nx(:,:),nblocha(:)
00064       integer,allocatable :: ngveccb(:,:), iqib(:),ifppb(:) !,lx(:) ngvecc(:,:),
00065       complex(8),allocatable:: geigb(:,:,:,:) ,geig(:,:),vcoul(:,:),
00066      &     zw(:,:),zw0(:,:), zxq(:,:,:),zxqi(:,:,:)
00067       real(8),allocatable :: eqt(:), !ppbrd (:,:,:,:,:,:,:),cgr(:,:,:,:)
00068      &     ppbrdx(:,:,:,:,:,:,:),aaa(:,:),symope(:,:),
00069      &     ppb(:,:),pdb(:,:),dpb(:,:),ddb(:,:), qbze(:,:),qibze(:,:) !,ecore(:,:)
00070 c     &  freqr(:),freqi(:) !rw(:,:),cw(:,:) --->zw
00071       complex(8),allocatable :: rcxq(:,:,:,:)
00072       complex(8) :: fff,img=(0d0,1d0)
00073       complex(8),allocatable :: wwk(:,:,:)
00074       real(8) ::qbzx(3)
00075       logical :: debug
00076 c      integer,allocatable:: ibasf(:)
00077       logical :: realomega, imagomega
00078       complex(8),allocatable:: zzr(:,:),ppovl(:,:),ppovlzinv(:,:) !,ppovlz(:,:)
00079       complex(8) :: epxxx,vcmean
00080       character*9 fileps
00081       character*15 filepsnolfc
00082       logical :: paralellx0=.true. !, hist
00083       character(5) :: charnum5
00084       character(20):: xxt
00085       real(8) :: emin, emax      ,emax2,emin2
00086 c     integer :: iSigma_en  !sf..21May02  !iSigma_en is integer
00087                                 !parameter stored in GWIN_V2
00088                                 !which determines approximation for  self-energy.
00089                                 !Self-energy should be made hermitian for energies to be real
00090 cxxx  !iSigma_en==0 SE_nn'(ef)+img integral:delta_nn'([SE_nn(e_n)+c.c.]/2-SE_nn(ef))
00091 cxxx  !iSigma_en==1 SE_nn'(ef)+delta_nn'([SE_nn(e_n)+c.c.]/2-SE_nn(ef))
00092                                 !iSigma_en==2 [SE_nn'((e_n+e_n')/2)+h.c.]/2
00093                                 !iSigma_en==3 [(SE_nn'(e_n)+SE_nn'(e_n'))/2+h.c.]/2
00094       real(8) :: omg2max,omg1max
00095       logical::imagonly=.false. , noq0p !,readgwinput
00096       integer::nwin, incwfin, verbose,nbcut,nbcut2,ifpomat,nnmx,ikpo,nn_,noo,iqxxx,nomx
00097       real(8)::efin
00098       logical :: nolfco=.false.
00099       integer:: isp1,isp2, ngc,mrecg ! bzcase,
00100       real(8)::  quu(3),deltaq(3),qqq(3)=0d0 !
00101       complex(8),allocatable:: wgt(:,:,:)
00102       real(8),allocatable:: qbz2(:,:)
00103       logical :: qbzreg
00104 !    logical ::smbasis !smbasis will be implemented in m_zmel.f which generates <phi|phi M>
00105       real(8):: q_r(3)
00106       complex(8),allocatable:: pomat(:,:)
```

```
00107       logical   :: timereversal,onceww
00108       integer :: jpm,ncc
00109       real(8) :: frr !, sciss
00110       integer :: ngb0,ifvcoud,idummy,igb1,igb2,ngb_in,nmbas1,nmbas2,iq0,ifisk,iqx,ig,nmbas1x !ifepstinv,
00111       complex(8),allocatable:: zcousq(:,:),epstinv(:,:),epstilde(:,:),zcousqrsum(:,:,:),zcousqr(:,:),eemat(
      :,:),zcousq0(:,:)
00112       real(8),allocatable:: vcousq(:),vcousq0(:),vcoudummy(:)
00113       real(8):: fourpi,sqfourpi,tpioa,absq,vcoul,vcoulsq
00114 !! Eq.(40) in PRB81 125102
00115 c     complex(8),allocatable::sk(:,:,:),sks(:,:,:),skI(:,:,:),sksI(:,:,:),
00116 c     &  w_k(:,:,:),w_ks(:,:,:),w_kI(:,:,:),w_ksI(:,:,:), llw(:,:), llwI(:,:),
00117       complex(8),allocatable::sk(:,:,:),sks(:,:,:),ski(:,:,:),sksi(:,:,:),
00118      &     w_k(:),w_ks(:),w_ki(:), w_ksi(:)
00119       complex(8),allocatable:: llw(:,:), llwi(:,:),w0(:),w0i(:),aaamat(:,:)
00120       real(8),allocatable:: dmlx(:,:),epinvq0i(:,:),epinv(:,:,:),epinvq0i_m1(:,:),wklm(:),qeibz(:,:,:)
00121       integer:: lxklm,nlxklm,ifidmlx,ifrcwx,iq0xx,ircw,nini,nend,iwxx,nw_ixxx,nwxxx,niwxxx,iwx,icc1,icc2
00122       complex(8):: vc1vc2
00123       integer,allocatable:: neibz(:),ngrpt(:),igx(:,:,:),igxt(:,:,:),eibzsym(:,:,:)
00124       real(8),allocatable:: aik(:,:,:,:)
00125       integer,allocatable:: aiktimer(:,:)
00126       integer:: l2nl, nmbas_in , iqxendx,imb2 !iqqv,
00127       logical:: eibz4x0,tiii,iprintx,chipm=.false.,iqinit,localfieldcorrectionllw
00128       real(8)::qvv(3),ecut,ecuts,hartree,rydberg,pi
00129       character(128):: vcoudfile
00130       integer :: iqeibz
00131       complex(8):: epslfc, axxx(10)
00132       integer:: src,dest
00133       integer:: ifw0w0i
00134       logical :: readw0w0i,symmetrize,eibzmode
00135       real(8):: schi=-9999 !dummy
00136       integer:: i_reduction_npm, i_reduction_nwhis,  i_reduction_nmbas2
00137       logical:: crpa
00138       integer,allocatable :: iclasst(:), invgx(:)
00139       integer:: ibasx,ificlass,ifile_handle,ifiq0p
00140       complex(8),allocatable:: ppovl_(:,:)
00141       logical:: tetra
00142 !-----------------------------------------------------------------------
00143 !TIME0_1001 ProgAll
00144 !TIME0_11001 readbzdata
00145       call mpi__initializeqspbm()
00146       call mpi__consoleout('hx0fp0_sc')
00147       call cputid(0)
00148       allocate( zzr(1,1)) !dummy
00149       hartree= 2d0*rydberg()
00150       pi     = 4d0*datan(1d0)
00151       fourpi = 4d0*pi
00152       sqfourpi=sqrt(fourpi)
00153       write(6,*) ' --- hx0fp0_sc Choose omodes below ----------------'
00154       write(6,*) '  ixc= 11,10011,or 1011 '
00155       write(6,*) ' --- Put number above ! -----------------'
00156       if( mpi__root ) then
00157          read(5,*) ixc !c     call readin5(ixc,iqxini,iqxend)
00158       end if
00159       call mpi__broadcast(ixc)
00160       crpa=.false.
00161       if(ixc==0) call rx( ' --- ixc=0 --- Choose computational mode!')
00162       call headver('hx0fp0_sc',ixc)
00163 c      call getkeyvalue("GWinput","ScaledGapX0",sciss,default=1d0)
00164 c      write(6,"(' ScaledGapX0=',f8.3)") sciss
00165       if(ixc==11) then
00166          write(6,*) " OK ixc=11 normal mode "
00167       elseif(ixc==10011) then
00168          write(6,*) " OK ixc=10011 crpa mode "
00169          crpa=.true.
00170       elseif(ixc==1011) then
00171          write(6,*) 'OK ixc=1011 Add W0W0I part at q=0'
00172       else
00173          write(6,*)'we only allow ixc==11. given ixc=',ixc
00174          call rx( 'error:we only allow ixc==11.')
00175       endif
00176 !! newaniso2 is now fixed to be .true.
00177       call getkeyvalue("GWinput","ecut_p" ,ecut, default=1d10 )
00178       call getkeyvalue("GWinput","ecuts_p",ecuts,default=1d10 )
00179 c    Prof.Naraga says this cause a stop in ifort --->why???
00180 c    write(6,*)'Timereversal=',Timereversal()
00181
00182 !! Readin BZDATA. See m_read_bzdata in gwsrc/rwbzdata.f
00183       call read_bzdata()
00184
00185 !TIME1_11001 "readbzdata"
00186 !TIME0_12001 Q0P
00187 !! Use regular mesh even for bzcase==2 and qbzreg()=T
00188 !!     off-regular mesh for bzcase==1 and qbzreg()=F
00189 c      if( ( bzcase()==2.and.qbzreg() )       .or.
00190 c     &     ( bzcase()==1.and.(.not.qbzreg()))      ) then
00191       if(.not.qbzreg()) then ! set off-gamma mesh
00192          deltaq= qbas(:,1)/n1 + qbas(:,2)/n2 +qbas(:,3)/n3
```

```
00193          do i=1,nqbz
00194             qbz(:,i) = qbz(:,i) - deltaq/2d0
00195             write(6,"('i qbz=',i3,3f8.4)") i,qbz(:,i)
00196          enddo
00197       endif
00198       write(6,"(' nqbz nqibz ngrp=',3i5)") nqbz,nqibz,ngrp
00199 !! === Readin by genallcf ===
00200 !! See "use m_genallcf_v3" at the begining of this routine
00201 !! We set basic data.
00202       nwin   = 0                !Readin nw from NW file
00203       incwfin= 0                !use ForX0 for core in GWIN
00204       efin =  0d0               !readin EFERMI
00205       call genallcf_v3(nwin,efin,incwfin) !in module m_genallcf_v3
00206       if(ngrp/= ngrp2) call rx( 'ngrp inconsistent: BZDATA and LMTO GWIN_V2')
00207 c      nw_input = nw ;
00208       write(6,*) 'nw delta=',nw_input,delta
00209       debug=.false.
00210       if(verbose()>=100) debug=.true.
00211       if(debug) write(6,*)' end of genallc'
00212       tpioa=2d0*pi/alat
00213 !!!!  WE ASSUME iclass(iatom)= iatom,  nclass = natom.  !!!!!!!!!!!!!!!!!!!!!!!!
00214       if(nclass /= natom) call rx( ' hx0fp0_sc: nclass /= natom ')
00215 !! --- tetra or not
00216 c      if(delta <= 0d0) then
00217       tetra =  .true.
00218       delta = -delta
00219       write(6,*)' hx0fp0.sc: tetrahedron mode delta=',delta
00220 c      else
00221 c         tetra = .false. ! switch for tetrahedron method for dielectric functions
00222 c      endif
00223 !! --- read dimensions of h,hb
00224       ifhbe      = iopen('hbe.d',1,0,0)
00225       read (ifhbe,*) nprecb,mrecb,mrece,nlmtot,nqbzt,nband,mrecg
00226       if(nlmto/=nlmtot) call rx( ' hx0fp0: nlmto/=nlmtot in hbe.d')
00227       if(nqbz /=nqbzt ) call rx( ' hx0fp0: nqbz /=nqbzt  in hbe.d')
00228 !! --- Readin Offset Gamma --------
00229       if(debug) write(6,*) 'reading QOP'
00230       ifiq0p=ifile_handle()
00231       open (ifiq0p,file='Q0P')
00232       read (ifiq0p,"(i5)") nq0i
00233       write(6,*) ' ### nqibz nq0i=', nqibz,nq0i
00234       allocate( wqt(1:nq0i),q0i(1:3,1:nq0i) )
00235       do i=1,nq0i
00236          read (ifiq0p, * ) wqt(i),q0i(1:3,i)
00237       enddo
00238       nq0ix = nq0i
00239       do i=1,nq0i
00240          if(wqt(i)==0d0 ) then
00241             nq0ix = i-1
00242             exit
00243          endif
00244       enddo
00245       neps=nq0i-nq0ix ! number of zero weight q0p which are used for ixc=2 or 3 mode.
00246       write(6,*) ' num of zero weight q0p=',neps
00247       write(6,"(i3,f14.6,2x, 3f14.6)" )(i, wqt(i),q0i(1:3,i),i=1,nq0i)
00248       close(ifiq0p)
00249 c$$$      if(.not.newaniso2) then
00250 c$$$         wqtsum = sum(abs(wqt(1:nq0i)))
00251 c$$$         call getkeyvalue("GWinput","TestNoQ0P",noq0p,default=.false.)
00252 c$$$      endif
00253
00254 !TIME1_12001 "Q0P"
00255 !TIME0_13001 mptauof
00256       call getsrdpp2(nclass,nl,nxx)
00257       call readngmx('QGpsi',ngpmx)
00258       call readngmx('QGcou',ngcmx)
00259       write(6,*)' ngcmx ngpmx=',ngcmx,ngpmx
00260       nqbze  = nqbz *(1 + nq0i)
00261       nqibze = nqibz + nq0i
00262       allocate( qbze(3, nqbze), qibze(3, nqibze))
00263       call dcopy(3*nqbz, qbz,  1, qbze,1)
00264       call dcopy(3*nqibz,qibz, 1, qibze,1)
00265       do i = 1,nq0i
00266          qibze(:,nqibz+i)  = q0i(:,i)
00267          ini = nqbz*(1 + i -1)
00268          do ix=1,nqbz
00269             qbze(:,ini+ix)   = q0i(:,i) + qbze(:,ix)
00270          enddo
00271       enddo
00272 !! ---------------- dummy ngrpx=1 -------------------
00273       ngrpx = 1
00274       l2nl=2*(nl-1)
00275       allocate(symope(3,3))
00276       symope(1:3,1) = (/1d0,0d0,0d0/)
00277       symope(1:3,2) = (/0d0,1d0,0d0/)
00278       symope(1:3,3) = (/0d0,0d0,1d0/)
00279 !!  dummy. Get space-group transformation information. See header of mptaouof.
```

```
00280       ificlass=ifile_handle()
00281       open (ificlass,file='CLASS')
00282       allocate(iclasst(natom),invgx(ngrp)
00283      & ,miat(natom,ngrp),tiat(3,natom,ngrp),shtvg(3,ngrp))
00284       write(6,*)'  --- Readingin CLASS info ---'
00285       do ibas = 1,natom
00286         read(ificlass,*) ibasx, iclasst(ibas)
00287         write(6, "(2i10)") ibasx, iclasst(ibas)
00288       enddo
00289       close(ificlass)
00290       call mptauof(symope,ngrpx,plat,natom,pos,iclasst
00291      o ,miat,tiat,invgx,shtvg ) !note: miat,tiat,shtvg are defined in m_zmel.
00292       if(verbose()>=40) write (*,*)' hsfp0.sc.m.F: end of mptauof'
00293 !!  call rdpp gives ppbrd = radial integrals and cgr = rotated cg coeffecients.
00294       call rdpp(nxx, nl, ngrpx, nn, nclass, nspin, symope,qbas)
00295       ntq=nband
00296       allocate(itq(ntq))
00297       do i=1,ntq
00298         itq(i)=i
00299       enddo
00300 !! Pointer to optimal product basis
00301 c      allocate(imdim(natom))
00302 c      call indxmdm (nblocha,nclass,iclass,natom,
00303 c     o imdim )                    !in m_zmel
00304       nblochpmx = nbloch + ngcmx
00305       allocate(ngveccb(3,ngcmx))
00306       iqxend = nqibz + nq0i
00307       write(6,*) ' nqibz nqibze=',nqibz,nqibze
00308 !TIME1_13001 "mptauof"
00309 !TIME0_14001 init_readeigen
00310 !!... initialization of readEigen !readin m_hamindex
00311       call init_readeigen(ginv,nspin,nband,mrece)!EVU EVD are read in init_readeigen
00312       call init_readeigen2(mrecb,nlmto,mrecg)
00313 c     --- ecore ---
00314 c     allocate(ecore(nctot,nspin)) !core energies
00315 c     do  is = 1,nspin
00316 c     if (nctot .gt. 0) then
00317 c     call catch1 (w(iecore),is,nctot,2,ecore(:,is)) !core energies
00318 c     write(6,*)' ecore is=',is,ecore(:,is)
00319 c     endif
00320 c     enddo
00321
00322 c     --- set realomega, imagomega tetra nw niw nwp ifgb0vec ---------------------
00323 !     nwp, freq_r,  frhis(1:nwhis+1)
00324 c     if   ( ixc==1 ) then !old imagw = 2 case
00325 c     realomega =.true.
00326 c     imagomega =.true.
00327 c     stop 'hsfp0sc: ixc==1 is not implimented'
00328 ccccccccccccccccccccfaleev 21May02,  use only ixc=1,11 modes ccccccccc
00329 c     elseif( ixc==2.or.ixc==3 ) then
00330 c     realomega =.true.
00331 c     imagomega =.false.
00332 c     niw = 0
00333 c     ifepscond = 2102
00334 c     open (ifepscond,file='EPScond')
00335 c     read (ifepscond,*) epsrng, dwry !epsrng dw in Ry
00336 c     dw = dwry/2d0
00337 c     close(ifepscond)
00338 c     if(dw==0d0) then
00339 c     nw = 1
00340 c     else
00341 c     nw = (epsrng/2d0 - 1d-10)/(dw/2d0) + 2 !epsrng/2d0 corresponds to in a.u.
00342 c     endif
00343 c     allocate(epsi(nw,neps))
00344 c     if(paralellx0) then
00345 c     ifgb0vec = iopen ( "Mix0vec."
00346 c     &     //xxt(iqxini,iqxend),1,3,0)
00347 c     c          "//charnum5(iqxini)//'to'//charnum5(iqxend),1,3,0)
00348 c     else
00349 c     ifgb0vec = iopen ( "Mix0vec",1,3,0)
00350 c     endif
00351 c     elseif(ixc==4.or.ixc==5.or.ixc==6) then
00352 c     ! ... These are test modes.
00353 c     ! ixc=4 tetrahedren weight test. tetwt5.vs.tetwt5. Write tethis.chk
00354 c     ! ixc=5 Spectrum function (Img part) along the Real axis with tetwt4
00355 c     ! ixc=6 Spectrum function (Img part) along the Real axis with tetwt5. Histgram method.
00356 c     realomega = .true.
00357 c     imagomega = .false.
00358 c     tetra     = .true.
00359 c     niw = 0
00360 c     ! ---  For tetwt5 ---  the tetrahedron weight for spectrum function (imaginary part)
00361 c     !   Histogram bins are specified by freq_r(1:nwp)
00362 c     !     nwp=nw+1; frhis(1)=0
00363 c     !     The 1st   bin  is     [frhis(1),  frhis(2)]   ...
00364 c     !     The last  bin  is     [frhis(nw), frhis(nwp)].
00365 c
00366 c     ! ... These parameters specifies a test histgram bins;Sergey's mesh just for test modes.
```

```
00367 c     nw0 = 200    !100    800
00368 c     dwh = 0.01d0 !0.02d0 0.0025d0 !in hartree
00369 c     ! ...
00370 c     call findemaxmin(ifev,nband,nqbz,nspin,emax,emin)
00371 c     if (nctot .gt. 0) Emin = minval(ecore)
00372 c     omg2max = (Emax-Emin)*.5+.2d0    !(in Hartree) covers  all relevant omega, +.2 for margin
00373 c     omg1max = dwh*(nw0-1)
00374 c     nwp = int(sqrt(omg2max*(2*nw0-1d0)/dwh-(nw0**2-3*nw0+1d0)))+1 ! + 1 for margin
00375 c     nw  = nwp-1
00376 c     write(6,*) Emax,Emin,nw0,nw  ! nwp is new max number in frequency array
00377 c     write(6,'(a32,2i7,2d15.3)')'hx0fp1: nw0,nw,omg1max,omg2max='
00378 c     &              , nw0,nw,  omg1max,omg2max
00379 c     if (nw <= nw0) stop 'hx0fp0:ixc==[456] nw2 <= nw'
00380 c     allocate(freq_r(nwp))
00381 c     do iw=1,nwp  !This is a test mesh by Sergey.Faleev
00382 c     if(iw<=nw0) then;  freq_r(iw)=dwh*(iw-1)
00383 c     else;  freq_r(iw)=dwh*(iw**2+nw0**2-3*nw0+1)/(2*nw0-1d0)
00384 c     endif
00385 c     enddo !freq_r(iw) is linear for iw<=nw and quadratic for nw<iw<=nw2
00386 c     !freq_r(iw) chosen in such a way that it is continues with
00387 c!!!  nw nwp=nw+1 freq_r(1:nwp) are used after here.
00388 c     allocate(frhis(nwp))
00389 c     frhis=freq_r(1:nwp)
00390 c     nwhis=nw
00391
00392 !! We get frhis,freq_r,freq_i, nwhis,nw,npm  by getfreq
00393       realomega = .true.
00394       imagomega = .true.
00395       tetra     = .true.
00396       call findemaxmin(nband,qbze,nqbze,nspin, emax,emin)
00397       if(.not.qbzreg()) then
00398          allocate(qbz2(3,nqbz))
00399          do iq=1,nqbz
00400             qbz2(:,iq)=qbz(:,iq)+dq_
00401          enddo
00402          call findemaxmin(nband,qbz2,nqbz,nspin ,emax2,emin2)
00403          emax=max(emax,emax2)
00404          emin=min(emin,emin2)
00405          deallocate(qbz2)
00406       endif
00407       if (nctot > 0) emin=minval(ecore(:,1:nspin))
00408       omg2max = (emax-emin)*.5d0+.2d0
00409             ! (in Hartree) covers all relevant omega, +.2 for margin
00410       write(6,"(' emin emax omega2max=',3f13.5)") emin, emax, omg2max
00411       call getfreq(.false.,realomega,imagomega,tetra,omg2max,nw_input,niw,ua,mpi__root)
00412       nwp = nw+1
00413
00414 !! We first accumulate Imaginary parts. Then do K-K transformation to get real part.
00415       noccxv = maxocc2(nspin,ef, nband, qbze,nqbze)
00416         !max no. of occupied valence states
00417       if(noccxv>nband) call rx( 'hx0fp0_sc: all the bands filled! too large Ef')
00418       noccx  = noccxv + nctot
00419       nprecx = ndble        !We use double precision arrays only.
00420       mrecl  = nprecx*2*nblochpmx*nblochpmx/nword()
00421       if(mpi__root)then
00422         ifwd   = iopen('WV.d',1,-1,0)
00423         write (ifwd,"(1x,10i14)") !"(1x,i3,i8,i5,5i4)")
00424      &  nprecx,mrecl,nblochpmx,nwp,niw,nqibz + nq0i-1,nw_i
00425         ifwd = iclose('WV.d'); ifwd=0
00426       endif
00427       allocate(  zw(nblochpmx,nblochpmx) )
00428       nspinmx = nspin
00429 !TIME1_14001 "init_readeigen"
00430 !TIME0_15001 ppbafp_v2
00431
00432 !!... these are used x0k
00433       call getkeyvalue("GWinput","nbcutlow",nbcut, default=0 )
00434       call getkeyvalue("GWinput","nbcutlowto",nbcut2, default=0 )
00435       write(6,"(' nbcut nbcutlowto=',2i5)") nbcut,nbcut2
00436 !! -- ppb= <Phi(SLn,r) Phi(SL'n',r) B(S,i,Rr)>
00437 !! This is general for rotated CG coefficient; but hx0fp0 mode is only for  ngrpx=1 (not rotated).
00438 !! Compare usage in hsfp0 modes.
00439       irot=1
00440       allocate( ppbir(nlnmx*nlnmx*mdimx*nclass,irot,nspin))
00441       do is = 1,nspin
00442         call ppbafp_v2(irot,ngrpx,is,nspin,
00443      i   il,in,im,nlnm,         !w(i_mnl),
00444      i   nl,nn,nclass,nlnmx,
00445      i   mdimx,lx,nx,nxx,       !Bloch wave
00446      i   cgr, nl-1,             !rotated CG
00447      i   ppbrd,                 !radial integrals
00448      o   ppbir(:,irot,is))      !this is in m_zmel
00449       enddo
00450       if(debug) write(6,*) ' end of ppbafp_v2'
00451 !TIME1_15001 "ppbafp_v2"
00452 !TIME0_16001 readqgcou
00453       call getkeyvalue("GWinput","nbcutlow",nbcut, default=0 )
```

```
00454       call getkeyvalue("GWinput","nbcutlowto",nbcut2, default=0 )
00455       write(6,"(' nbcut nbcutlowto=',2i5)") nbcut,nbcut2
00456       iqxini=1 !for newaniso
00457       eibzmode = eibz4x0()
00458
00459 !! === Use of symmetry. EIBZ procedure PRB81,125102 ===
00460 !!  For rotation of zcousq.  See readeigen.F rotwv.F ppbafp.fal.F(for index of product basis).
00461       if(eibzmode) then
00462 !! commentout block inversion Use iqxendx=iqxend because of full inversion
00463         iqxendx=iqxend
00464         allocate( nwgt(nqbz,iqxini:iqxendx), !qeibz(3,nqbz,iqxini:nqibz),neibz(iqxini:nqibz),
00465      &        igx(ngrp*2,nqbz,iqxini:iqxendx),igxt(ngrp*2,nqbz,iqxini:iqxendx),
00466      &        eibzsym(ngrp,-1:1,iqxini:iqxendx))
00467         iprintx=.false.
00468
00469         write(6,*)
00470         write(6,"('=== Goto eibzgen === TimeRevesal switch =',l1)")timereversal()
00471         if(mpi__root) iprintx=.true.
00472         call eibzgen(nqibz,symgg,ngrp,qibze(:,iqxini:iqxend),iqxini,iqxendx,qbz,nqbz,
00473      i        timereversal(),ginv,iprintx,
00474      o        nwgt,igx,igxt,eibzsym,tiii)
00475         write(6,"('Used timeRevesal for EIBZ = ',l1)") tiii
00476         call cputid(0)
00477 c$$$
00478 c$$$         write(6,"('TimeRevesal switch = ',l1)") timereversal()
00479 c$$$         call
      eibzgen(nqibz,symgg,ngrp,qibze(:,iqxini:iqxend),iqxini,iqxendx,qbz,nqbz,timereversal(),ginv,iprintx,
00480 c$$$     o        nwgt,igx,igxt,eibzsym)
00481 c$$$!! Check timereversal is required for symmetrization operation or not. If not tiii=timereversal=F is
      used.
00482 c$$$!! this is because the symmetrization is a little time-consuming.
00483 c$$$         tiii=timereversal()
00484 c$$$         if(minval(igxt)==1) tiii=.false.
00485 c$$$         iprintx=.true.
00486 c$$$ccccccccccccccccccc
00487 c$$$c     tiii=.true.
00488 c$$$ccccccccccccccccccc
00489 c$$$         write(6,"('=== goto eibzgen === used timereversal=',l1)")tiii
00490 c$$$         call
      eibzgen(nqibz,symgg,ngrp,qibze(:,iqxini:iqxend),iqxini,iqxendx,qbz,nqbz,tiii,ginv,iprintx,
00491 c$$$     o        nwgt,igx,igxt,eibzsym)
00492
00493 !All input. this returns requied index stored in arrays in m_pbindex.
00494         call pbindex(natom,lx,l2nl,nx)
00495                           ! PBindex: index for product basis.  We will unify this system; still similar is
      used in ppbafp_v2.
00496         call readqgcou() ! no input. Read QGcou and store date into variables.
00497 !!  call Spacegrouprot(symgg,ngrp,plat,natom,pos) ! all inputs.
00498       else !dummy allocation to overlaid -check bound !sep2014
00499         iqxendx=iqxend
00500         allocate( nwgt(1,iqxini:iqxendx),igx(1,1,iqxini:iqxendx)
00501      &    ,igxt(1,1,iqxini:iqxendx), eibzsym(1,1,iqxini:iqxendx)) !dummy
00502       endif
00503
00504       allocate( llw(nw_i:nw,nq0i), llwi(niw,nq0i) )
00505       llw=1d99
00506       llwi=1d99
00507       if(ixc==1011)then !ixc==11 is a debug mode to test contrib. at \Gamma point.
00508         goto 1191
00509       endif
00510
00511 !! rank divider
00512       call mpi__hx0fp0_rankdivider2q(iqxini,iqxend)
00513       call mpi__hx0fp0_rankdivider2s(nspinmx)
00514
00515 !! == Calculate x0(q,iw) and W == main loop 1001 for iq.
00516 !! NOTE: iq=1 (q=0,0,0) write 'EPS0inv', which is used for iq>nqibz for ixc=11 mode
00517 !! Thus it is necessary to do iq=1 in advance to performom iq >nqibz.
00518 !! (or need to modify do 1001 loop).
00519 !! ---------------------------------------------------------------
00520 !! === do 1001 loop over iq ==============================================
00521 !! ---------------------------------------------------------------
00522       iqinit=.true.
00523       write(6,'("irank=",i5," allocated(MPI__qtask)=",L5)')mpi__rank,allocated(mpi__qtask)
00524       do iq = iqxini,iqxend
00525         if(mpi__qtask(iq)) write(6,'("irank iq=",i5,i5)') mpi__rank,iq
00526       enddo
00527 !TIME1_16001 "readqgcou"
00528 !TIME0_170001 do1001
00529       do 1001 iq = iqxini,iqxend
00530         if( .not. mpi__qtask(iq) ) cycle
00531         if (mpi__roots) then
00532           ifrcwi = iopen('WVI.'//charnum5(iq),0,-1,mrecl)
00533           ifrcw  = iopen('WVR.'//charnum5(iq),0,-1,mrecl)
00534         endif
00535         call cputid(0)
00536         q = qibze(:,iq)
```

```
00537         call readqg('QGcou', q, ginv,  quu,ngc,ngveccb) ! q was qq
00538
00539 !! Caution : confusing point
00540 !!  ngc by QGcou is shown at the bottom of lqg4gw.
00541 !!  ngc read from PPOVL are given by rdata4gw.
00542 !!  Note that  ngc(iq>nqibz )=ngc (q=0), because when it is generated in mkqg.F
00543 !!
00544 c         if( newaniso2.and.iq==1 ) then ! *sanity check
00545         if( iq==1 ) then        ! *sanity check
00546           if(sum(q**2)>1d-10) call rx( ' hx0fp0.sc: sanity check. |q(iqx)| /= 0')
00547         endif
00548
00549 !! ==== readin Coulomb matrix ====
00550         ngb = nbloch + ngc
00551         write(6,"('do 1001: iq q=',i5,3f9.4)")iq,q
00552         write(6,*)'nbloch ngb ngc=',nbloch,ngb,ngc
00553
00554 !! === readin diagonalized Coulomb interaction ===
00555 !! zcousq: E(\nu,I), given in PRB81,125102; vcousq: sqrt(v), as well.
00556 c         if(newaniso2) then
00557         vcoudfile='Vcoud.'//charnum5(iq) ! iq was iqqv this is closed at the end of do 1001
00558         ifvcoud = iopen(trim(vcoudfile),0,-1,0)
00559         read(ifvcoud) ngb0
00560         read(ifvcoud) qvv
00561         if(sum(abs(qvv-q))>1d-10) then
00562           write(6,*)'qvv =',qvv
00563           call rx( 'hx0fp0: qvv/=0 hvcc is not consistent')
00564         endif
00565         if(allocated(zcousq)) deallocate( zcousq,vcousq )
00566         allocate( zcousq(ngb0,ngb0),vcousq(ngb0))
00567         read(ifvcoud) vcousq
00568         read(ifvcoud) zcousq
00569         idummy=iclose(trim(vcoudfile))
00570         vcousq=sqrt(vcousq)
00571
00572 c         if(newaniso2.and. iq>nqibz.and.(.not.localfieldcorrectionllw())  ) then
00573         if(iq>nqibz.and.(.not.localfieldcorrectionllw())  ) then
00574           if( ngb0/=ngb ) then
00575             call rx( 'hx0fp0.m.f:ngb0/=ngb')
00576           endif
00577           nolfco =.true.
00578           nmbas_in = 1
00579 c         elseif(newaniso2) then !.and.iq==1) then
00580         else
00581           nolfco = .false.
00582           nmbas_in = ngb
00583         endif
00584         nmbas1 = nmbas_in
00585         nmbas2 = nmbas1
00586
00587 !! newaniso=T case. Used in get_zmelt in m_zmel called in x0kf_v4hz
00588         if(allocated(ppovlz)) deallocate(ppovlz)
00589         if(allocated(ppovlzinv)) deallocate(ppovlzinv)
00590         if(allocated(ppovl)) deallocate(ppovl)
00591         allocate(ppovl(ngc,ngc),ppovlz(ngb,ngb),   ppovlzinv(ngb,ngb))
00592         call readppovl0(q,ngc,ppovl) !q was qq
00593 c         ppovlz(1:nbloch,:) = zcousq(1:nbloch,:)
00594 c         ppovlz(nbloch+1:nbloch+ngc,:)
00595 c     &     = matmul(ppovl,zcousq(nbloch+1:nbloch+ngc,:))
00596         allocate(ppovl_(ngb,ngb))
00597         ppovl_=0d0
00598         do i=1,nbloch
00599           ppovl_(i,i)=1d0
00600         enddo
00601         ppovl_(nbloch+1:nbloch+ngc,nbloch+1:nbloch+ngc)=ppovl
00602         if(.not.eibz4x0()) then !sep2014 added for eibz4x0=F
00603           ppovl_= matmul(ppovl_,zcousq)
00604         endif
00605         ppovlz = ppovl_
00606         deallocate(ppovl_,ppovl)
00607
00608 c$$$  if(ixc==11) then
00609 c$$$  write(6,*)" xxx2: memsize 8*ngb*ngb*nwhis=", 8*ngb*ngb*nwhis,' ngb nwhis=',ngb,nwhis
00610 c$$$  allocate( rcxq(ngb,ngb,nwhis,npm) )
00611 c$$$  rcxq=(0d0,0d0)
00612 c$$$  else
00613 c$$$  if(onceww(2)) write(6,*)" xxx2:allocate zxq zxqi memsize 16*ngb*ngb*(nwp+niw)=",
00614 c$$$  &    16*ngb*ngb*(1+nwp+niw),' ngb nwp niw=',ngb,nwp,niw
00615 c$$$  allocate(
00616 c$$$  &    zxq (ngb,ngb,nw_i:nw),   !,nwp) feb2006
00617 c$$$  &    zxqi(ngb,ngb,niw))
00618 c$$$  zxq=0d0; zxqi=0d0
00619 c$$$  endif
00620
00621         allocate( rcxq(nmbas1,nmbas2,nwhis,npm) )
00622         allocate( zw0(ngb,ngb) ) !, zxq (ngb,ngb,nw_i:nw), zxqi(ngb,ngb,niw) )
00623         rcxq = 0d0
```

```
00624
00625 !! ---------------------------------------------------------------
00626 !! === loop over spin=== ======================================
00627 !! ---------------------------------------------------------------
00628 !TIME0_180001 Do1003
00629 !         do 1003 is = 1,nspinmx
00630          do 1003 is = mpi__ss,mpi__se
00631            write(6,"(' ### ',2i4,' out of nqibz+n0qi nsp=',2i4,' ### ')")
00632      &     iq, is, nqibz + nq0i,nspin
00633            if(debug) write(6,*)' niw nw=',niw,nw
00634            isf = is
00635 !! Tetrahedron weight.
00636 !!     nbnbx
00637 !!     ihw(ibjb,kx): omega index, to specify the section of the histogram.
00638 !!     nhw(ibjb,kx): the number of histogram sections
00639 !!     jhw(ibjb,kx): pointer to whw
00640 !!     whw( jhw(ibjb,kx) ) \to whw( jhw(ibjb,kx) + nhw(ibjb),kx)-1 ), where ibjb=ibjb(ib,jb,kx)
00641 !!     : histogram weights for given ib,jb,kx for histogram sections
00642 !!     from ihw(ibjb,kx) to ihw(ibjb,kx)+nhw(ibjb,kx)-1.
00643 c           write(6,*) ' --- goto x0kf_v4hz ---- newaniso= ',newaniso2
00644            call gettetwt(q,iq,is,isf,nwgt(:,iq))
00645 !! == x0kf_v4hz is the main routine to accumalte imaginary part of x0 ==
00646            iqeibz=iq
00647            symmetrize=.false.
00648            if(npm==1) then
00649              ncc=0
00650            else
00651              ncc=nctot
00652            endif
00653            call x0kf_v4hz(npm,ncc,
00654      i      ihw,nhw,jhw,whw,nhwtot, ! tetwt5
00655      i      n1b,n2b,nbnbx,nbnb,  ! use whw by tetwt5 ,
00656      i      q,
00657      i      nspin,is,isf, symmetrize, !
00658      i      qbas,ginv,  qbz,wbz,
00659      d      nlmto,nqbz,nctot,    !noccx,noccxv,
00660      d      nbloch,  nwhis,      !nlnmx,mdimx,
00661      i      iq,ngb,ngc,ngpmx,ngcmx, !ngb/=ngc+nbloch for smbasis()=T oct2005
00662      i      nqbze,nband,nqibz,
00663      o      rcxq,                ! rcxq is the accumulating variable for spins
00664      i      nolfco,zzr,nmbas_in, zcousq, !ppovl,nmbas2 is removed.,nmbas1 ppovlz,
00665      i      chipm,eibzmode,      !zloffd,
00666      i      nwgt(:,iqeibz),igx(:,:,iqeibz),igxt(:,:,iqeibz),ngrp, eibzsym(:,:,iqeibz),crpa)
00667            write(6,*)' end of x0kf_v4h sum rcxq=',sum(abs(rcxq))
00668            deallocate(ihw,nhw,jhw, whw,ibjb )
00669            if(tetra)  deallocate( n1b,n2b)
00670  1003   continue;write(6,*) 'end of spin-loop nwp=',nwp !end of spin-loop
00671 !TIME1_180001 "Do1003"
00672 c===========end of spin loop============================================
00673 !! symmetrize and convert to Enu basis by dconjg(tranpsoce(zcousq)*rcxq8zcousq if eibzmode
00674 !TIME0_190001 x0kf_sym
00675          if(eibzmode)  then
00676            symmetrize= .true.
00677            is=1                 ! dummy
00678            call x0kf_v4hz(npm,ncc,
00679      i      ihw,nhw,jhw,whw,nhwtot, ! tetwt5
00680      i      n1b,n2b,nbnbx,nbnb,  ! use whw by tetwt5 ,
00681      i      q,
00682      i      nspin,is,isf, symmetrize, !
00683      i      qbas,ginv,  qbz,wbz,
00684 c     i              nblocha,!nlnm,nlnmv,nlnmc,iclass,
00685 c     i              ppb(1,is),
00686 c     i              icore,ncore,
00687      d      nlmto,nqbz,nctot,    !noccx,noccxv,
00688 c     d              natom, !nl,nclass,natom,nnc,
00689      d      nbloch,  nwhis,      !nlnmx,mdimx,
00690      i      iq,ngb,ngc,ngpmx,ngcmx, !ngb/=ngc+nbloch for smbasis()=T oct2005
00691      i      nqbze,nband,nqibz,
00692      o      rcxq,                ! rcxq is the accumulating variable for spins
00693      i      nolfco,zzr,nmbas_in, zcousq, !ppovl,nmbas2 is removed.,nmbas1 ppovlz,
00694      i      chipm,eibzmode,      !zloffd,
00695      i      nwgt(:,iqeibz),igx(:,:,iqeibz),igxt(:,:,iqeibz),ngrp, eibzsym(:,:,iqeibz),crpa)
00696          endif
00697
00698 !! reduction rcxq in the S-axis
00699          write(6,*) 'MPI__AllreduceSumS start'
00700          do i_reduction_npm=1,npm
00701            do i_reduction_nwhis=1,nwhis
00702              do i_reduction_nmbas2=1,nmbas2
00703                call mpi__allreducesums(
00704      .          rcxq(1,i_reduction_nmbas2,i_reduction_nwhis,i_reduction_npm), nmbas1)
00705              enddo
00706            enddo
00707          enddo
00708          write(6,*) 'MPI__AllreduceSumS end'
00709 !TIME1_190001 "x0kf_sym"
00710 !TIME0_200001 "HilbertTransformation"
```

```
00711 !! --- Hilbert transform.  Genrerate Real part from Imaginary part. ======
00712         if(allocated(zxq) ) deallocate(zxq,zxqi)
00713         allocate(zxq(nmbas1,nmbas2,nw_i:nw), zxqi(nmbas1,nmbas2,niw))
00714         write(6,'("goto dpsion5: nwhis nw_i niw nw_w nmbas1 nmbas2=",6i5)') nwhis,nw_i,nw,niw,nmbas1,nmbas2
00715         write(6,*)' -------- nmbas1,nmbas2=', nmbas1,nmbas2
00716         call dpsion5(frhis,nwhis, freq_r, nw, freq_i,niw, realomega, imagomega,
00717      i    rcxq, npm,nw_i, nmbas1,nmbas2, ! rcxq is alterd---used as work
00718      o    zxq, zxqi,
00719      i    chipm, schi,is,  ecut,ecuts)
00720         write(6,*)' --- end of dpsion5 ----',sum(abs(zxq)),sum(abs(zxqi))
00721         if(allocated(rcxq) ) deallocate(rcxq)
00722 !TIME1_200001 "HilbertTransformation"
00723 !! ===  RealOmega ===
00724         if (realomega) then
00725 !TIME0_210001 ralloc
00726           if (nspin == 1) zxq = 2d0*zxq !if paramagnetic, multiply x0 by 2
00727           nwmax = nw
00728           nwmin = nw_i
00729 !! prepare for iq0.
00730           iq0 = iq - nqibz
00731 c               if(newaniso2) then
00732 c$$$                 if( iq==1 ) then
00733 c$$$                    write(6,*)'open EPS0inv mpi=',MPI__rank
00734 c$$$                  ifepstinv = iopen('EPS0inv',0,-1,0)
00735 c$$$                  write(ifepstinv) ngb
00736 c$$$                 endif
00737           if(iqinit) then
00738             allocate( sk(ngb,nwmin:nwmax,nq0i),  sks(ngb,nwmin:nwmax,nq0i) )
00739             allocate( ski(ngb,niw,nq0i), sksi(ngb,niw,nq0i))
00740             iqinit=.false.
00741           endif
00742           allocate(epstilde(ngb,ngb))
00743           allocate(epstinv(ngb,ngb))
00744 c               endif
00745 !KINO              write(6,*)'kino: nwmin,nwmax,ngb=',nwmin,nwmax,ngb
00746           write(6, *)" === trace check for W-V === nwmin nwmax=",nwmin,nwmax
00747 !TIME1_210001 "ralloc"
00748 !TIME0_2200011 do1015
00749           do 1015 iw  = nwmin,nwmax
00750             frr= dsign(freq_r(abs(iw)),dble(iw))
00751             imode = 1
00752 c                if(newaniso2.and.iq<=nqibz) then !for mmmw
00753             if(iq<=nqibz) then  !for mmmw
00754               if(iq==1) then
00755                 ix=1
00756                 zw0(:,1)=0d0
00757                 zw0(1,:)=0d0
00758               else
00759                 ix=0
00760               endif
00761
00762 !!  Eqs.(37),(38) in PRB81 125102
00763               do igb1=ix+1,ngb
00764                 do igb2=ix+1,ngb
00765                   epstilde(igb1,igb2)= -vcousq(igb1)*zxq(igb1,igb2,iw)*vcousq(igb2)
00766                   if(igb1==igb2) epstilde(igb1,igb2)=1+epstilde(igb1,igb2)
00767                 enddo
00768               enddo
00769               epstinv(ix+1:ngb,ix+1:ngb)=epstilde(ix+1:ngb,ix+1:ngb)
00770               call matcinv(ngb-ix,epstinv(ix+1:ngb,ix+1:ngb))
00771 !TIME0_3000011 zweqzw0
00772
00773 c$$$  ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00774 c$$$  cmmm direct inversion vs. block inversion
00775 c$$$  if(iq>nqibz) then
00776 c$$$  c direct inversion
00777 c$$$  ix=0
00778 c$$$  do igb1=ix+1,ngb
00779 c$$$  do igb2=ix+1,ngb
00780 c$$$  epstilde(igb1,igb2)= -vcousq(igb1)*zxq(igb1,igb2,iw)*vcousq(igb2)
00781 c$$$  if(igb1==igb2) epstilde(igb1,igb2)=1+epstilde(igb1,igb2)
00782 c$$$  enddo
00783 c$$$  enddo
00784 c$$$  epstinv(ix+1:ngb,ix+1:ngb)=epstilde(ix+1:ngb,ix+1:ngb)
00785 c$$$  call matcinv(ngb-ix,epstinv(ix+1:ngb,ix+1:ngb))
00786 c$$$  do igb1=1+ix,ngb
00787 c$$$  do igb2=1+ix,ngb
00788 c$$$  zw0(igb1,igb2)= vcousq(igb1)*epstinv(igb1,igb2)*vcousq(igb2)
00789 c$$$  if(igb1==igb2) zw0(igb1,igb2)= zw0(igb1,igb2)-vcousq(igb1)*vcousq(igb2)
00790 c$$$  enddo
00791 c$$$  enddo
00792 c$$$  c               write(6,"('mmmmzp99x  ',i3,10(2d13.5,2x))") iw,zw0(1,1),zw0(2:10:3,1),zw0(63:70:3,1)
00793 c$$$  write(6,"('mmmmzp99x  ',i3,10(2d13.5,2x))") iw,1d0/epstinv(1,1),zw0(2:10:3,1),zw0(63:70:3,1)
00794 c$$$  c              write(6,"('mmmmzp99x  ',i3,10(2d13.5,2x))") iw,zw0(1,1),zw0(1,2:10:3),zw0(1,63:70:3)
00795 c$$$  c block inversion
00796 c$$$  ix=1
00797 c$$$  do igb1=ix+1,ngb
```

```
00798 c$$$  do igb2=ix+1,ngb
00799 c$$$  epstilde(igb1,igb2)= -vcousq(igb1)*zxq(igb1,igb2,iw)*vcousq(igb2)
00800 c$$$  if(igb1==igb2) epstilde(igb1,igb2)=1+epstilde(igb1,igb2)
00801 c$$$  enddo
00802 c$$$  enddo
00803 c$$$  epstinv(ix+1:ngb,ix+1:ngb)=epstilde(ix+1:ngb,ix+1:ngb)
00804 c$$$  call matcinv(ngb-ix,epstinv(ix+1:ngb,ix+1:ngb))
00805 c$$$  absq=sqrt(sum(q**2*tpioa**2))
00806 c$$$  sk(  1:ngb)= zxq(1,1:ngb,iw)
00807 c$$$  sks( 1:ngb)= zxq(1:ngb,1,iw)
00808 c$$$  w_k(1) =0d0
00809 c$$$  w_ks(1)=0d0
00810 c$$$  w_k( 2:ngb)= vcousq(2:ngb)*vcousq(1)*matmul(vcousq(1)*sk(2:ngb)*vcousq(2:ngb),epstinv(2:ngb,2:ngb))
00811 c$$$  w_ks(2:ngb)= vcousq(2:ngb)*vcousq(1)*matmul(epstinv(2:ngb,2:ngb),vcousq(1)*sks(2:ngb)*vcousq(2:ngb))
00812 c$$$  llw(iw,iq0)=
00813 c$$$  &             1d0
00814 c$$$  &            -vcousq(1)*sk(1)*vcousq(1) ! sk(1,1,iw)=sks(1,1,iw)=H of Eq.(40).
00815 c$$$  &            -vcousq(1)*vcousq(1)* sum( vcousq(2:ngb)*sk(2:ngb) *
      matmul(epstinv(2:ngb,2:ngb),sks(2:ngb)*vcousq(2:ngb)))
00816 c$$$  write(6,"('mmmmzwp99x ',i3,10(2d13.5,2x))") iw,llw(iw,iq0), !(1d0/llw(iw,iq0)-1d0)*vcousq(1)**2,
00817 c$$$  c     &                  w_k(2:10:3)/llw(iw,iq0), w_k(63:70:3)/llw(iw,iq0)
00818 c$$$  &                  w_ks(2:10:3)/llw(iw,iq0), w_ks(63:70:3)/llw(iw,iq0)
00819 c$$$  write(6,"('mmmmzwp99x ')")
00820 c$$$  endif
00821 c$$$  ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00822               do igb1=1+ix,ngb
00823                 do igb2=1+ix,ngb
00824                   zw0(igb1,igb2)= vcousq(igb1)*epstinv(igb1,igb2)*vcousq(igb2)
00825                   if(igb1==igb2) zw0(igb1,igb2)= zw0(igb1,igb2)-vcousq(igb1)*vcousq(igb2)
00826                 enddo
00827               enddo
00828 c$$$                  if(iq==1) write(ifepstinv) epstinv(ix+1:ngb,ix+1:ngb),iq,iw
00829               zw(1:ngb,1:ngb) = zw0
00830 !TIME1_3000011 "zweqzw0"
00831 !TIME0_3100011 tr_chkwrite
00832               if (mpi__roots)then
00833                 write(ifrcw, rec= iw-nw_i+1 ) zw !  WP = vsc-v
00834               endif
00835               call tr_chkwrite("freq_r iq iw realomg trwv=", zw, iw, frr,nblochpmx, nbloch,ngb,
      iq)
00836 !TIME1_3100011  "tr_chkwrite"
00837 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00838 c     if(iq>nqibz) then
00839 c     write(6,"('mmmmz99x ',i3,10(2d13.5,2x))") iw,zw0(1,1)+vcousq(1)**2,zw0(2:10:3,1),zw0(63:70:3,1)
00840 c     endif
00841 c     if(iq==1.or.iq>nqibz) then
00842 c     write(6,"('mmmz0  ',i3,10(2d13.5,2x))") iw,zxq(1,1,iw),zxq(1,2:10:3,iw),zxq(1,63:70:3,iw)
00843 c     write(6,"('mmmz0* ',i3,10(2d13.5,2x))") iw,zxq(1,1,iw),zxq(2:10:3,1,iw),zxq(63:70:3,1,iw)
00844 c     write(6,"('mmmmz99x ',i3,10(2d13.5,2x))") iw,zw0(1,1)+vcousq(1)**2,zw0(1,2:10:3),zw0(1,63:70:3)
00845 c     write(6,"('mmmzx  ',2i3,10(2d13.5,2x))") iq,iw,zxq(2,1,iw),zxq(2,2:10:3,iw),zxq(2,63:70:3,iw)
00846 c     write(6,"('mmmzx  ',2i3,10(2d13.5,2x))") iq,iw,zxq(3,1,iw),zxq(3,2:10:3,iw),zxq(3,63:70:3,iw)
00847 c     write(6,"('mmmzxs ',2i3,10(2d13.5,2x))") iq,iw,zxq(1,1,iw),zxq(2:10:3,1,iw),zxq(63:70:3,1,iw)
00848 c     write(6,"('mmmzxs ',2i3,10(2d13.5,2x))") iq,iw,zxq(1,2,iw),zxq(2:10:3,2,iw),zxq(63:70:3,2,iw)
00849 c     write(6,"('mmmmzee',2i3,10(2d13.5,2x))")iq,iw,epstilde(2,2),epstilde(2,2:10:3),epstilde(2,63:70:3)
00850 c     write(6,"('mmmmzee',2i3,10(2d13.5,2x))")iq,iw,epstilde(3,2),epstilde(3,2:10:3),epstilde(3,63:70:3)
00851 c     endif
00852 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00853             endif
00854
00855 c                 if(newaniso2.and.iq>nqibz) then
00856             if(iq>nqibz) then
00857 !! Full inversion to calculalte eps with LFC.
00858               vcou1 = fourpi/sum(q**2*tpioa**2) ! --> vcousq(1)**2!  !fourpi/sum(q**2*tpioa**2-eee)
00859               if(localfieldcorrectionllw()) then
00860                 ix=0
00861                 do igb1=ix+1,ngb
00862                   do igb2=ix+1,ngb
00863                     if(igb1==1.and.igb2==1) then
00864                       epstilde(igb1,igb2)= 1d0 - vcou1*zxq(1,1,iw)
00865                       cycle
00866                     endif
00867                     epstilde(igb1,igb2)= -vcousq(igb1)*zxq(igb1,igb2,iw)*vcousq(igb2)
00868                     if(igb1==igb2) then
00869                       epstilde(igb1,igb2)=1d0 + epstilde(igb1,igb2)
00870                     endif
00871                   enddo
00872                 enddo
00873 c !TIME0
00874                 epstinv(ix+1:ngb,ix+1:ngb)=epstilde(ix+1:ngb,ix+1:ngb)
00875                 call matcinv(ngb-ix,epstinv(ix+1:ngb,ix+1:ngb))
00876                 llw(iw,iq0)= 1d0/epstinv(1,1)
00877 c !TIME1 "end of matcinv_epstinv" !this gives wrong message, probably
00878 c         because of a bug of !TIME1 processing for MPI mode.
00879               else
00880 c commentout block inversion
00881 c$$$                  sk  (1:ngb,iw,iq0)= zxq(1,1:ngb,iw)
00882 c$$$                  sks (1:ngb,iw,iq0)= zxq(1:ngb,1,iw)
```

```
c$$$c                     sks (1:ngb,iw,iq0)= zxq(2,1:ngb,iw) !nmbas1=2 see z1stcol in x0kf_v4h.
c$$$                     vcou1 = fourpi/sum(q**2*tpioa**2) ! --> vcousq(1)**2!
 !fourpi/sum(q**2*tpioa**2-eee)
                     llw(iw,iq0)= 1d0 - vcou1*zxq(1,1,iw)
                  endif
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cmmmm
c     write(6,"('mmmw0  ',i3,10(2d13.5,2x))") iw,zxq(1,1,iw),zxq(1,2:10:3,iw),zxq(1,63:70:3,iw)
c     write(6,"('mmmw0* ',i3,10(2d13.5,2x))") iw,zxq(1,1,iw),zxq(2,2:10:3,iw),zxq(2,63:70:3,iw)
c     write(6,"('mmmmw99x ',i3,10(2d13.5,2x))") iw,fourpi/sum(q**2*tpioa**2)/llw(iw,iq0),
c     &                  w_k(2:10:3)/llw(iw,iq0),w_k(63:70:3)/llw(iw,iq0)
c     write(6,"('mmmmw99x ',i3,10(2d13.5,2x))") iw,llw(iw,iq0),
c     &                  w_ks(2:10:3)/llw(iw,iq0),w_ks(63:70:3)/llw(iw,iq0)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!TIME0_3200011 writeiqiwreal
                  write(6,"('iq iw_real eps(wLFC) eps(noLFC)',i4,i4,2f10.4,2x,2f10.4)")iq,iw,llw(iw,iq0),1d0-
vcou1*zxq(1,1,iw)
!TIME1_3200011 "writeiqiwreal"
                endif

c$$$                   if(.not.newaniso2) then ! Original mode
c$$$                      call rx( 'not checked here')
c$$$c     call wcf( ngb, vcoul, zxq(1,1,iw), imode, zw0)
c$$$                   endif

c$$$  !!... a debug mode
c$$$  write(6,"('hhh --- EigenValues for Im( W) --------')")
c$$$  allocate(ebb(ngb))
c$$$  call diagcvh2( (zw0-transpose(dconjg(zw0)))/2d0/img, ngb, ebb)
c$$$  do ii=1,ngb
c$$$  if( abs(ebb(ii))>1d-8 .and. ebb(ii)>0) then
c$$$  write(6, "('hhhIWq : iw ii eb=',2i4,d13.5)") iw, ii, ebb(ii)
c$$$  else
c$$$  write(6, "('hhhIWqxxx : iw ii eb=',2i4,d13.5)") iw, ii, ebb(ii)
c$$$  endif
c$$$  enddo
c$$$  deallocate(ebb)

c     if(newaniso2.and.iq>nqibz) then
c     c                 zw(1:ngb,1:ngb) = 0d0
c     c                 write(ifrcw, rec=((iq-iqxini)*(nw-nw_i+1)+ iw-nw_i+1 ) ) zw   !  WP = vsc-v
c     else
c     zw(1:ngb,1:ngb) = zw0
c     c                 write(ifrcw, rec=((iq-iqxini)*(nw-nw_i+1)+ iw-nw_i+1 ) ) zw   !  WP = vsc-v
c     write(ifrcw, rec= iw-nw_i+1 ) zw   !  WP = vsc-v
c     call tr_chkwrite("freq_r iq iw realomg trwv=", zw, iw, frr,nblochpmx, nbloch,ngb,iq)
c     endif
 1015      continue              !iw
!TIME1_2200011 "do1015"

c     if(newaniso2) then
c     if(allocated(sk)) deallocate(sk,sks,w_k,w_ks)
c     endif
           if( allocated(zzr) ) deallocate(zzr)
         endif
!! === RealOmega end ===

!! === ImagOmega ===
!TIME0_230001 imagomega
         if (imagomega) then
           write(6,*)' goto imag omega'
           if (nspin == 1) zxqi = 2d0*zxqi ! if paramagnetic, multiply x0 by 2
           imode=1
           do 1016 iw  = 1,niw
c                   if( newaniso2 .and. iq<=nqibz ) then
             if( iq<=nqibz ) then
!!  Eqs.(37),(38) in PRB81 125102
               if(iq==1) then
                 ix=1
                 zw0(:,1)=0d0
                 zw0(1,:)=0d0
               else
                 ix=0
               endif
               do igb1=ix+1,ngb
                 do igb2=ix+1,ngb
                   epstilde(igb1,igb2)= -vcousq(igb1)*zxqi(igb1,igb2,iw)*vcousq(igb2)
                   if(igb1==igb2) epstilde(igb1,igb2)=1+epstilde(igb1,igb2)
                 enddo
               enddo
               epstinv=epstilde
               call matcinv(ngb-ix,epstinv(ix+1:ngb,ix+1:ngb))
               do igb1=ix+1,ngb
                 do igb2=ix+1,ngb
                   zw0(igb1,igb2)= vcousq(igb1)*epstinv(igb1,igb2)*vcousq(igb2)
                   if(igb1==igb2) zw0(igb1,igb2)= zw0(igb1,igb2)-vcousq(igb1)*vcousq(igb2)
                 enddo
```

```
00968               enddo
00969 c$$$                    if(iq==1) write(ifepstinv) epstinv(ix+1:ngb,ix+1:ngb),iq,iw
00970
00971               zw(1:ngb,1:ngb) = zw0 ! zw(nblochpmx,nblochpmx)
00972               if (mpi__roots) then
00973                 write(ifrcwi, rec= iw)  zw !  WP = vsc-v
00974               endif
00975               call tr_chkwrite("freq_i iq iw imgomg trwv=",zw,iw,freq_i(iw),nblochpmx,nbloch,ngb
      ,iq)
00976             endif
00977
00978 c                  if( newaniso2.and.iq>nqibz) then
00979             if(iq>nqibz) then
00980 !! Full inversion to calculalte eps with LFC.
00981               vcou1 = fourpi/sum(q**2*tpioa**2) ! --> vcousq(1)**2!  !fourpi/sum(q**2*tpioa**2-eee)
00982               if(localfieldcorrectionllw()) then
00983                 ix=0
00984                 do igb1=ix+1,ngb
00985                   do igb2=ix+1,ngb
00986                     if(igb1==1.and.igb2==1) then
00987                       epstilde(igb1,igb2)= 1d0 - vcou1*zxqi(1,1,iw)
00988                       cycle
00989                     endif
00990                     epstilde(igb1,igb2)= -vcousq(igb1)*zxqi(igb1,igb2,iw)*vcousq(igb2)
00991                     if(igb1==igb2) then
00992                       epstilde(igb1,igb2)=1d0 + epstilde(igb1,igb2)
00993                     endif
00994                   enddo
00995                 enddo
00996                 epstinv(ix+1:ngb,ix+1:ngb)=epstilde(ix+1:ngb,ix+1:ngb)
00997                 call matcinv(ngb-ix,epstinv(ix+1:ngb,ix+1:ngb))
00998                 llwi(iw,iq0)= 1d0/epstinv(1,1)
00999               else
01000 c commentout block inversion
01001 c$$$                    skI  (1:ngb,iw,iq0)= zxqi(1,1:ngb,iw)
01002 c$$$c                    sksI (1:ngb,iw,iq0)= zxqi(2,1:ngb,iw) !nmbas1=2 see z1stcol in x0kf_v4h.
01003 c$$$                    sksI (1:ngb,iw,iq0)= zxqi(1:ngb,1,iw) !nmbas1=2 see z1stcol in x0kf_v4h.
01004 c$$$                    vcou1  = fourpi/sum(q**2*tpioa**2) ! test-->vcousq(1)**2
      !fourpi/sum(q**2*tpioa**2-eee)
01005 c$$$                    vcou1sq= sqrt(vcou1)
01006 c$$$!! llwI without LFC. LFC contribution is added in
01007               llwi(iw,iq0)=  1d0 -vcou1*zxqi(1,1,iw) !- vcou1sq*sum( skI(2:ngb) *
      w_ksI(2:ngb)*vcousq(2:ngb) )
01008               endif
01009               write(6,"('iq iw_img eps(wLFC) eps(noLFC)',i4,i4,2f10.4,2x,2f10.4)")iq,iw,llwi(iw,iq0),1d0-
      vcou1*zxqi(1,1,iw)
01010             endif
01011
01012  1016     continue
01013 c               if(newaniso2) then
01014 c$$$                 if(iq==1) ifepstinv = iclose('EPS0inv') !iq==1 close write mode.
01015           deallocate(epstinv)
01016           if(allocated(epstilde)) deallocate(epstilde)
01017 c               endif
01018         endif
01019 !! === ImagOmega end ===
01020 !TIME1_230001 "imagomega"
01021
01022 c     1002 continue  ! end of frequency block-loop
01023         if(allocated(vcou1)) deallocate(vcou1)
01024         if(allocated(zw0)) deallocate(zw0)
01025         if(allocated(zxq )) deallocate(zxq)
01026         if(allocated(zxqi)) deallocate(zxqi)
01027
01028         if (mpi__roots) then
01029           ifrcwi = iclose('WVI.'//charnum5(iq))
01030           ifrcw  = iclose('WVR.'//charnum5(iq))
01031         endif
01032 !!
01033  1001 continue
01034 !TIME1_170001 "do1001"
01035 c============end of loop over q point ================================
01036 c=======================================================================
01037       call mpi__barrier()
01038 !TIME0_24001 w0mpi
01039 !! === Recieve llw and llwI at node 0, where q=0(iq=1) is calculated. ===
01040 c         write(6,*)'MPI__sizerrr=',MPI__size,MPI__rank,MPI__root,MPI__size,nqibz,iqxend
01041 ccccccccccccccccccccccccc
01042 CYY!$OMP parallel
01043 CYY!$OMP master
01044 c      write(6,*)' eeeeeeeeeeeeeeeee111 MPI__rank=',MPI__rank
01045       if(mpi__size/=1) then
01046         do iq=nqibz+1,iqxend
01047           iq0 = iq - nqibz
01048 c     write(6,*)' iq iq0 mpi_rank mpi_ranktab(iq)=',iq,
      iq0,MPI__rank,MPI__ranktab(iq),MPI__root,nw,nw_i,niw
01049           if(mpi__qranktab(iq)/=0) then !jan2012
```

```
01050            if(mpi__qranktab(iq) == mpi__rankq) then
01051 c     write(6,*)' mpi_send iq from',iq,MPI__ranktab(iq)
01052 c     write(6,*)' send llw sum=',sum(abs(llw(:,iq0))),nw,nw_i
01053 c     do i=nw_i,nw
01054 c     write(6,*)'sendxxx',i,llw(i,iq0)
01055 c     enddo
01056 c     write(6,*)' send llwI sum=',sum(abs(llwI(:,iq0))),niw
01057               dest=0
01058               call mpi__dblecomplexsendq(llw(nw_i,iq0),(nw-nw_i+1),dest)
01059               call mpi__dblecomplexsendq(llwi(1,iq0),niw,dest)
01060            elseif(mpi__rootq) then
01061 c     write(6,*)' mpi_recv iq from',iq,MPI__ranktab(iq),nw,nw_i,niw
01062               src=mpi__qranktab(iq)
01063               call mpi__dblecomplexrecvq(llw(nw_i,iq0),(nw-nw_i+1),src)
01064               call mpi__dblecomplexrecvq(llwi(1,iq0),niw,src)
01065 c     do i=nw_i,nw
01066 c     write(6,*)'recivxxx',i,llw(i,iq0)
01067 c     enddo
01068 c     write(6,*)' recv llw sum=',sum(abs(llw(:,iq0))),nw,nw_i
01069 c     write(6,*)' recv llwI sum=',sum(abs(llwI(:,iq0))),niw
01070            endif
01071          endif
01072        enddo
01073       endif
01074 c      write(6,*)' eeeeeeeeeeeeeeee222 MPI__rank=',MPI__rank
01075 CYY!$OMP end master
01076 CYY!$OMP end parallel
01077 !TIME1_24001 "w0mpi"
01078
01079 c commentout block inversion
01080 c$$$!! Add LFC (local field correction) to llw and llwI
01081 c$$$         if(newaniso2 .and. MPI__rank == 0 ) then ! only on root node
01082 c$$$           iq=1 !for q=0
01083 c$$$           vcoudfile='Vcoud.'//charnum5(iq)
01084 c$$$           ifvcoud = iopen(trim(vcoudfile),0,-1,0)
01085 c$$$           read(ifvcoud) ngb0
01086 c$$$           read(ifvcoud) qvv
01087 c$$$           if(sum(abs(qvv))>1d-10) then
01088 c$$$              write(6,*)'qvv =',qvv
01089 c$$$              stop 'hx0fp0: qvv/=0 hvcc is not consistent'
01090 c$$$           endif
01091 c$$$           if(allocated(zcousq0)) deallocate( zcousq0,vcousq0 )
01092 c$$$           allocate( zcousq0(ngb0,ngb0),vcousq0(ngb0))
01093 c$$$           read(ifvcoud) vcousq0
01094 c$$$           read(ifvcoud) zcousq0
01095 c$$$           idummy=iclose(trim(vcoudfile))
01096 c$$$           vcousq=sqrt(vcousq)
01097 c$$$           allocate(epstinv(ngb0,ngb0),w_k(ngb0),w_ks(ngb0),w_kI(ngb0),w_ksI(ngb0),eemat(ngb0,ngb0))
01098 c$$$
01099 c$$$           do iq0=1,nq0i
01100 c$$$             iq = iq0 + nqibz
01101 c$$$             q = qibze(:,iq)
01102 c$$$
01103 c$$$             vcoudfile='Vcoud.'//charnum5(iq)
01104 c$$$             ifvcoud = iopen(trim(vcoudfile),0,-1,0)
01105 c$$$             read(ifvcoud) ngb
01106 c$$$             read(ifvcoud) qvv
01107 c$$$             if(sum(abs(qvv-q))>1d-10) then
01108 c$$$              write(6,*)'qvv =',qvv
01109 c$$$              stop 'hx0fp0: qvv/=0 hvcc is not consistent'
01110 c$$$             endif
01111 c$$$             if(allocated(zcousq)) deallocate(zcousq)
01112 c$$$             if(allocated(vcousq)) deallocate(vcousq)
01113 c$$$             allocate( zcousq(ngb0,ngb0),vcousq(ngb0))
01114 c$$$             read(ifvcoud) vcousq
01115 c$$$             read(ifvcoud) zcousq
01116 c$$$             idummy=iclose(trim(vcoudfile))
01117 c$$$             vcousq=sqrt(vcousq)
01118 c$$$
01119 c$$$             ifepstinv = iopen('EPS0inv',0,0,0)
01120 c$$$             read(ifepstinv) ngb
01121 c$$$
01122 c$$$              ngc=ngb-nbloch
01123 c$$$              if(allocated(ppovlz)) deallocate(ppovlz)
01124 c$$$              if(allocated(ppovl)) deallocate(ppovl)
01125 c$$$              allocate(ppovl(ngc,ngc),ppovlz(ngb,ngb))
01126 c$$$              call readppovl0(q,ngc,ppovl) !q was qq
01127 c$$$              ppovlz(1:nbloch,:) = zcousq(1:nbloch,:)
01128 c$$$              ppovlz(nbloch+1:nbloch+ngc,:) = matmul(ppovl,zcousq(nbloch+1:nbloch+ngc,:))
01129 c$$$
01130 c$$$!  eemat: Z\mu_i(\bfk=0)^* <i|j> Z\nu_j(\bfk)
01131 c$$$             eemat =matmul(transpose(dconjg(zcousq0)),matmul(ppovlz,zcousq))
01132 c$$$             vcou1  = fourpi/sum(q**2*tpioa**2) ! test-->vcousq(1)**2 !fourpi/sum(q**2*tpioa**2-eee)
01133 c$$$             vcou1sq = vcou1**.5
01134 c$$$             write(6,*)
01135 c$$$
01136 c$$$             do iw=nwmin,nwmax
```

```
01137 c$$$                read(ifepstinv) epstinv(2:ngb,2:ngb),iqx,iwx
01138 c$$$                epstinv(2:ngb,2:ngb) = matmul( transpose(dconjg(eemat(2:ngb,2:ngb))),
01139 c$$$     &                                 matmul(epstinv(2:ngb,2:ngb),eemat(2:ngb,2:ngb)) )
01140 c$$$                if(iw/=iwx) then
01141 c$$$                write(6,*)'iw iwx=',iw,iwx
01142 c$$$                stop 'hx0fp0_sc: iw/=iwx'
01143 c$$$                endif
01144 c$$$                w_k(2:ngb) = vcoulsq*matmul( epstinv(2:ngb,2:ngb), sk(2:ngb,iw,iq0)*vcousq(2:ngb))
01145 c$$$                epslfc = -vcoulsq*sum( sks(2:ngb,iw,iq0) * w_k(2:ngb) *vcousq(2:ngb) )
01146 c$$$                llw(iw,iq0) = llw(iw,iq0)  + epslfc
01147 c$$$                write(6,"('eps(on real) iq iw',2i4,2f9.3,2x,2f9.3)") iq0,iw,
       llw(iw,iq0)-epslfc,llw(iw,iq0)
01148 c$$$              enddo
01149 c$$$              do iw=1,niw
01150 c$$$                read(ifepstinv) epstinv(2:ngb,2:ngb),iqx,iwx
01151 c$$$                if(iw/=iwx) then
01152 c$$$                 write(6,*)'iw iwx=',iw,iwx
01153 c$$$                 stop 'hx0fp0_sc: iw/=iwx'
01154 c$$$                endif
01155 c$$$                w_kI(2:ngb)= vcoulsq*matmul( epstinv(2:ngb,2:ngb), skI(2:ngb,iw,iq0)*vcousq(2:ngb))
01156 c$$$                epslfc=- vcoulsq*sum( sksI(2:ngb,iw,iq0)* w_kI(2:ngb)*vcousq(2:ngb) )
01157 c$$$                llwI(iw,iq0)= llwI(iw,iq0)+epslfc
01158 c$$$                write(6,"('eps(on img ) iq iw',2i4,2f9.3,2x,2f9.3)")iq0,iw,
       llwI(iw,iq0)-epslfc,llwI(iw,iq0)
01159 c$$$              enddo
01160 c$$$              ifepstinv = iclose('EPS0inv')
01161 c$$$            enddo
01162 c$$$          endif
01163
01164
01165 !! == W(0) divergent part and W(0) non-analytic constant part.==
01166  1191 continue
01167 !TIME0_40001 WVRI
01168 c         if(newaniso2 .and. MPI__rank == 0 ) then ! MIZUHO-IR only on root node
01169       if(mpi__rank == 0 ) then  ! MIZUHO-IR only on root node
01170
01171         if(ixc==1011) then      !this is only for test.
01172           ifw0w0i = iopen('W0W0I',0,-1,0)
01173           read(ifw0w0i) nw_i,nw,niw,nq0i
01174           write(6,*)'w0w0i: n=',nw_i,nw,niw,nq0i
01175           read(ifw0w0i) llw(nw_i:nw,1:nq0i)
01176           read(ifw0w0i) llwi(1:niw,1:nq0i)
01177 c            read(ifw0w0i) w0(nw_i:nw)
01178 c            read(ifw0w0i) w0i(1:niw)
01179           ifw0w0i = iclose('W0W0I')
01180         endif
01181
01182         write(6,*)
01183         write(6,*)' ==== newaniso2 mode W(0) divergent part ==== '
01184 !! == W(0) divergent part ==
01185 !! getw0 routine...
01186 !!NOTE: we usually only use lxklm=1 --> this should be stable.
01187 !!  EPSwklm is generated in gwsrc/mkqg.F
01188         ifidmlx = iopen('EPSwklm',0,0,0)
01189         read(ifidmlx) nq0i,lxklm
01190         allocate( dmlx(nq0i,9))
01191         allocate( epinvq0i(nq0i,nq0i),epinv(3,3,nq0i))
01192         nlxklm=(lxklm+1)**2
01193         allocate( wklm(nlxklm))
01194         read(ifidmlx) dmlx, epinv,epinvq0i
01195         read(ifidmlx) wklm
01196         ifidmlx = iclose('EPSwklm')
01197 !! starting from llw(iw,iq0),llwI(iw,iq0)
01198 !! === <e|L|e> (eq.36 in Friedrich paper) is expanded in YL -->stored in llwyl. ===
01199         allocate(w0(nw_i:nw),w0i(niw))
01200 c     write(6,*)' goto getw0 nq0i epinvq0i=',nq0i,epinvq0i
01201 !! wbz(1) is the weight for q=0 = 1/(n1*n2*n3)
01202 c           write(6,*)'wbz=',wbz
01203         call getw0(llw, nw_i,nw,nq0i,dmlx,epinvq0i,wklm,wbz(1), lxklm,  q0i,epinv,w0)
01204         call getw0(llwi,1,niw  ,nq0i,dmlx,epinvq0i,wklm,wbz(1), lxklm,  q0i,epinv,w0i)
01205         if(ixc/=1011) then
01206           ifw0w0i = iopen('W0W0I',0,-1,0)
01207           write(ifw0w0i) nw_i,nw,niw,nq0i
01208           write(ifw0w0i) llw(nw_i:nw,1:nq0i)
01209           write(ifw0w0i) llwi(1:niw,1:nq0i)
01210           write(ifw0w0i) w0(nw_i:nw)
01211           write(ifw0w0i) w0i(1:niw)
01212           ifw0w0i = iclose('W0W0I')
01213         endif
01214
01215         do i=nw_i,nw
01216           write(6,"('w0 =',i4,2f13.4)")i,w0(i)
01217         enddo
01218         do i=1,niw
01219           write(6,"('w0i=',i4,2f13.4)")i,w0i(i)
01220         enddo
01221 c     write(6,*)'sumcheck w0,w0i=',sum(abs(w0)),sum(abs(w0i))
```

```
01222 !! === w0,w0i are stored to zw for q=0 ===
01223 !! === w_ks*wk are stored to zw for iq >nqibz ===
01224 ! We assume iq=1 is for rank=0
01225         do iq = 1,1             !iq=1 only 4pi/k**2 /eps part only ! iq = iqxini,iqxend
01226 c               if( .not. MPI__task(iq) ) cycle
01227           q = qibze(:,iq)
01228           do ircw=1,2
01229             if    (ircw==1) then
01230               nini=nw_i
01231               nend=nw
01232               ifrcwx = iopen('WVR.'//charnum5(iq),0,-1,mrecl)
01233             elseif(ircw==2) then;  nini=1;      nend=niw;
01234               ifrcwx = iopen('WVI.'//charnum5(iq),0,-1,mrecl)
01235             endif
01236             do iw=nini,nend
01237 c     if(iq<=nqibz) read(ifrcwx, rec=((iq-iqxini)*(nend-nini+1)+ iw-nini+1 ) ) zw !(1:ngb,1:ngb)
01238               read(ifrcwx, rec= iw-nini+1 ) zw !(1:ngb,1:ngb)
01239               if( iq==1 ) then
01240                 if(ircw==1) zw(1,1) = w0(iw)
01241                 if(ircw==2) zw(1,1) = w0i(iw)
01242               endif
01243 c     write(ifrcwx,rec=((iq-iqxini)*(nend-nini+1)+ iw-nini+1 ) ) zw !(1:ngb,1:ngb)
01244               write(ifrcwx,rec=iw-nini+1) zw !(1:ngb,1:ngb)
01245             enddo
01246             if    (ircw==1) then
01247               ifrcwx = iclose('WVR.'//charnum5(iq))
01248             elseif(ircw==2) then
01249               ifrcwx = iclose('WVI.'//charnum5(iq))
01250             endif
01251           enddo
01252         end do
01253       endif
01254       is = iclose('hbe.d')
01255 !TIME1_40001 "WVRI"
01256 !TIME1_1001 "ProgAll"
01257 !TIMESHOW
01258       call cputid(0)
01259       write(6,*) '--- end of hx0fp0_sc --- irank=',mpi__rank
01260       call flush(6)
01261       call mpi__finalize
01262       if(ixc==11) call rx0( ' OK! hx0fp0_sc ixc=11 Sergey F. mode')
01263       if(ixc==1011) call rx0( ' OK! hx0fp0_sc ixc=1011 W0W0Ionly')
01264       end program hx0fp0_sc
01265
01266
01267 C=====================================================================
01268       subroutine tr_chkwrite(tagname,zw,iw,freqq,nblochpmx,nbloch,ngb,iq)
01269       implicit none
01270       integer:: nblochpmx,nbloch,ngb,iw,i,iq
01271       complex(8):: zw(nblochpmx,nblochpmx),trwv,trwv2
01272       real(8):: freqq
01273       character*(*)::tagname
01274       trwv=0d0
01275       do i = 1,nbloch
01276         trwv = trwv + zw(i,i)
01277       enddo
01278       trwv2 = 0d0
01279       do i = 1,ngb
01280          trwv2 = trwv2 + zw(i,i)
01281       enddo                     !  write(6,'(" realomg trwv=",2i6,4d22.14)') iq,iw,trwv(iw),trwv2(iw)
01282       write(6,'(a,f10.4,2i5,4d22.14)')tagname,freqq,iq,iw,trwv,trwv2
01283 c     do i = 1,ngb
01284 c     write(6,'("iii i=",i4,a,f10.4,2i5,4d22.14)')i,tagname,freqq,iq,iw,zw(i,i)
01285 c     enddo
01286       end
01287
01288
```

4.37 main/qg4gw.m.F File Reference

Functions/Subroutines

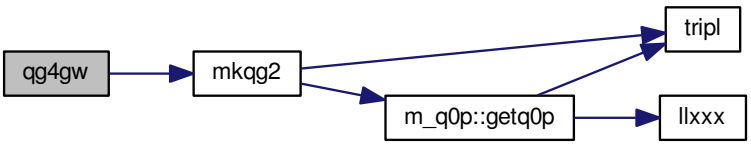
- program qg4gw

4.37.1 Function/Subroutine Documentation

4.37.1.1 program qg4gw ( )

Definition at line 1 of file qg4gw.m.F.

Here is the call graph for this function:



4.38 qg4gw.m.F

```
00001       program qg4gw
00002 !> Generate required q+G vectors and so on for GW calculations.
00003 !! input file
00004 !!   LATTC: contains these lattice informations;
00005 !!    alat       : lattice constant in a.u.
00006 !!    QpGcut_psi : maxmum of |q+G| in a.u. in the expansion of the eigenfunction.
00007 !!    QpGcut_Cou : maxmum of |q+G| in a.u. in the expansion of the Coulomb matrix.
00008 !!    plat(1:3,1): 1st primitive translation vector in the unit of alat
00009 !!    plat(1:3,2): 2nd primitive translation vector
00010 !!    plat(1:3,3): 3rd primitive translation vector
00011 !!   SYMOPS file : include point group operation. See sample.
00012 !!
00013 !! outtput files:
00014 !!   QGpsi: q and G vector for the eigenfunction
00015 !!   QGcou: q and G vector for the Coulomb matrix
00016 !!   Q0P  : offset Gamma point around \Gamma points
00017 !!   EPSwklm : offset Gamma method.
00018 !! and so on.
00019 !!ccc   Qmtet: q vectors for devided-tetrahedron.
00020 !! -------------------------
00021 !! For exampl,e QGpsi is written in the following manner. See mkqg2 in mkqg.F
00022 !!     open(ifiqg, file='QGpsi',)
00023 !!      write(ifiqg ) nqnum,ngpmx,QpGcut_psi,nqbz,nqi,imx,nqibz
00024 !!      allocate( ngvecprev(-imx:imx,-imx:imx,-imx:imx) ) !inverse mapping table
00025 !!      ngvecprev=9999
00026 !!      ngveccrev=9999
00027 !!      do iq = 1, nqnum
00028 !!         q = qq(1:3,iq)
00029 !!         write (ifiqg) q, ngp, irr(iq) ! irr=1 for irreducible points
00030 !!         do ig = 1,ngp
00031 !!            nnn3 = ngvecp(1:3, ig)
00032 !!            ngvecprev( nnn3(1), nnn3(2),nnn3(3)) = ig
00033 !!         enddo
00034 !!         write (ifiqg)  ngvecp,ngvecprev !ngvecprev is added on mar2012takao
00035 !!         do ig = 1,ngc
00036 !!            nnn3 = ngvecc(1:3, ig)
00037 !!            ngveccrev( nnn3(1), nnn3(2),nnn3(3)) = ig
00038 !!         enddo
00039 !!       enddo
00040 !!     close(ifiqg)
00041 !! -------------------------------------------------
00042 !! True q (in a.u. in Cartesian coordinate) is given by
00043 !!    q(1:3)     = 2*pi/alat * q(1:3)
00044 !! True q+G is given by
00045 !!    qplusG(1:3,igp) = 2*pi/alat * (q + matmul(qlat * ngvec(1:3,igp))), for igp=1,ngp
00046 !! ---------------------------------------------------------------------
00047       use keyvalue,only: getkeyvalue
00048       implicit none
00049       integer(4) ::n1q,n2q,n3q,ifiqg,ifiqgc,ifigw0,ngrp,ifi,i,ig,iq0pin,idummy
00050       real(8) ::  alat,qpgcut_psi, qpgcut_cou,dummy ,plat(3,3)
00051       real(8) :: volum,q0(3),qlat0(3,3),qpgx2,a1,a2,pi,unit !,QpGx1
00052       real(8),allocatable :: symops(:,:,:)
00053       character(len=150):: recrdxxx
00054       character(len=10) :: keyw1='unit_2pioa',keyw2
00055       logical ::unit2=.false. !  readgwinput,
00056       integer(4)::nnn(3),ret
```

```
00057       integer(4):: verbose,q0pchoice,wgtq0p    !,normcheck !version,
00058       logical:: gausssmear,keepeigen,core_orth,ldummy !keepppovl,
00059       integer(4):: iq0pinxxx ,ifile_handle
00060       pi= 4d0* atan(1d0)
00061       call cputid(0)
00062       write(6,*)' qg4gw: Generate Q0P->1; Readin Q0P->2; band mode->3; SW(chipm)->4'
00063       write(6,*)'        Generate Q0P->101(old offset Gamma)'
00064       read (5,*) iq0pin
00065       call headver('qg4gw',iq0pin)
00066       write(6,*) ' mode iq0pin = ',iq0pin
00067       if(iq0pin==-100.or.iq0pin==1.or.iq0pin==2.or.iq0pin==3.or.iq0pin==4.or.iq0pin==101) then
00068         continue
00069       else
00070         call rx( 'Not allowed iq0pin')
00071       endif
00072 !! Generate templeta of GWinput for iq0pin=-100
00073       if(iq0pin==-100) then
00074          call conv2gwinput()
00075          call rx0( ' OK! qg4gw mode=-100 to generate GWinput')
00076       endif
00077       idummy=q0pchoice()
00078       write(6,"(' q0pchoice() = ',i4)")  q0pchoice()
00079
00080       ifi=ifile_handle()
00081       open (ifi, file='LATTC')
00082       read(ifi,*) alat
00083       read(ifi,*) plat(1:3,1)
00084       read(ifi,*) plat(1:3,2)
00085       read(ifi,*) plat(1:3,3)
00086       read(ifi,*) !dummy
00087       close(ifi)
00088 !! --- readin SYMOPS. point group operations. r'=matmul(symops(:,:),r) for any ig.
00089       ifi=ifile_handle()
00090       open (ifi, file='SYMOPS')
00091       read(ifi,*) ngrp
00092       write(6,*) ' SYMOPS ngrp=',ngrp
00093       allocate(symops(3,3,ngrp))
00094       do ig = 1,ngrp
00095         read(ifi,*)
00096         do i=1,3
00097           read(ifi,*) symops(i,1:3,ig)
00098         enddo
00099       enddo
00100       close(ifi)
00101 !! --- check write
00102       write(6,*) ' --- primitive vectors ---'
00103       write(6,"(' unit(a.u.) alat  =',f13.6 )") alat
00104       write(6,"(' primitive_1 =',3f13.6)") plat(1:3,1)
00105       write(6,"(' primitive_2 =',3f13.6)") plat(1:3,2)
00106       write(6,"(' primitive_3 =',3f13.6)") plat(1:3,3)
00107       write(6,*) ' --- point group operations --- '
00108       do ig = 1,ngrp
00109         print *, ' ig=',ig
00110         do i=1,3
00111           write(6,"(3f14.6)") symops(i,1:3,ig)
00112         enddo
00113       enddo
00114 !! --- Readin GWinput
00115       call getkeyvalue("GWinput", "n1n2n3", nnn,3)
00116       n1q=nnn(1); n2q=nnn(2); n3q = nnn(3)
00117       call getkeyvalue("GWinput", "QpGcut_psi",qpgx2)
00118       call getkeyvalue("GWinput", "QpGcut_cou",qpgcut_cou)
00119       call getkeyvalue("GWinput", "unit_2pioa",unit2)
00120       if(unit2) then
00121         unit = 2d0*pi/alat
00122         qpgx2     = qpgx2      *unit
00123         qpgcut_cou= qpgcut_cou *unit
00124       endif
00125       qpgcut_psi = qpgx2
00126       write(6,"(' --- k points for GW from GWinput =',3i3)") n1q,n2q,n3q
00127       write(6,"(' ---  |k+G| < QpG(psi) QpG(Cou)=',2d13.6)") qpgcut_psi, qpgcut_cou
00128       ifiqg  = 401
00129       ifiqgc = 402
00130       iq0pinxxx=iq0pin
00131       if(iq0pin==4) then
00132          iq0pinxxx=2
00133          qpgcut_psi=0d0
00134          qpgcut_cou=0d0
00135       endif
00136       open(ifiqg ,file='QGpsi',form='unformatted')
00137       open(ifiqgc,file='QGcou',form='unformatted')
00138       call mkqg2(alat,plat,symops,ngrp,n1q,n2q,n3q,iq0pinxxx,
00139      &     qpgcut_psi, qpgcut_cou, ifiqg, ifiqgc)
00140       write(6,*) ' OK! End of qg4gw '
00141       if(iq0pin ==1) call rx0( ' OK! qg4gw mode=1 normal mode')
00142       if(iq0pin ==2) call rx0( ' OK! qg4gw mode=2 Readin Q0P mode')
00143       if(iq0pin ==3) call rx0( ' OK! qg4gw mode=3 band-plot mode')
```

```
00144       if(iq0pin ==4) call rx0( ' OK! qg4gw mode=4 Readin Q0P mode. Set ngp=ngc=0')
00145       end
```

4.39 Wannier/genMLWF File Reference

Variables

- if [$#-ne 3][$2!="-np"]
- then echo An example of usage

4.39.1 Variable Documentation

4.39.1.1 if[$#-ne 3][$2!="-np"]

Definition at line 9 of file genMLWF.

4.39.1.2 then echo An example of usage

Definition at line 26 of file genMLWF.

4.40 genMLWF

```
00001 #!/bin/bash
00002 # ------------------------------------------------------------------------
00003 # generate MLWF.
00004 # NOTE: Wannier is generated before wanplot (wanplot is only to make *.xsf file for plot).
00005 #       After wanplot, we goto calculate <wan wan |W |wan wan>
00006 # For cray, set machine="cray"
00007 #------------------------------------------------------------------------
00008 ### all input arguments are processed ###
00009 if [ $# -ne 3 ] || [ $2 != "-np" ] ; then
00010     echo "An example of usage: genMLWF cu -np 4"
00011     echo "Do job_band_* in advance to genMLWF to get superposition of Wannier band plot!"
00012     exit 101
00013 fi
00014 nfpgw=`dirname $0`
00015 MATERIAL=$1
00016 MPI_SIZE=$3
00017 NO_MPI=0
00018 ### end of processing input arguments ###
00019
00020
00021 ### Read funcitons run_arg and run_arg_tee defined in a file run_arg ###
00022 source $nfpgw/run_arg
00023
00024
00025 ######## start here ##########
00026 $echo_run echo "### START genMLWF: MPI size= " $MPI_SIZE, "MATERIAL= "$MATERIAL
00027 rm -f SYML BNDS
00028 ln -s syml.${MATERIAL} SYML
00029 ln -s bnds.${MATERIAL} BNDS
00030 ## Make softlink from sigm --> simg.$MATERIAL.
00031 ## If sigm and sigm.$MATERIAL coexist, sigm.$MATERIAL is moved to sigm.$MATERIAL.backup in advance.
00032 if [ -e sigm ]; then
00033     if [ -e sigm.$MATERIAL ]; then
00034         mv sigm.$MATERIAL sigm.$MATERIAL.bakup
00035         ln -s -f sigm sigm.$MATERIAL
00036         $echo_run echo '--- sigm is used. sigm.$MATERIAL is softlink to it  ---'
00037     fi
00038 else
00039     $echo_run echo '--- Neither sigm nor sigm.$MATERIAL exists. ==> LDA '
00040 fi
00041
00042 ######## lmf part ########################################
00043 run_arg '---' $NO_MPI   $nfpgw /lmfa  llmfa $MATERIAL # if lmfa is not yet.
00044 run_arg '---' $MPI_SIZE $nfpgw /lmf-MPIK llmf_start $MATERIAL
00045 rm -f ewindow.${MATERIAL}* qbyl.${MATERIAL}* eigze*.${MATERIAL}* # remove temporaly files.
00046
00047 ##### preparation of required inputs for GW (mainly prepare required eigenfuncitons) ######
```

```
00048 argin=0; run_arg $argin $NO_MPI   $nfpgw /lmfgw      llmfgw00 $MATERIAL
00049 argin=1; run_arg $argin $NO_MPI   $nfpgw /qg4gw      lqg4gw   #Generate requied q+G vectors.
00050 argin=1; run_arg $argin $MPI_SIZE $nfpgw /lmfgw-MPIK llmfgw01 $MATERIAL
00051 run_arg  '---' $NO_MPI   $nfpgw /lmf2gw     llmf2gw  #reform data for gw
00052
00053 ##### GW related part (up to preparation of MPB) ###########
00054 argin=0; run_arg $argin $NO_MPI   $nfpgw /rdata4gw_v2 lrdata4gw_v2
00055 if [ -e ANFcond ];then # This is for ANFcond. Unused recently
00056     #  cp EVU EVD
00057     $echo_run echo "Not maintained recently"
00058     exit 10
00059 fi
00060 argin=1; run_arg $argin $NO_MPI $nfpgw /heftet leftet # A file EFERMI for hx0fp0
00061 argin=1; run_arg $argin $NO_MPI $nfpgw /hchknw lchknw # A file NW, containing nw for given QPNT (probably
      only for one-shot GW).
00062 argin=0; run_arg $argin $NO_MPI $nfpgw /hbasfp0 lbas  # Product basis generation
00063
00064 ##### maxloc start here ########################
00065 argin=1 ;run_arg $argin $NO_MPI $nfpgw   /hmaxloc lmaxloc1     # b-vector BBVEC
00066 argin=1 ;run_arg $argin $MPI_SIZE $nfpgw /hpsig_MPI lpsig_MPI # PSIG* =<Psi|Gaussian>.
00067 # Gather all PSIG* into a file. (U meand UP isp=1, D means Down spin isp=2)
00068 cat PSIGU.* >PSIGU
00069 rm -f PSIGU.*
00070 if [ -e PSIGD.0000 ]; then
00071     cat PSIGD.* >PSIGD
00072     rm -f PSIGD.*
00073 fi
00074
00075 argin=2 ;run_arg $argin $MPI_SIZE $nfpgw /huumat_MPI luumat2   # UU (UUmatrix <u_k,i|u_k+b,j>) matrix are
      caltulated.
00076 # Gather all UU*.* into a file UUU/UUD.
00077 cat UUU.* >UUU
00078 rm -f UUU.*
00079 if [ -e UUD.0000 ]; then
00080     cat UUD.* >UUD
00081     rm -f UUD.*
00082 fi
00083 # -- Main part of Wannier (Both of Souza's and Marzari's and procedures sucessively).
00084 argin=2; run_arg $argin $NO_MPI $nfpgw /hmaxloc lmaxloc2  #(band plot data are generated.)
00085
00086
00087 ############## Wannier function plot. *.xsf for Xcrysden. ############
00088 run_arg '---' $NO_MPI $nfpgw  /wanplot lwanplot
00089
00090
00091 ### Here on, we calculate W (v and W-v) for Wannier.###########
00092 # -- UUmatrix for Q0P (offset Gamma point) are required calculation v and W at the limit of q \to 0.
00093 argin=3; run_arg $argin $MPI_SIZE $nfpgw /huumat_MPI luumat3
00094 # Gather all UU*.* into a file UU*, PSIG* as well. (U meand UP isp=1, D means Down spin isp=2)
00095 if [ -e UUq0U.0000 ]; then
00096     cat UUq0U.* > UUq0U
00097     rm -f UUq0U.*
00098 fi
00099 if [ -e UUq0D.0000 ]; then
00100     cat UUq0D.* > UUq0D
00101     rm -f UUq0D.*
00102 fi
00103
00104 ### pkm4crpa file mode for crpa ###
00105 argin=10011; run_arg $argin 1 $nfpgw /hwmatK_MPI lpkm4crpa
00106
00107 ### Main part of v, W-v for Wanniers. ########################
00108 argin=0;  run_arg $argin $MPI_SIZE $nfpgw /hvccfp0 lvcc       # Coulomb matrix v
00109 argin=1;  run_arg $argin $MPI_SIZE $nfpgw /hwmatK_MPI lwmatK1 # Matrix elements of v for Wannier
00110 grep "Wannier" lwmatK1 > Coulomb_v
00111 argin=11; run_arg $argin $MPI_SIZE $nfpgw /hx0fp0 lx011       # Screened Coulomb W minus v, W-v
00112 argin=2;  run_arg $argin $MPI_SIZE $nfpgw /hwmatK_MPI lwmatK2 # Matrix element of W-v
00113 grep "Wannier" lwmatK2 > Screening_W-v
00114 #$nfpgw/Cal_W.py
00115
00116 #### crpa
00117 argin=10011; run_arg $argin $MPI_SIZE $nfpgw /hx0fp0 lx011crpa # cRPA Screened Coulomb W minus v, W-v
00118 argin=2;  run_arg $argin $MPI_SIZE $nfpgw /hwmatK_MPI lwmatK2crpa # Matrix element of W-v
00119 grep "Wannier" lwmatK2crpa > Screening_W-v_crpa
00120 #$nfpgw/Cal_W.py
00121
00122 $echo_run echo "OK! It's finished well."
00123 exit 0
```

4.41 Wannier/hmaxloc.F File Reference

Functions/Subroutines

- program hmaxloc
- subroutine chk_amnkweight (qbz, iko_ix, iko_fx, amnk, nqbz, nwf, nband, nlmto)

  read dimensions of wc,b,hb
- subroutine chk_cnkweight (qbz, iko_ix, iko_fx, cnk, nqbz, nwf, nband, nlmto)
- subroutine chk_umn (cnk, umnk, qbz, iko_ix, iko_fx, iko_i, iko_f, nwf, nqbz, nband, nlmto)

4.41.1 Function/Subroutine Documentation

4.41.1.1 subroutine chk_amnkweight (real(8), dimension(3,nqbz) *qbz,* *iko_ix,* *iko_fx,* complex(8), dimension(iko_ix:iko_fx,nwf,nqbz) *amnk,* *nqbz,* *nwf,* *nband,* *nlmto*)

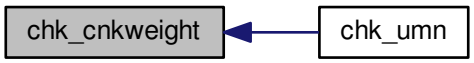
read dimensions of wc,b,hb

Definition at line 1129 of file hmaxloc.F.

4.41.1.2 subroutine chk_cnkweight (real(8), dimension(3,nqbz) *qbz,* *iko_ix,* *iko_fx,* complex(8), dimension(iko_ix:iko_fx,nwf,nqbz) *cnk,* *nqbz,* *nwf,* *nband,* *nlmto*)

Definition at line 1199 of file hmaxloc.F.

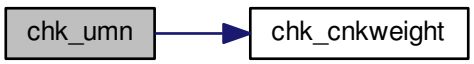
Here is the caller graph for this function:



4.41.1.3 subroutine chk_umn (complex(8), dimension(iko_ix:iko_fx,nwf,nqbz) *cnk,* complex(8), dimension(nwf,nwf,nqbz) *umnk,* real(8), dimension(3,nqbz) *qbz,* *iko_ix,* *iko_fx,* integer(4), dimension(nqbz) *iko_i,* integer(4), dimension(nqbz) *iko_f,* *nwf,* *nqbz,* *nband,* *nlmto*)

Definition at line 1269 of file hmaxloc.F.

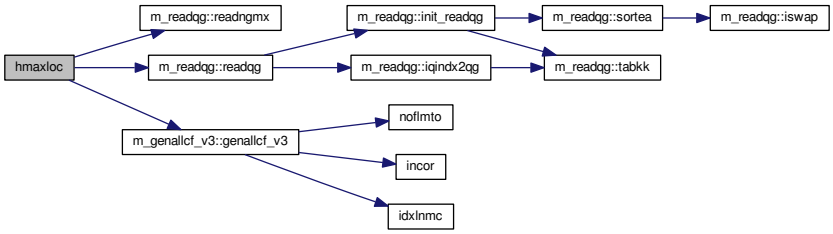
Here is the call graph for this function:



4.41.1.4 program hmaxloc ( )

Definition at line 1 of file hmaxloc.F.

Here is the call graph for this function:



4.42 hmaxloc.F

```
00001       program hmaxloc
00002 c-------------------------------------------------------------
00003 c construct maximally localized Wannier functions
00004 c
00005 c References
00006 c [1] N. Marzari and D.Vanderbilt, PRB56,12847(1997)
00007 c [2] I. Souza, N. Marzari and D.Vanderbilt, PRB65,035109(2002)
00008 c
00009 c mode 1:  determine parameters for <u(m,k)|u(n,k+b> (uu-matrix)
00010 c mode 2:  main part
00011 c    Step 1: choose Hilbert space  (Ref.[2])
00012 c    Step 2: maximally localize Wannier functions (Ref.[1])
00013 c    Step 3: construct effective Hamiltonian and interpolate bands (Ref.[2])
00014 c
00015 cm Oct 2008 Takashi Miyake, updated
00016 cm Aug 2007 Takashi Miyake, berry connection in the Wannier gauge
00017 c  May 2004 Takashi Miyake, from hwmat.f
00018 c-------------------------------------------------------------
00019       use m_readqg,only: readngmx,readqg
00020       use m_readeigen,only: init_readeigen,init_readeigen2,readeval
00021       use m_read_bzdata,only: read_bzdata,
00022      &     ngrp2=>ngrp,nqbz,nqibz,nqbzw,nteti,ntetf,n1,n2,n3,qbas,ginv,qbasmc,
00023      &     dq_,qbz,wbz,qibz,wibz,qbzw,
00024      &     idtetf,ib1bz,idteti,
00025      &     nstar,irk,nstbz
00026       use m_genallcf_v3,only: genallcf_v3,
00027      &     nclass,natom,nspin,nl,nn,ngrp,
00028      &     nlmto,nlnmx, nctot,niw,nw_input=>nw,
00029      &     alat,ef, diw,dw,delta,deltaw,esmr,symgrp,clabl,iclass,
00030      &     invg, il, in, im, nlnm,
00031      &     plat, pos, ecore, symgg , konf,z,
00032      &     spid
00033       use keyvalue,only: getkeyvalue
00034       implicit none
00035 c-------------------------------------
00036       real(8)    :: esmr2,shtw
00037       integer(4)::
00038      &   ixc,iopen,ifhbed, nprecb,mrecb,mrece,nlmtot,nqbzt, nband,
00039      &   ibas,ibasx,ngpmx,nxx,ngcmx,nbloch,ifqpnt,ifwd,ifbb,
00040      &   nprecx,mrecl,nblochpmx2,nwt,niwt, nqnum,mdimx,nblochpmx,
00041      &   ifrcw,ifrcwi,  noccxv,maxocc2,noccx,ifvcfpout,iqall,iaf,ntq,
00042      &   i,j,k,nspinmx, nq,is,ip,iq,idxk,ifoutsex,iclose,nq0i,ig,
00043      &   mxkp,nqibzxx,ntet,nene,iqi, ix,iw,
00044      &   nlnx4,niwx,irot,invr,invrot,ivsum, ifoutsec,ntqx,
00045      &    ifmlw(2),ifmlwe(2) !,ifcphi
00046      &   ,ifxc(2),ifsex(2), ifphiv(2),ifphic(2),ifec,ifexsp(2),
00047      &   ifsecomg(2),ifexx,ifwand,ndble=8
00048       real(8) :: pi,tpia,vol,voltot,rs,alpha,
00049      & qfermi,efx,valn,efnew,edummy,efz,qm,xsex,egex,
00050      & zfac1,zfac2,dscdw1,dscdw2,dscdw,zfac,ef2=1d99,exx,exxq,exxelgas
00051       logical lqall,laf
00052
00053       integer(4),allocatable :: itq(:)
00054       real(8),allocatable    :: q(:,:)
00055
```

```
00056 c takao
00057       integer(4),allocatable :: ngvecpb(:,:,:),!ngveccB(:,:,:),
00058      &  ngvecp(:,:), ngvecc(:,:),iqib(:), !,ngpn(:)ngcni(:)
00059      &  kount(:,:), nx(:,:),nblocha(:),lx(:) !ngveccBr(:,:,:)
00060       real(8),allocatable:: vxcfp(:,:,:),
00061      &  wqt(:), wgt0(:,:),q0i(:,:),
00062      &  ppbrd(:,:,:,:,:,:,:),cgr(:,:,:,:),eqt(:),
00063      &  ppbrdx(:,:,:,:,:,:,:),aaa(:,:), !symope(:,:,:)=symgg, ! qibz(:,:),
00064      &  ppb(:), eq(:), !,pdb(:),dpb(:),ddb(:)
00065      &  eqx(:,:,:),eqx0(:,:,:),ekc(:),coh(:,:)
00066      &         , rw_w(:,:,:,:,:),cw_w(:,:,:,:,:),
00067      &           rw_iw(:,:,:,:,:),cw_iw(:,:,:,:,:)
00068       complex(8),allocatable:: geigb(:,:,:,:)
00069 c
00070       logical :: screen, exchange, cohtest, legas, tote
00071       real(8) ::  rydberg,hartree
00072       real(8):: qreal(3), ntot,nocctotg2,tripl,xxx(3,3)
00073       logical ::nocore
00074
00075 c space group information
00076       integer(4),allocatable :: iclasst(:), invgx(:), miat(:,:)
00077       real(8),allocatable    :: tiat(:,:,:),shtvg(:,:)
00078
00079 c
00080       real(8),allocatable   :: eex1(:,:,:),exsp1(:,:,:),qqex1(:,:,:,:)
00081       integer(4),allocatable:: nspex(:,:),ieord(:),itex1(:,:,:)
00082       real(8)    :: qqex(1:3), eex,exsp,eee, exwgt,deltax0
00083       integer(4) :: itmx,ipex,itpex,itex,nspexmx,nnex,isig,iex,ifexspx
00084      & ,ifexspxx ,ifefsm, nq0ix,ifemesh,nz
00085       character(3)  :: charnum3,sss
00086       character(12) :: filenameex
00087       logical :: exspwrite=.false.
00088       character*8 xt
00089
00090
00091       integer(4)::nqbze,ini,nq0it,idummy
00092       real(8),allocatable:: qbze(:,:)
00093
00094       real(8)   :: ebmx
00095       integer(4):: nbmx
00096
00097       real(8):: volwgt
00098
00099       integer(4)::nwin, incwfin
00100       real(8)::efin,ddw
00101       integer(4),allocatable::imdim(:)
00102       real(8),allocatable::freqx(:),freqw(:),wwx(:),expa(:)
00103
00104       logical:: gausssmear !readgwinput,
00105       integer(4)::ret
00106       character*(150):: ddd
00107
00108
00109       integer(4):: bzcase,  ngpn1,mrecg,verbose,ngcn1,nwxx
00110       real(8)   :: wgtq0p,quu(3)
00111
00112       integer(4):: iii,isx,ivsumxxx
00113
00114 c for maxloc
00115       real(8)   :: wbb(12),wbbsum,bb(3,12),
00116      c             eomin,eomax,eimin,eimax,
00117      c             qwf0(3),dqwf0(3),qks(3),q0(3)
00118       complex(8),allocatable:: uumat(:,:,:,:),evecc(:,:),eveccs(:,:),
00119      c                         amnk(:,:,:),cnk(:,:,:),umnk(:,:,:)
00120       real(8),allocatable:: ku(:,:),kbu(:,:,:),eunk(:,:),eval(:),evals(:),
00121      c                      eks(:),rt(:,:),rt8(:,:,:),qbz0(:,:),r0g(:,:),
00122      c                      wphi(:,:)
00123       integer(4):: nbb,isc,nwf,ifmloc,ifq0p,
00124      c             nox,iko_ix,iko_fx,
00125      c             noxs(2),iko_ixs(2),iko_fxs(2),
00126      c             ieo_swt,iei_swt,itin_i,itin_f,itout_i,itout_f,nphix,
00127      c             nbbelow,nbabove
00128       integer(4),allocatable:: ikbidx(:,:)
00129       integer(4),allocatable:: iki_i(:),iki_f(:),
00130      c                         ikbi_i(:,:),ikbi_f(:,:),
00131      c                         iko_i(:),iko_f(:),
00132      c                         ikbo_i(:,:),ikbo_f(:,:),
00133      c                         iphi(:,:),iphidot(:,:),
00134      c                         nphi(:)
00135       logical :: leout,lein,lbin,lq0p,lsyml,lbnds
00136       logical :: debug=.false.
00137 !
00138       integer(4):: nlinex,ntmp
00139       parameter(nlinex=100)
00140       integer(4)::nline,np(nlinex)
00141       real(8):: qi(3,nlinex),qf(3,nlinex)
00142 c step 1
```

```
00143       complex(8),allocatable:: cnq0(:,:),
00144      c                         upu(:,:,:,:),cnk2(:,:,:),
00145      c                         zmn(:,:)
00146       complex(8):: ctmp
00147       real(8),allocatable:: omgik(:)
00148       real(8)   :: omgi,omgiold,conv1,alpha1,domgi,qtmp(3)
00149       integer(4):: nsc1,ndz,nin,ifhoev,ifuu0,ifpsig
00150 c step 2
00151       complex(8),allocatable:: mmn(:,:,:,:),mmn0(:,:,:,:),
00152      c                         rmn(:,:),smn(:,:),amn(:,:),
00153      c                         tmn(:,:),dwmn(:,:)
00154       real(8),allocatable:: rn(:,:),qn(:)
00155       real(8)   :: omgd,omgod,omgdod,omgidod,omgdododld,domgdod,
00156      c             conv2,alpha2
00157       integer(4):: nsc2,ibb,ii,ij,ik
00158       logical :: lrmn,lmmn
00159 c step 3
00160       complex(8),allocatable:: hrotk(:,:,:),hrotr(:,:,:),hrotkp(:,:)
00161      c                        , hrotkps(:,:)
00162       real(8):: e1,e2,rcut
00163       integer(4):: iband,ifbnd,iftb,ifsh,nsh,nsh1,nsh2
00164       logical :: lsh
00165       real(8),allocatable :: rws(:,:),drws(:)
00166       integer(4),allocatable:: irws(:)
00167       integer(4):: nrws,ifham
00168
00169 c ixc=3
00170       character(20)::filename
00171       complex(8),allocatable::  hrotrcut(:,:,:)
00172       integer:: ifh
00173       real(8):: heps ,r_v
00174
00175       real(8)::qold(3)
00176       real(8),allocatable:: xq(:),eval1(:,:),eval2(:,:),eval3(:,:)
00177
00178       integer::npin
00179       real(8):: qiin(3),qfin(3)
00180
00181       integer(4),allocatable::
00182      &  m_indx(:),n_indx(:),l_indx(:),ibas_indx(:),ibasiwf(:)
00183       integer:: ifoc,iwf,ldim2,ixx,ifile_handle
00184
00185       real(8):: enwfmax,qxx(3),eeee,enwfmaxi
00186       integer:: inii
00187       logical:: leauto,leinauto
00188 ccccccccccccccccccccccccccccccccccccccccccxxxxxxx
00189 c      open(1107,file='xxx1')
00190 c      open(1108,file='xxx2')
00191 ccccccccccccccccccccccccccccccccccccccccccc
00192
00193 c--------------------------------------
00194       hartree=2d0*rydberg()
00195
00196       iii=verbose()
00197       write(6,*)' verbose=',iii
00198
00199 c mode switch. ---------------
00200       write(6,*) ' --- Choose omodes below ----------------'
00201       write(6,*) '  bb vectors (1) or Wannier fn. (2) or TB Hamiltonian (3)'
00202       write(6,*) ' --- Put number above ! ------------------'
00203       call readin5(ixc,nz,idummy)
00204       write(6,*) ' ixc=',ixc
00205       if(ixc<1.or.ixc>3) call rx(' --- ixc=0 --- Choose computational mode!')
00206
00207 c---  readin BZDATA. See gwsrc/rwbzdata.f
00208 c--------readin data set when you call read_BZDATA ---------------
00209 c       integer(4)::ngrp,nqbz,nqibz,nqbzw,nteti,ntetf
00210 ccccc    ! &   ,n_index_qbz
00211 c       integer(4):: n1,n2,n3
00212 c       real(8):: qbas(3,3),ginv(3,3),qbasmc(3,3)
00213 c       real(8),allocatable:: qbz(:,:),wbz(:),qibz(:,:)
00214 c     &    ,wibz(:),qbzw(:,:)
00215 c       integer(4),allocatable:: idtetf(:,:),ib1bz(:),idteti(:,:)
00216 c     &    ,nstar(:),irk(:,:),nstbz(:)          !,index_qbz(:,:,:)
00217 c-----------------------------------------------------------------
00218       call read_bzdata()
00219       write(6,*)' nqibz ngrp=',nqibz,ngrp
00220       write(6,*)' nqbz  =',nqbz
00221 c      write(6,*)  qbz
00222 c      write(6,*)' irk=',irk
00223 c      write(6,*)' #### idtetf: ####'
00224 c      write(6,*) idtetf
00225
00226 c set up work array
00227 c      call wkinit (iwksize)
00228       call pshprt(60)
00229
```

```
00230 C--- readin GWIN and LMTO, then allocate and set datas.
00231       nwin =-999    !not readin NW file
00232       efin =-999d0  !not readin EFERMI
00233 c      efin = 0d0    !readin EFERMI
00234       incwfin= -1  !use 7th colmn for core at the end section of GWIN
00235       call genallcf_v3(nwin,efin,incwfin) !in module m_genallcf_v3
00236       if(ngrp/= ngrp2) stop 'ngrp inconsistent: BZDATA and LMTO GWIN_V2'
00237 c---  These are allocated and setted.
00238 c      integer(4):: nclass,natom,nspin,nl,nn,nnv,nnc, ngrp,
00239 c     o  nlmto,nlnx,nlnxv,nlnxc,nlnmx,nlnmxv,nlnmxc, nctot,niw, !not readin nw
00240 c      real(8) :: alat,ef, diw,dw,delta,deltaw,esmr
00241 c      character(120):: symgrp
00242 c      character(6),allocatable :: clabl(:)
00243 c      integer(4),allocatable:: iclass(:)
00244 c     &  ,nindxv(:,:),nindxc(:,:),ncwf(:,:,:) ,
00245 c     o    invg(:), il(:,:), in(:,:), im(:,:),   ilnm(:),  nlnm(:),
00246 c     o    ilv(:),inv(:),imv(:),  ilnmv(:), nlnmv(:),
00247 c     o    ilc(:),inc(:),imc(:),  ilnmc(:), nlnmc(:),
00248 c     o    nindx(:,:),konf(:,:),icore(:,:),ncore(:),
00249 c     &    occv(:,:,:),unoccv(:,:,:)
00250 c     &   ,occc(:,:,:),unoccc(:,:,:),
00251 c     o    nocc(:,:,:),nunocc(:,:,:)
00252 c      real(8), allocatable::
00253 c     o  plat(:,:),pos(:,:),z(:),  ecore(:,:),  symgg(:,:,:) ! symgg=w(igrp),freq(:)
00254 c-----------------------------------------------------------------------
00255
00256 ccccccccccccccccccccccccccccccccccccccccccccc
00257       do i=1,natom
00258        print *,'  iatom, spid= ',i,spid(i)
00259       enddo
00260 ccccccccccccccccccccccccccccccccccccccccccccc
00261
00262 c--- Get maximums takao 18June03
00263       call getnemx(nbmx,ebmx,8,.true.) !8+1 th line of GWIN0
00264
00265 c-----------------------------------------------------------------------
00266 c      if (nclass > mxclass) stop ' hsfp0: increase mxclass'
00267 c!!!! WE ASSUME iclass(iatom)= iatom !!!!!!!!!!!!!!!!!!!!!!!!
00268       if (nclass /= natom ) stop ' hsfp0: nclass /= natom ' ! We assume nclass = natom.
00269       write(6,*)' hsfp0: end of genallcf2'
00270 c
00271
00272       call pshprt(30)
00273       pi   = 4d0*datan(1d0)
00274       tpia = 2d0*pi/alat
00275
00276       call dinv33(plat,1,xxx,vol)
00277       voltot = dabs(vol)*(alat**3)
00278
00279       ifmlw(1) = iopen('MLWU',0,-1,0)
00280       ifmlwe(1)= iopen('MLWEU',0,-1,0)
00281       if (nspin == 2) then
00282          ifmlw(2) = iopen('MLWD',0,-1,0)
00283          ifmlwe(2)= iopen('MLWED',0,-1,0)
00284       endif
00285
00286 c>> read dimensions of wc,b,hb
00287       ifhbed     = iopen('hbe.d',1,0,0)
00288       read (ifhbed,*) nprecb,mrecb,mrece,nlmtot,nqbzt, nband,mrecg
00289       if (nprecb == 4) stop 'hsfp0: b,hb in single precision'
00290
00291       call init_readeigen(ginv,nspin,nband,mrece) !initialization of readEigen
00292
00293 c --- get space group information ---------------------------------
00294 c true class information in order to determine the space group -----------
00295 c because the class in the generated GW file is dummy.(iclass(ibas)=ibas should be kept).
00296       open (102,file='CLASS')
00297       allocate(iclasst(natom),invgx(ngrp)
00298      &         ,miat(natom,ngrp),tiat(3,natom,ngrp),shtvg(3,ngrp))
00299       write(6,*)'  --- Readingin CLASS info ---'
00300       do ibas = 1,natom
00301         read(102,*) ibasx, iclasst(ibas)
00302         write(6, "(2i10)") ibasx, iclasst(ibas)
00303       enddo
00304
00305 c Get space-group transformation information. See header of mptaouof.
00306       call mptauof(symgg,ngrp,plat,natom,pos,iclasst
00307      o             ,miat,tiat,invgx,shtvg )
00308 c        write (*,*)  'tiat=', tiat(1:3,1:natom,invr),invr
00309
00310 c-----------------------------------------------
00311       call pshprt(60)
00312
00313 c... Readin eigen functions
00314 c      ifev(1)   = iopen('EVU', 0,0,mrece)
00315 c      if (nspin==2) ifev(2) = iopen('EVD', 0,0,mrece)
00316
```

```
00317 ! read EF from 'BNDS' if exists
00318       lbnds=.false.
00319       inquire(file='BNDS',exist=lbnds)
00320       if (lbnds) then
00321         write(*,*)'Read EF from BNDS'
00322         open(99,file='BNDS',status='old')
00323         read(99,*)ntmp,ef
00324         close(99)
00325       else ! lbnds
00326 c --- determine Fermi energy ef for given valn (legas case) or corresponding charge given by z and konf.
00327 ! When esmr is negative, esmr is geven automatically by efsimplef.
00328         write(*,*)'Calculate EF in efsimplef2a'
00329         legas = .false.
00330         call efsimplef2a(nspin,wibz,qibz,ginv,
00331      i        nband,nqibz
00332      i       ,konf,z,nl,natom,iclass,nclass
00333      i       ,valn, legas, esmr,  !!! valn is input for legas=T, output otherwise.
00334 c
00335      i        qbz,nqbz !index_qbz, n_index_qbz,
00336      o       ,efnew)
00337 c
00338 c        write(6,*)' end of efsimple'
00339         ef = efnew
00340       endif ! lbnds
00341 c- check total ele number -------
00342       ntot  = nocctotg2(nspin, ef,esmr, qbz,wbz, nband,nqbz) !wbz
00343       write(6,*)' ef    =',ef
00344       write(6,*)' esmr  =',esmr
00345       write(6,*)' valn  =',valn
00346       write(6,*)' ntot  =',ntot
00347
00348 c      ifcphi  = iopen('CPHI',0,0,mrecb)
00349
00350       call init_readeigen2(mrecb,nlmto,mrecg) !initialize m_readeigen
00351
00352 !c QPNT data
00353 ctm, 080222
00354 ! read QPNT from 'SYML' if exists
00355       lsyml=.false.
00356       inquire(file='SYML',exist=lsyml)
00357       if (lsyml) then
00358         write(*,*)'Read k points for bands from SYML'
00359         lqall      = .false.
00360         laf        = .false.
00361         open(99,file='SYML',status='old')
00362         nline=0
00363         do i = 1,nlinex
00364           read(99,*,err=551,end=552)npin,qiin,qfin
00365           if (npin==0) exit
00366           nline = nline+1
00367           np(nline)=npin
00368           qi(1:3,nline)=qiin
00369           qf(1:3,nline)=qfin
00370  551      continue
00371         enddo
00372  552    continue
00373         if (nline.eq.nlinex) call rx('hmaxloc: too many lines in SYML')
00374         close(99)
00375         nq = 0
00376         do i = 1,nline
00377            nq = nq + np(i)
00378         enddo ! i
00379         allocate(q(3,nq),xq(nq))
00380         iq = 0
00381         xq=0d0
00382         qold=q(:,1)
00383         do i = 1,nline
00384         do j = 0,np(i)-1
00385           iq = iq + 1
00386           q(:,iq) = qi(:,i) + (qf(:,i)-qi(:,i))*dble(j)/dble(np(i)-1)
00387           if(iq>1) then
00388             xq(iq)= xq(iq-1) + dsqrt( sum((q(:,iq)-qold)**2) )
00389           endif
00390           qold=q(:,iq)
00391         enddo ! j
00392         enddo ! i
00393       else ! lsyml
00394         write(*,*)'Read k points for bands from GWinput'
00395         call getkeyvalue("GWinput","<QPNT>",unit=ifqpnt,status=ret)
00396         write(6,*)' ifqpnt ret=',ifqpnt,ret
00397 c
00398         lqall      = .false.
00399         laf        = .false.
00400         call readx(ifqpnt,10)
00401         read (ifqpnt,*) iqall,iaf
00402         if (iqall == 1) lqall = .true.
00403         if (iaf   == 1)   laf = .true.
```

```
00404         call readx(ifqpnt,100)
00405 ctm 040622
00406         read (ifqpnt,*)
00407         read (ifqpnt,*)
00408
00409         if (lqall) then !all q-points case
00410           nq         = nqibz
00411           allocate(q(3,nq))
00412           call dcopy(3*nqibz,qibz,1,q,1)
00413         else
00414           call readx(ifqpnt,100)
00415           read (ifqpnt,*) nq
00416           allocate(q(3,nq))
00417           do       k = 1,nq
00418           read (ifqpnt,*) i,q(1,k),q(2,k),q(3,k)
00419           write(6,'(i3,3f13.6)') i,q(1,k),q(2,k),q(3,k)
00420           enddo
00421         endif ! lqall
00422         close(ifqpnt)
00423         allocate(xq(nq))
00424         xq=0d0
00425       endif ! syml
00426 c
00427       nspinmx = nspin
00428       if (laf) nspinmx =1
00429 c------------
00430 c input parameters specific to MAXLOC
00431       call getkeyvalue("GWinput","<MLWF>",unit=ifmloc,status=ret)
00432       write(6,*)' ifmloc ret=',ifmloc,ret
00433       read (ifmloc,*) nwf
00434       allocate (nphi(nwf))
00435       read (ifmloc,*) (nphi(i),i=1,nwf)
00436       nphix = 0
00437       do i = 1,nwf
00438          if(nphi(i).gt.nphix)nphix = nphi(i)
00439       enddo
00440       allocate (r0g(nphix,nwf),iphi(nphix,nwf),iphidot(nphix,nwf),
00441      &          wphi(nphix,nwf))
00442       do i=1,nwf
00443          do j=1,nphi(i)
00444             read(ifmloc,*)iphi(j,i),iphidot(j,i),r0g(j,i),wphi(j,i)
00445          enddo
00446       enddo
00447       close(ifmloc)
00448
00449       call wan_input(leout,lein,lbin,ieo_swt,iei_swt,
00450      &    eomin,eomax,itout_i,itout_f,nbbelow,nbabove,
00451      &    eimin,eimax,itin_i,itin_f,
00452      &    nsc1,nsc2,conv1,conv2,alpha1,alpha2,rcut)
00453 c
00454
00455 cskino
00456       r_v=rcut
00457       call getkeyvalue("GWinput",'wan_tbcut_rcut',heps,default=r_v)
00458       call getkeyvalue("GWinput",'wan_tbcut_heps',heps,default=0.0d0)
00459       write(*,*) 'mloc.heps ', heps
00460 cekino
00461
00462
00463 cc --- read LDA eigenvalues
00464       ntq = nwf
00465 cc     ntp0=ntq
00466 c      allocate(eqx(ntq,nq,nspin),eqx0(ntq,nq,nspin),eqt(nband))
00467 c      do      is = 1,nspin
00468 c      do      ip = 1,nq
00469 cc        iq       = idxk (q(1,ip),qbze,nqbze)
00470 cc        call rwdd1   (ifev(is), iq, nband, eqt) !direct access read b,hb and e(q,t)
00471 c        call readeval(q(1,ip),is,eqt)
00472 cc        write(6,*)' eqt=',eqt
00473 c        eqx0(1:ntq,ip,is) = eqt(itq(1:ntq))
00474 c        eqx (1:ntq,ip,is) = rydberg()*(eqt(itq(1:ntq))- ef)
00475 c      enddo
00476 c      enddo
00477 c      deallocate(eqt)
00478
00479 c --- info
00480       call winfo(6,nspin,nq,ntq,is,nbloch
00481      &    ,0,0,nqbz,nqibz,ef,deltaw,alat,esmr)
00482
00483 c
00484       iii=ivsumxxx(irk,nqibz*ngrp)
00485       write(6,*) " sum of nonzero iirk=",iii, nqbz
00486
00487 c-------------------------------------------------------------
00488 c debug:
00489 c      allocate(eqt(nband))
00490 c      do ip = 1,nqbz
```

```
00491 c        call readeval(qbz(1,ip),1,eqt)
00492 c        write(80,"('***',3f10.5)")qbz(:,ip)
00493 c        do is=1,nband
00494 c           write(80,"(i5,f12.6)")is,eqt(is)
00495 c        enddo
00496 c      enddo
00497
00498 c Rt vectors
00499       allocate (rt(3,nqbz),rt8(3,8,nqbz),qbz0(3,nqbz))
00500 c      write(6,"(a,9f9.4)")'qbas=',qbas
00501 c      write(6,"(a,9f9.4)")'plat=',plat
00502       call getrt(qbz,qbas,plat,n1,n2,n3,nqbz,
00503      o           rt,rt8,qbz0)
00504
00505 c b vectors
00506       call getbb(plat,alat,n1,n2,n3,
00507      o           nbb,wbb,wbbsum,bb)
00508
00509 c index for k and k+bb
00510       allocate (ku(3,nqbz),kbu(3,nbb,nqbz),ikbidx(nbb,nqbz))
00511
00512       call kbbindx(qbz,ginv,bb,
00513      d             nqbz,nbb,
00514      o             ikbidx,ku,kbu)
00515
00516
00517       allocate (iko_i(nqbz),iko_f(nqbz),
00518      &          iki_i(nqbz),iki_f(nqbz),
00519      &          ikbo_i(nbb,nqbz),ikbo_f(nbb,nqbz),
00520      &          ikbi_i(nbb,nqbz),ikbi_f(nbb,nqbz))
00521
00522 !! takao list eigen -----
00523       enwfmax =-1d9
00524       enwfmaxi=1d9
00525       allocate(eqt(1:nband))
00526       do is = 1,nspin
00527       do iq = 1,nqbz
00528          qxx = qbz(:,iq)
00529          call readeval(qxx,is,eqt)
00530          ini=1
00531          do i=1,nband
00532 c           write(6,*)'eqeq',eqt(i),eomin,eqt(nwf)
00533            if (eqt(i)>eomin) then
00534              inii=i
00535              exit
00536            endif
00537          enddo
00538          eeee= (eqt(nwf+inii-1)-ef)*rydberg()
00539          write(6,"('elist: q iq is nwfi nwfe e(nwf)= ',3f9.4,i5,i2,2i5,f10.3)") qxx,iq,is,inii,nwf+inii-1,
     eeee
00540          if (enwfmax < eeee) enwfmax  = eeee
00541          if (enwfmaxi >eeee) enwfmaxi = eeee
00542       enddo
00543       enddo
00544       deallocate(eqt)
00545       write(6,"('elist max enwf enwfmaxi=',2f13.5)") enwfmax,enwfmaxi
00546       call getkeyvalue("GWinput","wan_out_emax_auto",leauto,default=.false.)
00547       if(leauto) then
00548          eomax= enwfmax + 1d-4
00549          write(6,*)
00550          write(6,"(' WE USE wan_out_emax_auto on ==> +1d-3 ==> eomax=',3f13.5)") eomax
00551       endif
00552       call getkeyvalue("GWinput","wan_in_emax_auto",leinauto,default=.false.)
00553       if(leinauto) then
00554          eimax= enwfmaxi + 1d-4
00555          write(6,*)
00556          write(6,"(' WE USE  wan_in_emax_auto on ==> +1d-3 ==> eimax=',3f13.5)") eimax
00557       endif
00558
00559 c      stop 'qqqqqqqqqqqqqqqqqq'
00560
00561 !! ixc = 1 ----------------
00562       if (ixc.eq.1) then
00563          do is = 1,nspin
00564          call ewindow(is,ieo_swt,iei_swt,itout_i,itout_f,itin_i,itin_f,
00565      i                eomin,eomax,eimin,eimax,ef,qbz,ikbidx,
00566      i                nbbelow,nbabove,
00567      d                nqbz,nbb,nband,nwf,nspin,
00568      o                iko_i,iko_f,iki_i,iki_f,
00569      o                ikbo_i,ikbo_f,ikbi_i,ikbi_f,
00570      o                iko_ixs(is),iko_fxs(is),noxs(is),
00571      o                leout,lein)
00572          enddo
00573
00574 c write bb vectors to 'BBVEC'
00575          call writebb(ifbb,wbb(1:nbb),bb(1:3,1:nbb),
00576      i                ikbidx,ku,kbu,
```

```
00577      i                iko_ixs,iko_fxs,noxs,
00578      d                nspin,nqbz,nbb)
00579
00580 ctm, 060923 !!!
00581          ifwand = iopen('wan.d',1,-1,0)
00582          iko_ix = iko_ixs(1)
00583          iko_fx = iko_fxs(1)
00584          if (nspin.eq.2) then
00585             if (iko_ixs(2).lt.iko_ix) iko_ix = iko_ixs(2)
00586             if (iko_fxs(2).gt.iko_fx) iko_fx = iko_fxs(2)
00587          endif
00588          write(ifwand,*)nqbz,nwf,iko_ix,iko_fx
00589          write(ifwand,*)nspin
00590          do is = 1,nspin
00591            write(ifwand,*)nqbz,nwf,iko_ixs(is),iko_fxs(is)
00592          enddo
00593          isx = iclose('wan.d')
00594          call rx0('hmaxloc: ixc=1 ok')
00595       endif
00596
00597 !! loop over spin ------------------------
00598       do 1000 is = 1,nspin
00599       write(*,*)'is =',is,'  out of',nspin
00600 c energy window
00601       call ewindow(is,ieo_swt,iei_swt,itout_i,itout_f,itin_i,itin_f,
00602      i             eomin,eomax,eimin,eimax,ef,qbz,ikbidx,
00603      i             nbbelow,nbabove,
00604      d             nqbz,nbb,nband,nwf,nspin,
00605      o             iko_i,iko_f,iki_i,iki_f,
00606      o             ikbo_i,ikbo_f,ikbi_i,ikbi_f,
00607      o             iko_ix,iko_fx,nox,
00608      o             leout,lein)
00609 !      call chk_ewindow(ifbb,is,nspin,nqbz,nbb,iko_ix,iko_fx)
00610
00611 c read uu-matrix
00612       allocate (uumat(iko_ix:iko_fx,iko_ix:iko_fx,nbb,nqbz))
00613       call readuu(is,iko_ix,iko_fx,ikbidx,
00614      d            nqbz,nbb,
00615      o            uumat)
00616       call chkuu(is,iko_ix,iko_fx,ikbidx,uumat,
00617      d           nqbz,nbb)
00618
00619 !! step 1  -- choose Hilbert space -- determine cnk
00620       write(*,*)'Step 1: Hilbert space branch'
00621       write(6,*)' iko_ix iko_fx=',iko_ix,iko_fx
00622       allocate (amnk(iko_ix:iko_fx,nwf,nqbz),
00623      &          upu(iko_ix:iko_fx,iko_ix:iko_fx,nbb,nqbz),
00624      &          cnk(iko_ix:iko_fx,nwf,nqbz),
00625      &          cnk2(iko_ix:iko_fx,nwf,nqbz),
00626      &          omgik(nqbz))
00627 ! amnk appered in Eq.22 in Ref.II. <psi|Gaussian>
00628       call init_unkg(is,qbz,ginv,ef,lein,
00629      i              iko_ix,iko_fx,iko_i,iko_f,
00630      i              iki_i,iki_f,
00631      d              nwf,nband,nqbz,
00632      o              amnk,cnk)
00633 !      call chk_amnkweight(qbz,iko_ix,iko_fx,amnk,
00634 !     &     nqbz,nwf,nband,nlmto)
00635 !      call chk_cnkweight(qbz,iko_ix,iko_fx,cnk,
00636 !     &     nqbz,nwf,nband,nlmto)
00637       do isc = 1,nsc1
00638          do iq = 1,nqbz
00639             call dimz(lein,iko_i(iq),iko_f(iq),iki_i(iq),iki_f(iq),
00640      o                ndz,nin)
00641             if (nwf.gt.nin) then
00642               if (ndz.lt.1) call rx('ndz < 1')
00643 c (1-2) <u_mk | P_k+b | u_nk>
00644                call getupu(isc,
00645      i                     uumat(:,:,:,iq),cnk,
00646      i                     lein,alpha1,iq,ikbidx(:,iq),
00647      i                     iko_ix,iko_fx,
00648      i                     iko_i(iq),iko_f(iq),
00649      i                     iki_i(iq),iki_f(iq),
00650      i                     ikbo_i(:,iq),ikbo_f(:,iq),
00651      i                     ikbi_i(:,iq),ikbi_f(:,iq),
00652      d                     nwf,nbb,nqbz,
00653      u                     upu(:,:,:,iq))
00654 c (1-3) Zmn(k) > phi,eval
00655                allocate (zmn(ndz,ndz),evecc(ndz,ndz),eval(ndz))
00656                call getzmn(upu(:,:,:,iq),wbb,lein,
00657      i                     iko_ix,iko_fx,
00658      i                     iko_i(iq),iko_f(iq),
00659      i                     iki_i(iq),iki_f(iq),
00660      d                     nwf,nbb,nqbz,ndz,
00661      o                     zmn)
00662
00663                call chk_hm(zmn,ndz)
```

```
00664                call diag_hm(zmn,ndz,eval,evecc)
00665                call new_cnk(cnk(:,:,iq),evecc,iq,
00666      i                      iko_ix,iko_fx,
00667      i                      iko_i(iq),iko_f(iq),
00668      i                      iki_i(iq),iki_f(iq),
00669      d                      nwf,ndz,
00670      o                      cnk2(:,:,iq))
00671 c (1-3) w_I(k)  eq.(18)
00672                call chk_eval(wbb,eval,nbb,ndz)
00673                call get_omgik(wbb,eval,
00674      i                        iko_i(iq),iko_f(iq),
00675      i                        iki_i(iq),iki_f(iq),
00676      d                        nbb,nwf,ndz,
00677      o                        omgik(iq))
00678                deallocate (zmn,evecc,eval)
00679             else
00680                omgik(iq) = 0d0
00681                cnk2(:,:,iq) = cnk(:,:,iq)
00682 c end if (ndz>1)
00683             endif
00684 c end of iq-loop
00685          enddo
00686 c (1-5) w_I(k) > Omaga_I  eq.(11)
00687          omgi = sum(omgik(:)*wbz(:))
00688 c (1-6) check self-consistency
00689          write(*,"('#SC-loop, conv.',i5,d13.5)")isc,omgi
00690          if (isc.ge.2) then
00691             domgi = dabs((omgiold - omgi) / omgiold)
00692             if (domgi .lt. conv1) then
00693                write(*,*) 'step1: converged!'
00694                goto 810
00695             endif
00696          endif
00697 c update
00698          omgiold = omgi
00699          cnk     = cnk2
00700 c end of self-consistent loop
00701       enddo
00702       write(*,*)'step1: not converged'
00703  810  continue
00704       deallocate(upu,cnk2)
00705
00706 c      call chk_cnkweight(qbz,iko_ix,iko_fx,cnk,
00707 c     &     nqbz,nwf,nband,nlmto)
00708
00709 !! NOTE: cnk is the final results of step 1
00710 !!   cnk(iko_ix:iko_fx,nwf,nqbz)
00711 !!   cnk(iko_i(iq):iko_f(iq),nwf,iq) gives nwf-dimentional space.
00712 !!   step 1 (minimization of Omega_I)
00713
00714
00715 !! === step 2 -- localize Wannier fn. ================
00716       write(*,*)'Step 2: Wannier fn. branch'
00717
00718       allocate (mmn(nwf,nwf,nbb,nqbz),mmn0(nwf,nwf,nbb,nqbz),
00719      &          umnk(nwf,nwf,nqbz),
00720      &          rmn(nwf,nwf),amn(nwf,nwf),smn(nwf,nwf),
00721      &          rn(3,nwf),qn(nwf),tmn(nwf,nwf),dwmn(nwf,nwf),
00722      &          eunk(nwf,nqbz))
00723
00724 !! (2-0) construct initlal u~ from u
00725 !! eunk(= e~) of {H~}_mn): eigenvalue within the nwf-dimentional Hilbert space
00726       call diag_unk(is,qbz,
00727      i              iko_ix,iko_fx,iko_i,iko_f,
00728      d              nband,nwf,nqbz,
00729      u              cnk,
00730      o              eunk)
00731
00732 !      call chk_cnkweight(qbz,iko_ix,iko_fx,cnk,
00733 !     &     nqbz,nwf,nband,nlmto)
00734 !
00735 ! check ortho-normality of u~'s
00736 !      call chk_cnk(cnk,
00737 !     i             iko_ix,iko_fx,iko_i,iko_f,
00738 !     d             nband,nwf,nqbz)
00739 !
00740 ! check: eunk vs. KS energy
00741 !      call chk_eunk(is,qbz,eunk,ef,
00742 !     d              nqbz,nband,nwf)
00743
00744 !! (2-1) initial: uumat -> M_mn(0) Eq.58 in Ref.[1]
00745       call init_mmn(cnk,uumat,ikbidx,
00746      i              iko_ix,iko_fx,iko_i,iko_f,ikbo_i,ikbo_f,
00747      d              nwf,nqbz,nbb,
00748      o              mmn0)
00749
00750 !! (2-2) initial U
```

```
00751 !!    umnk= U(m,n) = ( A S^{-1/2} )_mn. See Eq.23 in Ref.II.
00752       call init_umnk(amnk,cnk,
00753      i               iko_ix,iko_fx,iko_i,iko_f,
00754      d               nwf,nqbz,
00755      o               umnk)
00756
00757 !      call chk_umn(cnk,umnk,qbz,
00758 !     i             iko_ix,iko_fx,iko_i,iko_f,
00759 !     d             nwf,nqbz,nband,nlmto)
00760
00761       call updt_mmn(umnk,mmn0,ikbidx,
00762      d              nwf,nqbz,nbb,
00763      u              mmn)
00764
00765 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00766 c         do i=1,nbb
00767 c            write(1106+is,"(a,i4,13f13.5)")'bbbb',i,bb(1:3,i),wbb(i)
00768 c         enddo
00769 c         do i=1,nqbz
00770 c            write(1106+is,"(a,i4,13f13.5)")'www',i,wbz(i)
00771 c         enddo
00772 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00773
00774
00775
00776 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00777       do isc = 1, nsc2
00778 cccccccccccccccccccccc
00779 c      mmn=mmn+(0d0,1d-8)
00780 cccccccccccccccccccccc
00781
00782 c <r_n> ([1] eq.31)
00783          call get_rn(mmn,bb,wbb,wbz,
00784      d               nwf,nqbz,nbb,
00785      o               rn)
00786 ccccccccccccccccccccccccccccccccccccccccc
00787 c         do i=1,nwf
00788 c            write(1106+is,"(a,3f13.5)")'rrrrrn',rn(1:3,i)
00789 c         enddo
00790 ccccccccccccccccccccccccccccccccccccccccccccc
00791
00792          do iq = 1,nqbz
00793             dwmn = (0d0,0d0)
00794             do ibb = 1,nbb
00795 ccccccccccccccccccccccccccccccccccccccccccccc
00796 c      do i = 1,nwf
00797 c      do j = 1,nwf
00798 c         write(1106+is,"(a,4i5,2f13.3)")' mmmmm ',i,j,ibb,iq,mmn(i,j,ibb,iq)+(0d0,0.0001)
00799 c      enddo
00800 c      enddo
00801 c      do i = 1,nwf
00802 c      do j = 1,nwf
00803 c         write(1106+is,"(a,4i5,2f13.3)")' nnnnnn ',i,j,ibb,iq,mmn0(i,j,ibb,iq)+(0d0,0.0001)
00804 c      enddo
00805 c      enddo
00806 ccccccccccccccccccccccccccccccccccccccccccccc
00807
00808 c (2-3) A[R] matrix
00809                call getrmn(mmn(:,:,ibb,iq),
00810      d                     nwf,
00811      o                     rmn)
00812                call getamn(rmn,
00813      d                     nwf,
00814      o                     amn)
00815
00816 c (2-4) S[T] matrix
00817                call gettmn(rn,mmn(:,:,ibb,iq),bb(:,ibb),
00818      d                     nwf,
00819      o                     qn,tmn)
00820                call getsmn(tmn,
00821      d                     nwf,
00822      o                     smn)
00823
00824 ccccccccccccccccccccccccccccc
00825 c               smn=0d0
00826 c               amn=0d0
00827 ccccccccccccccccccccccccccccc
00828
00829 c DW(k) ([1] eq.57)
00830                dwmn(:,:) =  dwmn(:,:)
00831      &       + wbb(ibb) * (amn(:,:) - smn(:,:)) * alpha2 / wbbsum
00832
00833 c end of ibb-loop
00834             enddo
00835
00836 ccccccccccccccccccccc
00837 c            dwmn=0d0
```

```
00838 ccccccccccccccccccccccc
00839 c (2-5) DW(k) -> U(k) ([1] eq.60)
00840             call updt_uk(dwmn,
00841      d                   nwf,
00842      u                   umnk(:,:,iq))
00843 c            call chk_um(umnk(:,:,iq),nwf)
00844
00845 ccccccccccccccccccccccccccccccccccccccccccccccccccc
00846 c      do i = 1,nwf
00847 c      do j = 1,nwf
00848 c         write(1106+is,"(a,3i5,2f13.3)")' zzzzz',iq,i,j,umnk(i,j,iq)
00849 c      enddo
00850 c      enddo
00851 ccccccccccccccccccccccccccccccccccccccccccccccccccc
00852
00853 c end of iq-loop
00854          enddo
00855
00856
00857
00858 c update Mmn ([1] eq.61)
00859          call updt_mmn(umnk,mmn0,ikbidx,
00860      d                 nwf,nqbz,nbb,
00861      u                 mmn)
00862
00863 c (2-6) Omeg_I, Omega_D and Omega_OD ([1] eq.34,35,36)
00864          call getomg(mmn,rn,bb,wbb,wbz,
00865      d               nwf,nqbz,nbb,
00866      o               omgi,omgd,omgod,omgdod,omgidod)
00867
00868 c check self-consistency
00869 c         write(*,*)'#SC-loop, conv.',isc,omgdod
00870 c         write(*,950)'Omg: I, OD, D ',omgi,omgod,omgd
00871          write(*,"('#SC-loop, conv.',i6,e13.5,' Omg:_I,_OD,_D= ',3f17.10)")
00872      &   isc,omgdod,omgi,omgod,omgd
00873          if (isc.ge.2) then
00874             domgdod = dabs((omgdodold - omgdod) / omgdodold)
00875             if (domgdod .lt. conv2) then
00876                write(*,*) 'step2: converged!'
00877                goto 820
00878             endif
00879          endif
00880          omgdodold = omgdod
00881
00882 c end of self-consistent loop
00883       enddo
00884       write(*,*)'step2: not converged'
00885  820  continue
00886
00887 !      call chk_dnk(is,eunk,qbz,
00888 !     i             umnk,cnk,
00889 !     i             iko_ix,iko_fx,iko_i,iko_f,
00890 !     d             nband,nwf,nqbz)
00891 !
00892 !      call chk_umn(cnk,umnk,qbz,
00893 !     i             iko_ix,iko_fx,iko_i,iko_f,
00894 !     d             nwf,nqbz,nband,nlmto)
00895
00896 c output
00897       write(*,*)"---------- wlaxloc isp =",is
00898       call wmaxloc(ifmlw(is),ifmlwe(is),
00899      i             qbz,umnk,cnk,eunk,
00900      i             iko_ix,iko_fx,iko_i,iko_f,
00901      d             nwf,nqbz,nband,nlmto, is)
00902       call writeomg(is,mmn,rn,bb,wbb,wbz,tpia,
00903      d             nwf,nqbz,nbb)
00904 c 070824
00905       call getkeyvalue("GWinput","wan_write_rmn",lrmn,default=.false.)
00906       if (lrmn)
00907      & call writermn(is,mmn,bb,wbb,qbz,qbz0,wbz,rt,
00908      d              nwf,nqbz,nbb,n1,n2,n3)
00909 c 070830
00910       call getkeyvalue("GWinput","wan_write_mmn",lmmn,default=.false.)
00911       if (lmmn)
00912      & call writemmn(is,mmn,bb,wbb,qbz,wbz,rt,
00913      d              nwf,nqbz,nbb,n1,n2,n3)
00914
00915       deallocate(uumat,amnk,omgik,mmn,mmn0,
00916      &           rmn,amn,smn,rn,qn,tmn,dwmn)
00917
00918
00919 !! step 3 -- reduced Hamiltonian ------------------------------
00920       write(*,*)'Step 3: reduced Hamiltonian branch'
00921 c open file
00922       if (is .eq. 1) then
00923          ifbnd = iopen('bnds.maxloc.up',1,-1,0)
00924          iftb  = iopen('bnds.tb.up',1,-1,0)
```

```
00925       else
00926          ifbnd = iopen('bnds.maxloc.dn',1,-1,0)
00927          iftb  = iopen('bnds.tb.dn',1,-1,0)
00928       endif
00929       write(ifbnd,*)nq
00930       write(ifbnd,*)nwf
00931       write(iftb,*)nq
00932       write(iftb,*)nwf
00933 c allocate
00934       allocate (hrotk(nwf,nwf,nqbz), ! hrotr(nwf,nwf,nqbz),
00935      o          hrotkp(nwf,nwf),evecc(nwf,nwf),eval(nwf))
00936 c for small Hamiltonian
00937       call getkeyvalue("GWinput","wan_small_ham",lsh,default=.false.)
00938       if (lsh) then
00939          call getkeyvalue("GWinput","wan_nsh1",nsh1, default=1 )
00940          call getkeyvalue("GWinput","wan_nsh2",nsh2, default=2 )
00941          write(*,*)'SmallHam on',nsh1,nsh2
00942          nsh = nsh2 - nsh1 + 1
00943          if (is .eq. 1) then
00944             ifsh = iopen('bnds.sh.up',1,-1,0)
00945          else
00946             ifsh = iopen('bnds.sh.dn',1,-1,0)
00947          endif
00948          write(ifsh,*)nq
00949          write(ifsh,*)nsh
00950          allocate (hrotkps(nsh,nsh),eveccs(nsh,nsh),evals(nsh))
00951       endif
00952 c (3-1) ~H(k) -> Hrot(k): note eunk is eigenvalues in the basis of cnk
00953       call rot_hmnk(umnk,eunk,
00954      d              nwf,nqbz,
00955      o              hrotk) !rotated Hamiltonian in MLW basis.
00956 c (3-2) Hrot_mn(R)
00957       allocate(irws(n1*n2*n3*8),rws(3,n1*n2*n3*8),drws(n1*n2*n3*8))
00958       call wigner_seitz(alat,plat,n1,n2,n3,nrws,rws,irws,drws)
00959       allocate(hrotr(nwf,nwf,nrws)) !real space Hamiltonian in Wannier funciton basis
00960       if (ixc.eq.2) then
00961          call get_hrotr_ws(hrotk,qbz,wbz,
00962      i        rws,irws,drws,
00963      d        nwf,nqbz,nrws,
00964      o        hrotr)
00965 c     skino
00966 c     write hrotr and *rws
00967          if (is .eq. 1) then
00968             ifh = iopen('hrotr.up',1,-1,0)
00969          else
00970             ifh = iopen('hrotr.dn',1,-1,0)
00971          endif
00972
00973          call write_hrotr(ifh, hrotr,
00974      i        rws,irws,drws,
00975      d        nwf,nrws )
00976
00977          close (ifh)
00978 c     ekino
00979 c     skino
00980       else if (ixc.eq.3) then
00981          if (is .eq. 1) then
00982             filename='hrotr.up'
00983          else
00984             filename = 'hrotr.dn'
00985          endif
00986          call read_hrotr(filename,nwf,nrws,
00987      o        hrotr)
00988          if (is .eq. 1) then
00989             ifh = iopen('hrotr.cut.up',1,-1,0)
00990          else
00991             ifh = iopen('hrotr.cut.dn',1,-1,0)
00992          endif
00993          allocate(hrotrcut(nwf,nwf,nrws))
00994          call make_hrotrcut( hrotr,
00995      i        rws,irws,drws,
00996      i        rcut,heps,
00997      d        nwf,nrws,
00998      o        hrotrcut )
00999          call write_hrotr(ifh, hrotrcut,
01000      i        rws,irws,drws,
01001      d        nwf,nrws )
01002          close (ifh)
01003          deallocate(hrotrcut)
01004 c     ekino
01005       endif
01006
01007 !! ------------------------------------------------------------
01008 !! k-point mesh
01009       call get_nqbze(nqbz,nqbze)
01010       allocate(qbze(3,nqbze))
01011       call get_qbze(qbz,nqbz,
```

```
01012      o             qbze,nqbze)
01013       write(ifmlw(is))nqbze,nwf
01014       write(ifmlwe(is))nqbze,nwf
01015       do iq = 1,nqbze
01016 c          write(*,*)'goto get_hrotkp_ws iq=',iq,nqbze
01017           call get_hrotkp_ws(hrotr,rws,drws,irws,qbze(:,iq), !july2014 qbz->qbze
01018      d                     nwf,nqbz,nrws,
01019      o                     hrotkp)
01020           call diag_hm(hrotkp,nwf,eval,evecc)
01021           call wmaxloc_diag(ifmlw(is),ifmlwe(is),
01022      i                 iq,qbze(1:3,iq),umnk,cnk,eunk,evecc,eval,
01023      i                 iko_ix,iko_fx,iko_i,iko_f,
01024      d                 nwf,nqbz)
01025       enddo
01026 c      write(6,*)'eeeeeeeee'
01027       deallocate(qbze)
01028 ccc          write(*,990)'iq =',iq,qbz(1:3,iq)
01029 cc          if (iq.le.nqbz) then
01030 cc          do iband = 1,nwf
01031 cc             e1 = (eval(iband)   -ef)*rydberg()
01032 cc             e2 = (eunk(iband,iq)-ef)*rydberg()
01033
01034
01035 !! ------------------------------------------------------------------
01036 c --- Readin nlam index
01037       ifoc = iopen('@MNLA_CPHI',1,0,0)
01038       ldim2 = nlmto
01039       read(ifoc,*)
01040       allocate(m_indx(ldim2),n_indx(ldim2),l_indx(ldim2),ibas_indx(ldim2))
01041       do ix =1,ldim2
01042         read(ifoc,*)m_indx(ix),n_indx(ix),l_indx(ix),ibas_indx(ix),ixx
01043         if(ixx/=ix) call rx('failed to readin @MNLA_CPHI')
01044       enddo
01045       ix = iclose('@MNLA_CPHI')
01046       allocate(ibasiwf(nwf))
01047       do iwf=1,nwf
01048         ibasiwf(iwf) = ibas_indx(iphi(1,iwf))
01049       enddo
01050
01051 !! write HrotRS
01052       ifh=ifile_handle()
01053       if(is==1) open(ifh,file='HrotRS.up',form='unformatted')
01054       if(is==2) open(ifh,file='HrotRS.dn',form='unformatted')
01055       write(ifh)alat,plat,natom
01056       write(ifh)pos
01057       write(ifh)ef
01058       write(ifh)nwf,nrws,n1,n2,n3
01059       write(ifh) irws,rws,hrotr, ibasiwf
01060       close(ifh)
01061
01062
01063 !! other k-points
01064       write(ifbnd,*)ef,' ef'
01065       write(iftb,*)ef,' ef'
01066       if (lsh) write(ifsh,*)ef,' ef'
01067       allocate(eval1(nwf,nq),eval3(nwf,nq))
01068       if(lsh) allocate(eval2(nwf,nq))
01069       do iq = 1,nq
01070 c          write(6,*)' got get_hrotkp_ws iq =',iq
01071 c (3-3) Hrot_mn(k')
01072           call get_hrotkp_ws(hrotr,rws,drws,irws,q(:,iq),
01073      d                     nwf,nqbz,nrws,
01074      o                     hrotkp)
01075 c (3-4) diagonalize
01076           call diag_hm(hrotkp,nwf,eval,evecc)
01077           eval1(1:nwf,iq)=eval
01078 c (3-4) diagonalize  -- Small Hamiltonian --
01079           if (lsh) then
01080              hrotkps(1:nsh,1:nsh) = hrotkp(nsh1:nsh2,nsh1:nsh2)
01081              call diag_hm(hrotkps,nsh,evals,eveccs)
01082              write(ifsh,*)'iq =',iq
01083              write(ifsh,990)q(1:3,iq)
01084              eval2(1:nsh,iq)= evals(1:nsh)
01085           endif                  ! lsh
01086 c (3-3) Hrot_mn(k')  -- Tight-binding ---
01087           call get_hrotkp_tb_ws(rcut,plat,alat,
01088      i         hrotr,rws,drws,irws,q(:,iq),  ibasiwf,pos,natom,
01089      d         nwf,nqbz,nrws,
01090      o         hrotkp)
01091 c     (3-4) diagonalize -- Tight-binding --
01092           call diag_hm(hrotkp,nwf,eval,evecc)
01093           eval3(1:nwf,iq)=eval
01094       enddo
01095 !     ! write eval july2014takao
01096       do iband = 1,nwf
01097          do iq = 1,nq
01098             write(ifbnd,"(i5,3f13.5,'  ',f13.6,f13.6,i5,' !eee! x eval-ef(ev) iband' )")
```

```
01099      &           iq,q(1:3,iq),  xq(iq),(eval1(iband,iq)-ef)*rydberg(),iband
01100             write(iftb,"(i5,3f13.5,'  ',f13.6,f13.6,i5,' !eee! x eval-ef(ev) iband' )")
01101      &           iq,q(1:3,iq),  xq(iq),(eval3(iband,iq)-ef)*rydberg(),iband
01102          enddo
01103          write(ifbnd,*)
01104          write(iftb,*)
01105        enddo
01106        deallocate(eval1,eval3)
01107
01108        if(lsh) then
01109           do iband = 1,nsh
01110              do iq = 1,nq
01111                 write(ifsh,"(i5,3f13.5,'  ',f13.6,f13.6,i5,' !eee! x eval-ef(ev) iband' )")
01112      &             iq,q(1:3,iq),  xq(iq),(eval2(iband,iq)-ef)*rydberg(),iband
01113              enddo
01114           enddo
01115        endif
01116        call writeham(ifham,is,ef,alat,plat,pos,qbz,wbz,rws,irws,hrotk,nspin,natom,nwf,nqbz,nrws)
01117        deallocate(cnk,umnk,eunk,hrotk,hrotr,hrotkp,evecc,eval,irws,rws,drws)
01118        if (lsh) deallocate(hrotkps,eveccs,evals)
01119        close(ifbnd)
01120  c end of loop over spin
01121   1000 continue
01122   950  format(a14,3f23.16)
01123   990  format(3f12.6)
01124        call cputid(0)
01125        call rx0('hmaxloc: ixc=2 ok')
01126        end
01127
01128  c-------------------------------------------------------------------
01129        subroutine chk_amnkweight(qbz,iko_ix,iko_fx,amnk,
01130       &     nqbz,nwf,nband,nlmto)
01131        use m_readqg
01132        use m_readeigen
01133        use keyvalue
01134        implicit real*8(a-h,o-z)
01135
01136        complex(8) :: amnk(iko_ix:iko_fx,nwf,nqbz)
01137        complex(8),allocatable:: cphi1(:,:),cphi2(:,:)
01138        real(8) :: qbz(3,nqbz),q(3),quu(3)
01139        real(8),allocatable:: wbas(:,:)
01140        integer(4),allocatable::
01141       &  m_indx(:),n_indx(:),l_indx(:),ibas_indx(:)
01142
01143  c --- Readin nlam index
01144        ifoc = iopen('@MNLA_CPHI',1,0,0)
01145        ldim2 = nlmto
01146        read(ifoc,*)
01147        allocate(m_indx(ldim2),n_indx(ldim2),l_indx(ldim2),ibas_indx(ldim2))
01148        do ix =1,ldim2
01149          read(ifoc,*)m_indx(ix),n_indx(ix),l_indx(ix),ibas_indx(ix),ixx
01150          if(ixx/=ix) call rx('failed to readin @MNLA_CPHI')
01151        enddo
01152
01153        nbas = ibas_indx(nlmto)
01154        allocate(cphi1(nlmto,nband),cphi2(nlmto,nwf),wbas(nbas,nwf))
01155        wbas = 0d0
01156        cphi2=0d0
01157        do iq = 1,nqbz
01158        q = qbz(:,iq)
01159        call readcphi(q,nlmto,1,quu,cphi1)
01160
01161         do iwf=1,nwf
01162           do ib=iko_ix,iko_fx
01163              cphi2(:,iwf) = cphi2(:,iwf) + cphi1(:,ib)*amnk(ib,iwf,iq)
01164           enddo
01165        enddo
01166
01167        enddo ! iq
01168
01169        do iwf=1,nwf
01170           do ia=1,nlmto
01171              ibas = ibas_indx(ia)
01172              wbas(ibas,iwf) = wbas(ibas,iwf) +
01173       &                conjg(cphi2(ia,iwf))*cphi2(ia,iwf)
01174           enddo ! ia
01175        enddo ! iwf
01176        wbas = wbas / dble(nqbz**2)
01177
01178        write(*,*)'*** ibas,iwf,wbas'
01179        do iwf=1,nwf
01180        do ibas=1,nbas
01181           write(*,*)ibas,iwf,wbas(ibas,iwf)
01182        enddo
01183        write(*,*)
01184        enddo
01185        write(*,*)'*** ibas,wbas'
```

```
01186       do ibas=1,nbas
01187          w = 0d0
01188          do iwf=1,nwf
01189             w = w + wbas(ibas,iwf)
01190          enddo
01191          write(*,*)ibas,w
01192       enddo
01193
01194       deallocate(cphi1,cphi2,wbas,m_indx,l_indx,n_indx,ibas_indx)
01195       ix = iclose('@MNLA_CPHI')
01196
01197       end
01198 c--------------------------------------------------------------------
01199       subroutine chk_cnkweight(qbz,iko_ix,iko_fx,cnk,
01200      &     nqbz,nwf,nband,nlmto)
01201       use m_readqg
01202       use m_readeigen
01203       use keyvalue
01204       implicit real*8(a-h,o-z)
01205
01206       complex(8) :: cnk(iko_ix:iko_fx,nwf,nqbz)
01207       complex(8),allocatable:: cphi1(:,:),cphi2(:,:)
01208       real(8) :: qbz(3,nqbz),q(3),quu(3)
01209       real(8),allocatable:: wbas(:,:)
01210       integer(4),allocatable::
01211      &  m_indx(:),n_indx(:),l_indx(:),ibas_indx(:)
01212
01213 c --- Readin nlam index
01214       ifoc = iopen('@MNLA_CPHI',1,0,0)
01215       ldim2 = nlmto
01216       read(ifoc,*)
01217       allocate(m_indx(ldim2),n_indx(ldim2),l_indx(ldim2),ibas_indx(ldim2))
01218       do ix =1,ldim2
01219         read(ifoc,*)m_indx(ix),n_indx(ix),l_indx(ix),ibas_indx(ix),ixx
01220         if(ixx/=ix) call rx('failed to readin @MNLA_CPHI')
01221       enddo
01222
01223       nbas = ibas_indx(nlmto)
01224       allocate(cphi1(nlmto,nband),cphi2(nlmto,nwf),wbas(nbas,nwf))
01225       wbas = 0d0
01226       cphi2=0d0
01227
01228       do iq = 1,nqbz
01229       q = qbz(:,iq)
01230       call readcphi(q,nlmto,1,quu,cphi1)
01231
01232       do iwf=1,nwf
01233          do ib=iko_ix,iko_fx
01234             cphi2(:,iwf) = cphi2(:,iwf) + cphi1(:,ib)*cnk(ib,iwf,iq)
01235          enddo
01236       enddo
01237
01238       enddo ! iq
01239
01240       do iwf=1,nwf
01241          do ia=1,nlmto
01242             ibas = ibas_indx(ia)
01243             wbas(ibas,iwf) = wbas(ibas,iwf) +
01244      &                 conjg(cphi2(ia,iwf))*cphi2(ia,iwf)
01245          enddo ! ia
01246       enddo ! iwf
01247       wbas = wbas / dble(nqbz*nqbz)
01248
01249       write(*,*)'*** ibas,iwf,wbas'
01250       do iwf=1,nwf
01251       do ibas=1,nbas
01252          write(*,*)ibas,iwf,wbas(ibas,iwf)
01253       enddo
01254       write(*,*)
01255       enddo
01256
01257       write(*,*)'*** ibas,wbas'
01258       do ibas=1,nbas
01259          w = 0d0
01260          do iwf=1,nwf
01261             w = w + wbas(ibas,iwf)
01262          enddo
01263          write(*,*)ibas,w
01264       enddo
01265       deallocate(cphi1,cphi2,wbas,m_indx,l_indx,n_indx,ibas_indx)
01266       ix = iclose('@MNLA_CPHI')
01267       end
01268 c--------------------------------------------------------------------
01269       subroutine chk_umn(cnk,umnk,qbz,
01270      i                   iko_ix,iko_fx,iko_i,iko_f,
01271      d                   nwf,nqbz,nband,nlmto)
01272       use m_readqg
```

```
01273       use m_readeigen
01274       use keyvalue
01275       implicit real*8(a-h,o-z)
01276
01277       complex(8) :: cnk(iko_ix:iko_fx,nwf,nqbz),
01278      &              dnk(iko_ix:iko_fx,nwf,nqbz),
01279      &              umnk(nwf,nwf,nqbz)
01280       real(8) :: qbz(3,nqbz)
01281       integer(4) :: iko_i(nqbz),iko_f(nqbz)
01282
01283       dnk = (0d0,0d0)
01284       do iq = 1,nqbz
01285          do imp = iko_i(iq),iko_f(iq)
01286          do in = 1,nwf
01287             do im = 1,nwf
01288                dnk(imp,in,iq) = dnk(imp,in,iq)
01289      &                 + umnk(im,in,iq) * cnk(imp,im,iq)
01290             enddo ! im
01291          enddo ! in
01292          enddo ! imp
01293       enddo ! iq
01294
01295       call chk_cnkweight(qbz,iko_ix,iko_fx,dnk,
01296      &     nqbz,nwf,nband,nlmto)
01297
01298       end
```

4.43 /home/takao/ecalj/lm7K/run_arg File Reference

4.44 run_arg

```
00001 # T.Kotani Jan.2015
00002 # SeungWoo Jang Sep.2014
00003 echo_run=""                    # standard
00004 serial_run=""                  # standard
00005 #echo_run="aprun"                                    # cray
00006 #serial_run="aprun"                                  # cray
00007 function run_arg
00008 {
00009     local argin=$1
00010     local MPI_SIZE=$2
00011     local nfpgw=$3
00012     local command=$4
00013     local output=$5
00014     local TARGET=${@:6:($#-2)}
00015     local mpi_run="mpirun -np $MPI_SIZE"                          # standard
00016     #local pi_run="aprun -n $LSB_PROCS -d $LSB_CPUS -N $LSB_PPN"  # cray
00017     $echo_run echo -n 'OK! --> Start'
00018     $echo_run echo $argin > _IN_
00019     if [ $MPI_SIZE == '0' ]; then
00020         $echo_run echo " echo $argin | $nfpgw$command $TARGET > $output "
00021         $serial_run $nfpgw$command  $TARGET < _IN_ > $output
00022     else
00023         $echo_run echo " echo $argin | mpirun -np $MPI_SIZE $nfpgw$command $TARGET > $output "
00024         $mpi_run $nfpgw$command $TARGET < _IN_ > $output
00025     fi
00026     if [ $? != 0 ]; then
00027         $echo_run echo Error in $command input_arg=$argin. See OutputFile=$output
00028         exit 10
00029     fi
00030 }
00031
00032 echo "NOTE: Use run_arg defined in $nfpgw/run_arg"
00033
00034
00035 ### takao. This sometimes cause error. (only replace > with |tee
00036 # Because of hakozaki@kyushu-u.ac.jp ?
00037 # function run_arg_tee
00038 # {
00039 #     local argin=$1
00040 #     local MPI_SIZE=$2
00041 #     local nfpgw=$3
00042 #     local command=$4
00043 #     local output=$5
00044 #     local TARGET=${@:6:($#-2)}
00045 #     $echo_run echo -n 'OK! --> Start'
00046 #     $echo_run echo $argin > _IN_
00047 #     if [ $MPI_SIZE == '0' ]; then
00048 #         $echo_run echo " echo $argin | $nfpgw$command $TARGET |tee $output "
00049 #         $serial_run $nfpgw$command  $TARGET < _IN_ |tee $output
00050 #     else
```

```
00051 #         $echo_run echo " echo $argin | mpirun -np $MPI_SIZE $nfpgw$command $TARGET |tee $output "
00052 #         $mpi_run $nfpgw$command $TARGET < _IN_ |tee $output
00053 #     fi
00054 #     if [ $? != 0 ]; then
00055 #         $echo_run echo Error in $command input_arg=$argin. See OutputFile=$output
00056 #         exit 10
00057 #     fi
00058 # }
00059
```

Index