

ecalj/fpgw/ code document

0.1

Generated by Doxygen 1.8.6

Fri Nov 11 2016 17:43:19



# Contents

<b>1</b>	<b>Data Type Index</b>	<b>1</b>
1.1	Data Types List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Type Documentation</b>	<b>5</b>
3.1	m_anf Module Reference . . . . .	5
3.1.1	Detailed Description . . . . .	5
3.1.2	Member Function/Subroutine Documentation . . . . .	5
3.1.2.1	anfcond . . . . .	5
3.1.3	Member Data Documentation . . . . .	5
3.1.3.1	ibasf . . . . .	5
3.1.3.2	laf . . . . .	6
3.2	m_freq Module Reference . . . . .	6
3.2.1	Detailed Description . . . . .	6
3.2.2	Member Function/Subroutine Documentation . . . . .	7
3.2.2.1	getfreq . . . . .	7
3.2.3	Member Data Documentation . . . . .	7
3.2.3.1	freq_i . . . . .	7
3.2.3.2	freq_r . . . . .	7
3.2.3.3	frhis . . . . .	7
3.2.3.4	npm . . . . .	7
3.2.3.5	nw . . . . .	7
3.2.3.6	nw_i . . . . .	7
3.2.3.7	nwhis . . . . .	7
3.2.3.8	wiw . . . . .	8
3.3	m_genallcf_v3 Module Reference . . . . .	8
3.3.1	Detailed Description . . . . .	10
3.3.2	Member Function/Subroutine Documentation . . . . .	10
3.3.2.1	genallcf_v3 . . . . .	10
3.3.3	Member Data Documentation . . . . .	10

3.3.3.1	alat	10
3.3.3.2	clabl	11
3.3.3.3	delta	11
3.3.3.4	deltaw	11
3.3.3.5	done_genallcf_v3	11
3.3.3.6	ecore	11
3.3.3.7	esmr	11
3.3.3.8	iantiferro	11
3.3.3.9	iclass	11
3.3.3.10	icore	11
3.3.3.11	il	11
3.3.3.12	ilc	11
3.3.3.13	ilnm	11
3.3.3.14	ilnmc	12
3.3.3.15	ilnmv	12
3.3.3.16	ilv	12
3.3.3.17	im	12
3.3.3.18	imc	12
3.3.3.19	imv	12
3.3.3.20	in	12
3.3.3.21	inc	12
3.3.3.22	inv	12
3.3.3.23	invg	12
3.3.3.24	konf	12
3.3.3.25	natom	12
3.3.3.26	nclass	13
3.3.3.27	ncore	13
3.3.3.28	nctot	13
3.3.3.29	ncwf	13
3.3.3.30	ngrp	13
3.3.3.31	nindx	13
3.3.3.32	nindxc	13
3.3.3.33	nindxv	13
3.3.3.34	niw	13
3.3.3.35	nl	13
3.3.3.36	nlmto	13
3.3.3.37	nlnm	13
3.3.3.38	nlnmc	14
3.3.3.39	nlnmv	14
3.3.3.40	nlnmx	14

3.3.3.41	nlnmxc	14
3.3.3.42	nlnmxv	14
3.3.3.43	nlnx	14
3.3.3.44	nlnxc	14
3.3.3.45	nlnxv	14
3.3.3.46	nn	14
3.3.3.47	nnc	14
3.3.3.48	nnv	14
3.3.3.49	nocc	14
3.3.3.50	nspin	15
3.3.3.51	nunocc	15
3.3.3.52	occo	15
3.3.3.53	occv	15
3.3.3.54	plat	15
3.3.3.55	pos	15
3.3.3.56	spid	15
3.3.3.57	symgg	15
3.3.3.58	symgrp	15
3.3.3.59	unocco	15
3.3.3.60	unoccv	15
3.3.3.61	z	15
3.4	m_hamindex Module Reference	16
3.4.1	Detailed Description	17
3.4.2	Member Function/Subroutine Documentation	17
3.4.2.1	getikt	17
3.4.2.2	readhamindex	17
3.4.2.3	writehamindex	18
3.4.3	Member Data Documentation	18
3.4.3.1	ag	18
3.4.3.2	debug	18
3.4.3.3	dlmm	18
3.4.3.4	ibasindex	18
3.4.3.5	ibastab	18
3.4.3.6	iclasst	18
3.4.3.7	igmap	18
3.4.3.8	igv2	18
3.4.3.9	igv2rev	18
3.4.3.10	imx	18
3.4.3.11	invgx	19
3.4.3.12	iqimap	19

3.4.3.13	iqmap	19
3.4.3.14	ispec	19
3.4.3.15	ktab	19
3.4.3.16	kxx	19
3.4.3.17	ltab	19
3.4.3.18	lxx	19
3.4.3.19	lxxa	19
3.4.3.20	miat	19
3.4.3.21	napwk	19
3.4.3.22	napwmx	19
3.4.3.23	nbas	20
3.4.3.24	ndimham	20
3.4.3.25	ngpmx	20
3.4.3.26	ngrp	20
3.4.3.27	norbmto	20
3.4.3.28	norbtx	20
3.4.3.29	nqi	20
3.4.3.30	nqnum	20
3.4.3.31	nqtt	20
3.4.3.32	null	20
3.4.3.33	offl	20
3.4.3.34	offrev	20
3.4.3.35	plat	21
3.4.3.36	qlat	21
3.4.3.37	qq	21
3.4.3.38	qtt	21
3.4.3.39	qtti	21
3.4.3.40	shtvg	21
3.4.3.41	symops	21
3.4.3.42	tiat	21
3.5	m_readfermi Module Reference	21
3.5.1	Detailed Description	21
3.5.2	Member Function/Subroutine Documentation	22
3.5.2.1	readfermi	22
3.5.3	Member Data Documentation	22
3.5.3.1	bandgap	22
3.5.3.2	ef	22
3.6	m_readq0p Module Reference	22
3.6.1	Detailed Description	22
3.6.2	Member Function/Subroutine Documentation	22

3.6.2.1	readq0p	22
3.6.3	Member Data Documentation	23
3.6.3.1	ixyz	23
3.6.3.2	nq0i	23
3.6.3.3	nq0iadd	23
3.6.3.4	q0i	23
3.6.3.5	wgt0	23
3.6.3.6	wqt	23
3.7	m_readqg Module Reference	23
3.7.1	Detailed Description	25
3.7.2	Member Function/Subroutine Documentation	25
3.7.2.1	init_readqg	25
3.7.2.2	iqindx2qg	25
3.7.2.3	iswap	25
3.7.2.4	readngmx	26
3.7.2.5	readqg	26
3.7.2.6	readqg0	26
3.7.2.7	sortea	26
3.7.2.8	tabkk	27
3.7.3	Member Data Documentation	27
3.7.3.1	epsd	27
3.7.3.2	ginv_	27
3.7.3.3	init	27
3.7.3.4	iqkkk	27
3.7.3.5	iqkkkc	27
3.7.3.6	iqkkkp	27
3.7.3.7	keyc	27
3.7.3.8	keyp	27
3.7.3.9	kk1	27
3.7.3.10	kk1c	28
3.7.3.11	kk1p	28
3.7.3.12	kk2	28
3.7.3.13	kk2c	28
3.7.3.14	kk2p	28
3.7.3.15	kk3	28
3.7.3.16	kk3c	28
3.7.3.17	kk3p	28
3.7.3.18	ngc	28
3.7.3.19	ngcmx	28
3.7.3.20	ngp	28

3.7.3.21	ngpmx	28
3.7.3.22	ngvecc	29
3.7.3.23	ngvecp	29
3.7.3.24	nkey	29
3.7.3.25	nkeyc	29
3.7.3.26	nkeyp	29
3.7.3.27	nqnumc	29
3.7.3.28	nqnump	29
3.7.3.29	nqtt	29
3.7.3.30	qc	29
3.7.3.31	qp	29
3.7.3.32	qpgcut_cou	29
3.7.3.33	qpgcut_psi	29
3.7.3.34	qtt	30
3.8	m_sxcfsc Module Reference	30
3.8.1	Detailed Description	30
3.8.2	Member Function/Subroutine Documentation	30
3.8.2.1	sxcf_fal3_scz	30
3.8.2.2	weightset4intreal	33
3.9	m_tetwt Module Reference	33
3.9.1	Detailed Description	34
3.9.2	Member Function/Subroutine Documentation	34
3.9.2.1	gettetwt	34
3.9.2.2	tetdeallocate	34
3.9.3	Member Data Documentation	34
3.9.3.1	ibjb	34
3.9.3.2	ihw	34
3.9.3.3	jhw	34
3.9.3.4	n1b	35
3.9.3.5	n2b	35
3.9.3.6	nbnb	35
3.9.3.7	nbnbx	35
3.9.3.8	nhw	35
3.9.3.9	nhwtot	35
3.9.3.10	whw	35
3.10	m_zmel Module Reference	35
3.10.1	Detailed Description	36
3.10.2	Member Function/Subroutine Documentation	36
3.10.2.1	get_zmelt	36
3.10.2.2	get_zmelt2	37



3.10.3	Member Data Documentation	37
3.10.3.1	cmelt	37
3.10.3.2	cphim	37
3.10.3.3	cphiq	37
3.10.3.4	init	37
3.10.3.5	itq	37
3.10.3.6	kxold	37
3.10.3.7	miat	37
3.10.3.8	nband	37
3.10.3.9	ngcmx	37
3.10.3.10	ngpmx	38
3.10.3.11	ntq	38
3.10.3.12	null	38
3.10.3.13	ppbir	38
3.10.3.14	ppovlz	38
3.10.3.15	q_bk	38
3.10.3.16	qbasinv	38
3.10.3.17	qk_bk	38
3.10.3.18	rmelt	38
3.10.3.19	shtvg	38
3.10.3.20	tiat	38
3.10.3.21	zmel	38
3.10.3.22	zmeltt	39
<b>4</b>	<b>File Documentation</b>	<b>41</b>
4.1	exec/makefile File Reference	41
4.1.1	Variable Documentation	41
4.1.1.1	dep	41
4.1.1.2	doxygen	41
4.1.1.3	latex	41
4.1.1.4	NOTE	42
4.1.1.5	PLATFORM	42
4.1.1.6	programs	42
4.2	makefile	42
4.3	gwsr/genallcf_mod.F File Reference	49
4.3.1	Function/Subroutine Documentation	50
4.3.1.1	idxInmc	50
4.3.1.2	incor	50
4.3.1.3	nallow	50
4.3.1.4	nalwln	50

4.3.1.5	noflmt0	50
4.3.1.6	nofln	50
4.3.1.7	noflnm	50
4.4	genallcf_mod.F	50
4.5	gwsrc/m_anf.F File Reference	58
4.6	m_anf.F	58
4.7	gwsrc/m_freq.F File Reference	58
4.8	m_freq.F	59
4.9	gwsrc/m_hamindex.F File Reference	60
4.10	m_hamindex.F	61
4.11	gwsrc/m_tetwt.F File Reference	62
4.12	m_tetwt.F	62
4.13	gwsrc/m_zmel.F File Reference	65
4.13.1	Function/Subroutine Documentation	65
4.13.1.1	timeshowx	65
4.14	m_zmel.F	65
4.15	gwsrc/mkjp.F File Reference	70
4.15.1	Function/Subroutine Documentation	70
4.15.1.1	fac2m	70
4.15.1.2	genjh	70
4.15.1.3	intn_smpxxx	70
4.15.1.4	mkjb_4	71
4.15.1.5	mkjp_4	71
4.15.1.6	sigint_4	71
4.15.1.7	sigintan1	71
4.15.1.8	sigintpp	72
4.15.1.9	vcoulq_4	72
4.16	mkjp.F	72
4.17	gwsrc/mkqg.F File Reference	82
4.17.1	Function/Subroutine Documentation	83
4.17.1.1	mkqg2	83
4.18	mkqg.F	83
4.19	gwsrc/readqg.F File Reference	89
4.19.1	Function/Subroutine Documentation	89
4.19.1.1	readppv0	89
4.20	readqg.F	90
4.21	gwsrc/sxcf_fal2.F File Reference	94
4.21.1	Function/Subroutine Documentation	95
4.21.1.1	sxcf_fal3z	95
4.22	sxcf_fal2.F	95

4.23	gwsrc/sxcf_fal2.sc.F File Reference	109
4.23.1	Function/Subroutine Documentation	109
4.23.1.1	get_nwx	109
4.24	sxcf_fal2.sc.F	109
4.25	gwsrc/x0kf_v4h.F File Reference	126
4.25.1	Function/Subroutine Documentation	126
4.25.1.1	checkbelong	126
4.25.1.2	dpsion5	126
4.25.1.3	hilbertmat	126
4.25.1.4	wcutef	127
4.25.1.5	x0kf_v4hz	127
4.25.1.6	x0kf_v4hz_symmetrize	127
4.26	x0kf_v4h.F	127
4.27	main/hbasfp0.m.F File Reference	147
4.27.1	Function/Subroutine Documentation	147
4.27.1.1	hbasfp0_v2	147
4.28	hbasfp0.m.F	147
4.29	main/hsfp0.sc.m.F File Reference	150
4.29.1	Function/Subroutine Documentation	150
4.29.1.1	hsfp0_sc	151
4.29.1.2	zsecsym	151
4.30	hsfp0.sc.m.F	151
4.31	main/hvccfp0.m.F File Reference	167
4.31.1	Function/Subroutine Documentation	168
4.31.1.1	checkagree	168
4.31.1.2	diagcvh	168
4.31.1.3	hvccfp0	168
4.31.1.4	mkb0	168
4.31.1.5	mkradmatch	168
4.31.1.6	phimatch	168
4.31.1.7	pmatorth	169
4.31.1.8	zgesvdnn2	169
4.32	hvccfp0.m.F	169
4.33	main/hx0fp0.m.F File Reference	187
4.33.1	Function/Subroutine Documentation	187
4.33.1.1	diagno00	187
4.33.1.2	eclda_bh	187
4.33.1.3	eclda_pz	188
4.33.1.4	getsqovlp	188
4.33.1.5	hx0fp0	188

4.33.1.6	matcinvf	188
4.33.1.7	tr_chkwrite	188
4.33.1.8	wecqw	188
4.34	hx0fp0.m.F	188
4.35	main/hx0fp0.sc.m.F File Reference	216
4.35.1	Function/Subroutine Documentation	216
4.35.1.1	hx0fp0_sc	216
4.35.1.2	tr_chkwrite	217
4.36	hx0fp0.sc.m.F	217
4.37	main/qg4gw.m.F File Reference	233
4.37.1	Function/Subroutine Documentation	233
4.37.1.1	qg4gw	233
4.38	qg4gw.m.F	233
4.39	Wannier/genMLWF File Reference	235
4.39.1	Variable Documentation	235
4.39.1.1	argin	235
4.39.1.2	exists	235
4.39.1.3	if	236
4.39.1.4	syml	236
4.39.1.5	usage	236
4.40	genMLWF	236
4.41	Wannier/hmaxloc.F File Reference	238
4.41.1	Function/Subroutine Documentation	238
4.41.1.1	chk_amnkweight	238
4.41.1.2	chk_cnkweight	238
4.41.1.3	chk_umn	238
4.41.1.4	hmaxloc	238
4.42	hmaxloc.F	239
4.43	/home/takao/ecalj/lm7K/run_arg File Reference	254
4.44	run_arg	254

# Chapter 1

## Data Type Index

### 1.1 Data Types List

Here are the data types with brief descriptions:

<a href="#">m_anf</a>	Antiferro condition module. We have line AFcond at the bottom of 'LMTO' file. Currently(feb2016), only laf is used (thus AF symmetry is not used yet for hx0fp0_sc) To access laf, need to call <a href="#">anfcond()</a> in advance . . . . .	5
<a href="#">m_freq</a>	Frequency mesh generator . . . . .	6
<a href="#">m_genallcf_v3</a>	Get basic settings of crystal structure and nlm info . . . . .	8
<a href="#">m_hamindex</a>	This is in lm7K/subs/m_hamindex.F and in <a href="#">fpgw/gwsrc/m_hamindex.F</a> We will need to unify make system and source code in fpgw and lmf. norbtx is given in gwsrc/readeigen.F init_readeigen2	16
<a href="#">m_readfermi</a>	. . . . .	21
<a href="#">m_readq0p</a>	. . . . .	22
<a href="#">m_readqg</a>	Return QGcou and QGpsi == . . . . .	23
<a href="#">m_sxcfsc</a>	This module is only because name=name argument binding. No data . . . . .	30
<a href="#">m_tetwt</a>	Get the weights and index for tetrahedron method for the Lindhard function . . . . .	33
<a href="#">m_zmel</a>	Get the matrix element $zmel = ZO^{-1} \langle MPB \psi   \psi \rangle$ , 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..	35



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">exec/makefile</a>	42
<a href="#">gwsrc/genallcf_mod.F</a>	50
<a href="#">gwsrc/m_anf.F</a>	58
<a href="#">gwsrc/m_freq.F</a>	59
<a href="#">gwsrc/m_hamindex.F</a>	61
<a href="#">gwsrc/m_tetwt.F</a>	62
<a href="#">gwsrc/m_zmel.F</a>	65
<a href="#">gwsrc/mkjp.F</a>	72
<a href="#">gwsrc/mkqg.F</a>	83
<a href="#">gwsrc/readqg.F</a>	90
<a href="#">gwsrc/sxcf_fal2.F</a>	95
<a href="#">gwsrc/sxcf_fal2.sc.F</a>	109
<a href="#">gwsrc/x0kf_v4h.F</a>	127
<a href="#">main/hbasfp0.m.F</a>	147
<a href="#">main/hsfp0.sc.m.F</a>	151
<a href="#">main/hvccfp0.m.F</a>	169
<a href="#">main/hx0fp0.m.F</a>	188
<a href="#">main/hx0fp0.sc.m.F</a>	217
<a href="#">main/qg4gw.m.F</a>	233
<a href="#">Wannier/genMLWF</a>	236
<a href="#">Wannier/hmaxloc.F</a>	239
<a href="#">/home/takao/ecalj/lm7K/run_arg</a>	254





## Chapter 3

# Data Type Documentation

### 3.1 m\_anf Module Reference

Antiferro condition module. We have line AFcond at the bottom of 'LMTO' file. Currently(feb2016), only laf is used (thus AF symmetry is not used yet for hx0fp0\_sc) To access laf, need to call [anfcond\(\)](#) in advance.

#### Public Member Functions

- subroutine [anfcond](#) ()

#### Public Attributes

- logical, protected [laf](#)
- integer, dimension(:), allocatable, protected [ibasf](#)

#### 3.1.1 Detailed Description

Antiferro condition module. We have line AFcond at the bottom of 'LMTO' file. Currently(feb2016), only laf is used (thus AF symmetry is not used yet for hx0fp0\_sc) To access laf, need to call [anfcond\(\)](#) in advance.

Definition at line [5](#) of file [m\\_anf.F](#).

#### 3.1.2 Member Function/Subroutine Documentation

##### 3.1.2.1 subroutine m\_anf::anfcond ( )

Definition at line [14](#) of file [m\\_anf.F](#).

Here is the caller graph for this function:

#### 3.1.3 Member Data Documentation

##### 3.1.3.1 integer, dimension(:), allocatable, protected m\_anf::ibasf

Definition at line [8](#) of file [m\\_anf.F](#).

### 3.1.3.2 logical, protected m\_anf::laf

Definition at line 7 of file [m\\_anf.F](#).

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

- gwsrsrc/[m\\_anf.F](#)

## 3.2 m\_freq Module Reference

Frequency mesh generator.

### Public Member Functions

- subroutine [getfreq](#) (epsmode, realomega, imagomega, tetra, omg2max, wemax, niw, ua, mpi\_\_root)  
*Get data set for [m\\_freq](#). All arguments are input.*

### Public Attributes

- real(8), dimension(:),  
allocatable, protected [fhris](#)
- real(8), dimension(:),  
allocatable, protected [freq\\_r](#)
- real(8), dimension(:),  
allocatable, protected [freq\\_i](#)
- real(8), dimension(:),  
allocatable, protected [wiw](#)
- integer, protected [nwhis](#)
- integer, protected [npm](#)
- integer, protected [nw\\_i](#)
- integer, protected [nw](#)

### 3.2.1 Detailed Description

Frequency mesh generator.

- OUTPUT
  - fhris : histogram bins to accumulate 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](#).

## 3.2.2 Member Function/Subroutine Documentation

3.2.2.1 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*, real(8) *wemax*, 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

`dw*(nw_input-3)` then `!omg` is in unit of Hartree

Definition at line 19 of file `m_freq.F`.

Here is the caller graph for this function:

## 3.2.3 Member Data Documentation

3.2.3.1 real(8), dimension(:), allocatable, protected `m_freq::freq_i`

Definition at line 10 of file `m_freq.F`.

3.2.3.2 real(8), dimension(:), allocatable, protected `m_freq::freq_r`

Definition at line 10 of file `m_freq.F`.

3.2.3.3 real(8), dimension(:), allocatable, protected `m_freq::frhis`

Definition at line 10 of file `m_freq.F`.

3.2.3.4 integer, protected `m_freq::npm`

Definition at line 11 of file `m_freq.F`.

3.2.3.5 integer, protected `m_freq::nw`

Definition at line 11 of file `m_freq.F`.

3.2.3.6 integer, protected `m_freq::nw_i`

Definition at line 11 of file `m_freq.F`.

3.2.3.7 integer, protected `m_freq::nwhis`

Definition at line 11 of file `m_freq.F`.

---

Definition at line 10 of file m\_freq.F.

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

- `gwsrsrc/m_freq.F`

### 3.3 m\_genallcf\_v3 Module Reference

get basic settings of crystal structure and nlm info

## Public Member Functions

- subroutine `genallcf_v3` (incwfx)

## Public Attributes

- `character(120)`, protected [symgrp](#)
- `character(6)`, `dimension(:)`,  
allocatable, protected [clabl](#)
- `integer`, `dimension(:)`,  
allocatable, protected [iclass](#)
- `integer`, `dimension(:, :)`,  
allocatable, protected [nindxv](#)
- `integer`, `dimension(:, :)`,  
allocatable, protected [nindxc](#)
- `integer`, `dimension(:, :, :)`,  
allocatable, protected [ncwf](#)
- `integer`, `dimension(:)`,  
allocatable, protected [invg](#)
- `integer`, `dimension(:, :)`,  
allocatable, protected [il](#)
- `integer`, `dimension(:, :)`,  
allocatable, protected [in](#)
- `integer`, `dimension(:, :)`,  
allocatable, protected [im](#)
- `integer`, `dimension(:)`,  
allocatable, protected [ilnm](#)
- `integer`, `dimension(:)`,  
allocatable, protected [nlnm](#)
- `integer`, `dimension(:)`,  
allocatable, protected [ilv](#)
- `integer`, `dimension(:)`,  
allocatable, protected [inv](#)
- `integer`, `dimension(:)`,  
allocatable, protected [imv](#)
- `integer`, `dimension(:)`,  
allocatable, protected [ilnmv](#)
- `integer`, `dimension(:)`,  
allocatable, protected [nlnmv](#)
- `integer`, `dimension(:)`,  
allocatable, protected [ilc](#)

- integer, dimension(:), allocatable, protected [inc](#)
- integer, dimension(:), allocatable, protected [imc](#)
- integer, dimension(:), allocatable, protected [ilnmc](#)
- integer, dimension(:), allocatable, protected [nlncmc](#)
- integer, dimension(:, :), allocatable, protected [nindx](#)
- integer, dimension(:, :), allocatable, protected [konf](#)
- integer, dimension(:, :), allocatable, protected [icore](#)
- integer, dimension(:), allocatable, protected [ncore](#)
- integer, dimension(:, :, :), allocatable, protected [occv](#)
- integer, dimension(:, :, :), allocatable, protected [unoccv](#)
- integer, dimension(:, :, :), allocatable, protected [occc](#)
- integer, dimension(:, :, :), allocatable, protected [unoccc](#)
- integer, dimension(:, :, :), allocatable, protected [noccc](#)
- integer, dimension(:, :, :), allocatable, protected [nunc](#)
- integer, dimension(:, :, :), allocatable, protected [nunoccc](#)
- integer, dimension(:), allocatable, protected [iantiferro](#)
- integer, protected [nclass](#)
- integer, protected [natom](#)
- integer, protected [nspin](#)
- integer, protected [nl](#)
- integer, protected [nn](#)
- integer, protected [nnv](#)
- integer, protected [nnc](#)
- integer, protected [ngrp](#)
- integer, protected [nlmto](#)
- integer, protected [nlx](#)
- integer, protected [nlxv](#)
- integer, protected [nlxc](#)
- integer, protected [nlmx](#)
- integer, protected [nlmxv](#)
- integer, protected [nlmxc](#)
- integer, protected [nctot](#)
- real(8), dimension(:, :), allocatable, protected [plat](#)
- real(8), dimension(:, :), allocatable, protected [pos](#)
- real(8), dimension(:), allocatable, protected [z](#)
- real(8), dimension(:, :, :), allocatable, protected [symgg](#)
- real(8), protected [alat](#)
- real(8), protected [deltaw](#)

- logical, protected `done_genallcf_v3` =.false.
- character(8), dimension(:), allocatable, protected `spid`
- real(8), dimension(:,,:), allocatable `ecore`
- real(8) `delta`
- integer `niw`
- real(8) `esmr`

### 3.3.1 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 20 of file `genallcf_mod.F`.

### 3.3.2 Member Function/Subroutine Documentation

#### 3.3.2.1 subroutine `m_genallcf_v3::genallcf_v3` ( integer(4) *incwfx* )

BZ&

frequencies&

coulomb

product basis&

core&

dimensions and constants

combine nocc,nunocc,nindx

index for allowed core states

core energies

index for core and LMTO basis

Definition at line 69 of file `genallcf_mod.F`.

Here is the call graph for this function:

Here is the caller graph for this function:

### 3.3.3 Member Data Documentation

#### 3.3.3.1 real(8), protected `m_genallcf_v3::alat`

Definition at line 58 of file `genallcf_mod.F`.

3.3.3.2 `character(6), dimension(:), allocatable, protected m_genallcf_v3::clabl`

Definition at line 43 of file [genallcf\\_mod.F](#).

3.3.3.3 `real(8) m_genallcf_v3::delta`

Definition at line 63 of file [genallcf\\_mod.F](#).

3.3.3.4 `real(8), protected m_genallcf_v3::deltaw`

Definition at line 58 of file [genallcf\\_mod.F](#).

3.3.3.5 `logical, protected m_genallcf_v3::done_genallcf_v3 = .false.`

Definition at line 59 of file [genallcf\\_mod.F](#).

3.3.3.6 `real(8), dimension(:, :), allocatable m_genallcf_v3::ecore`

Definition at line 62 of file [genallcf\\_mod.F](#).

3.3.3.7 `real(8) m_genallcf_v3::esmr`

Definition at line 65 of file [genallcf\\_mod.F](#).

3.3.3.8 `integer, dimension(:), allocatable, protected m_genallcf_v3::iantiferro`

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.9 `integer, dimension(:), allocatable, protected m_genallcf_v3::iclass`

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.10 `integer, dimension(:, :), allocatable, protected m_genallcf_v3::icore`

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.11 `integer, dimension(:, :), allocatable, protected m_genallcf_v3::il`

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.12 `integer, dimension(:), allocatable, protected m_genallcf_v3::ilc`

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.13 `integer, dimension(:), allocatable, protected m_genallcf_v3::ilnm`

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.14 integer, dimension(:), allocatable, protected m\_genallcf\_v3::ilnmc

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.15 integer, dimension(:), allocatable, protected m\_genallcf\_v3::ilnmv

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.16 integer, dimension(:), allocatable, protected m\_genallcf\_v3::ilv

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.17 integer, dimension(:, :), allocatable, protected m\_genallcf\_v3::im

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.18 integer, dimension(:), allocatable, protected m\_genallcf\_v3::imc

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.19 integer, dimension(:), allocatable, protected m\_genallcf\_v3::imv

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.20 integer, dimension(:, :), allocatable, protected m\_genallcf\_v3::in

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.21 integer, dimension(:), allocatable, protected m\_genallcf\_v3::inc

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.22 integer, dimension(:), allocatable, protected m\_genallcf\_v3::inv

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.23 integer, dimension(:), allocatable, protected m\_genallcf\_v3::invg

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.24 integer, dimension(:, :), allocatable, protected m\_genallcf\_v3::konf

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.25 integer, protected m\_genallcf\_v3::natom

Definition at line 53 of file [genallcf\\_mod.F](#).



3.3.3.26 integer, protected m\_genallcf\_v3::nclass

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.27 integer, dimension(:), allocatable, protected m\_genallcf\_v3::ncore

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.28 integer, protected m\_genallcf\_v3::nctot

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.29 integer, dimension(:, :, :), allocatable, protected m\_genallcf\_v3::ncwf

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.30 integer, protected m\_genallcf\_v3::ngrp

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.31 integer, dimension(:, :), allocatable, protected m\_genallcf\_v3::nindx

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.32 integer, dimension(:, :), allocatable, protected m\_genallcf\_v3::nindxc

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.33 integer, dimension(:, :), allocatable, protected m\_genallcf\_v3::nindxv

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.34 integer m\_genallcf\_v3::niw

Definition at line 64 of file [genallcf\\_mod.F](#).

3.3.3.35 integer, protected m\_genallcf\_v3::nl

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.36 integer, protected m\_genallcf\_v3::nlmto

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.37 integer, dimension(:), allocatable, protected m\_genallcf\_v3::nlm

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.38 integer, dimension(:), allocatable, protected m\_genallcf\_v3::nlnmc

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.39 integer, dimension(:), allocatable, protected m\_genallcf\_v3::nlnmv

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.40 integer, protected m\_genallcf\_v3::nlnmx

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.41 integer, protected m\_genallcf\_v3::nlnmxc

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.42 integer, protected m\_genallcf\_v3::nlnmxv

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.43 integer, protected m\_genallcf\_v3::nlnx

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.44 integer, protected m\_genallcf\_v3::nlnxc

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.45 integer, protected m\_genallcf\_v3::nlnxv

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.46 integer, protected m\_genallcf\_v3::nn

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.47 integer, protected m\_genallcf\_v3::nnc

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.48 integer, protected m\_genallcf\_v3::nnv

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.49 integer, dimension(:, :, :), allocatable, protected m\_genallcf\_v3::nocc

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.50 integer, protected m\_genallcf\_v3::nspin

Definition at line 53 of file [genallcf\\_mod.F](#).

3.3.3.51 integer, dimension(:, :, :), allocatable, protected m\_genallcf\_v3::nunocc

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.52 integer, dimension(:, :, :), allocatable, protected m\_genallcf\_v3::occc

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.53 integer, dimension(:, :, :), allocatable, protected m\_genallcf\_v3::occv

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.54 real(8), dimension(:, :), allocatable, protected m\_genallcf\_v3::plat

Definition at line 56 of file [genallcf\\_mod.F](#).

3.3.3.55 real(8), dimension(:, :), allocatable, protected m\_genallcf\_v3::pos

Definition at line 56 of file [genallcf\\_mod.F](#).

3.3.3.56 character(8), dimension(:), allocatable, protected m\_genallcf\_v3::spid

Definition at line 60 of file [genallcf\\_mod.F](#).

3.3.3.57 real(8), dimension(:, :, :), allocatable, protected m\_genallcf\_v3::symgg

Definition at line 56 of file [genallcf\\_mod.F](#).

3.3.3.58 character(120), protected m\_genallcf\_v3::symgrp

Definition at line 42 of file [genallcf\\_mod.F](#).

3.3.3.59 integer, dimension(:, :, :), allocatable, protected m\_genallcf\_v3::unoccc

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.60 integer, dimension(:, :, :), allocatable, protected m\_genallcf\_v3::unoccv

Definition at line 44 of file [genallcf\\_mod.F](#).

3.3.3.61 real(8), dimension(:), allocatable, protected m\_genallcf\_v3::z

Definition at line 56 of file [genallcf\\_mod.F](#).

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

- [gwsr/genallcf\\_mod.F](#)

### 3.4 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`. `norbt` 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, protected `ngrp` =null
- integer, protected `lxx` =null
- integer, protected `kxx` =null
- integer, protected `norbmto` =null
- integer, protected `nbas`
- integer, protected `nqtt`
- integer, protected `ndimham` =null
- integer, dimension(:),  
allocatable, protected `ltab`
- integer, dimension(:),  
allocatable, protected `ktab`
- integer, dimension(:),  
allocatable, protected `offl`
- integer, dimension(:),  
allocatable, protected `ispec`
- integer, dimension(:),  
allocatable, protected `iclasst`
- integer, dimension(:,,:),  
allocatable, protected `offlrev`
- integer, dimension(:),  
allocatable, protected `ibastab`
- integer, dimension(:),  
allocatable, protected `iqimap`
- integer, dimension(:),  
allocatable, protected `iqmap`
- integer, dimension(:),  
allocatable, protected `igmap`
- integer, dimension(:),  
allocatable, protected `invgx`
- integer, dimension(:,,:),  
allocatable, protected `miat`
- integer, dimension(:),  
allocatable, protected `ibasindex`
- real(8), dimension(:,,:),  
allocatable, protected `symops`
- real(8), dimension(:,,:),  
allocatable, protected `ag`

- real(8), dimension(:, :, :), allocatable, protected [tiat](#)
- real(8), dimension(:, :), allocatable, protected [shtvg](#)
- real(8), dimension(:, :, :, :), allocatable, protected [dlmm](#)
- real(8), dimension(:, :), allocatable, protected [qq](#)
- real(8), dimension(3, 3), protected [plat](#)
- real(8), dimension(3, 3), protected [qlat](#)
- real(8), dimension(:, :), allocatable, protected [qtt](#)
- real(8), dimension(:, :), allocatable, protected [qtti](#)
- integer, dimension(:, :, :), allocatable, protected [igv2](#)
- integer, dimension(:), allocatable, protected [napwk](#)
- integer, dimension(:, :, :, :), allocatable, protected [igv2rev](#)
- integer, protected [napwmx](#) =null
- integer, protected [lxxa](#) =null
- integer [norbtx](#) =null
- integer [nqi](#)
- integer [nqnum](#)
- integer [ngpmx](#)
- integer [imx](#) =null

## Private Attributes

- integer, parameter, private [null](#) =-999999
- logical, private [debug](#) =.false.

### 3.4.1 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`.

### 3.4.2 Member Function/Subroutine Documentation

#### 3.4.2.1 integer function `m_hamindex::getikt` ( `real(8), dimension(3) qin` )

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

Definition at line 24 of file `m_hamindex.F`.

#### 3.4.2.2 subroutine `m_hamindex::readhamindex` ( )

read info for wave rotation.

Definition at line 65 of file `m_hamindex.F`.

#### 3.4.2.3 subroutine m\_hamindex::writehamindex ( )

write info for wave rotation.

Definition at line 43 of file [m\\_hamindex.F](#).

### 3.4.3 Member Data Documentation

#### 3.4.3.1 real(8), dimension(:,:), allocatable, protected m\_hamindex::ag

Definition at line 12 of file [m\\_hamindex.F](#).

#### 3.4.3.2 logical, private m\_hamindex::debug =.false. [private]

Definition at line 6 of file [m\\_hamindex.F](#).

#### 3.4.3.3 real(8), dimension(:,:,:), allocatable, protected m\_hamindex::dlmm

Definition at line 12 of file [m\\_hamindex.F](#).

#### 3.4.3.4 integer, dimension(:), allocatable, protected m\_hamindex::ibasindex

Definition at line 11 of file [m\\_hamindex.F](#).

#### 3.4.3.5 integer, dimension(:), allocatable, protected m\_hamindex::ibastab

Definition at line 10 of file [m\\_hamindex.F](#).

#### 3.4.3.6 integer, dimension(:), allocatable, protected m\_hamindex::iclasst

Definition at line 10 of file [m\\_hamindex.F](#).

#### 3.4.3.7 integer, dimension(:), allocatable, protected m\_hamindex::igmap

Definition at line 11 of file [m\\_hamindex.F](#).

#### 3.4.3.8 integer, dimension(:,:,:), allocatable, protected m\_hamindex::igv2

Definition at line 15 of file [m\\_hamindex.F](#).

#### 3.4.3.9 integer, dimension(:,:,:), allocatable, protected m\_hamindex::igv2rev

Definition at line 15 of file [m\\_hamindex.F](#).

#### 3.4.3.10 integer m\_hamindex::imx =null

Definition at line 20 of file [m\\_hamindex.F](#).

3.4.3.11 integer, dimension(:), allocatable, protected m\_hamindex::invgx

Definition at line 11 of file [m\\_hamindex.F](#).

3.4.3.12 integer, dimension(:), allocatable, protected m\_hamindex::iqimap

Definition at line 11 of file [m\\_hamindex.F](#).

3.4.3.13 integer, dimension(:), allocatable, protected m\_hamindex::iqmap

Definition at line 11 of file [m\\_hamindex.F](#).

3.4.3.14 integer, dimension(:), allocatable, protected m\_hamindex::ispec

Definition at line 10 of file [m\\_hamindex.F](#).

3.4.3.15 integer, dimension(:), allocatable, protected m\_hamindex::ktab

Definition at line 10 of file [m\\_hamindex.F](#).

3.4.3.16 integer, protected m\_hamindex::kxx =null

Definition at line 8 of file [m\\_hamindex.F](#).

3.4.3.17 integer, dimension(:), allocatable, protected m\_hamindex::ltab

Definition at line 10 of file [m\\_hamindex.F](#).

3.4.3.18 integer, protected m\_hamindex::lxx =null

Definition at line 8 of file [m\\_hamindex.F](#).

3.4.3.19 integer, protected m\_hamindex::lxxa =null

Definition at line 16 of file [m\\_hamindex.F](#).

3.4.3.20 integer, dimension(:, :), allocatable, protected m\_hamindex::miat

Definition at line 11 of file [m\\_hamindex.F](#).

3.4.3.21 integer, dimension(:), allocatable, protected m\_hamindex::napwk

Definition at line 15 of file [m\\_hamindex.F](#).

3.4.3.22 integer, protected m\_hamindex::napwmx =null

Definition at line 16 of file [m\\_hamindex.F](#).

3.4.3.23 integer, protected m\_hamindex::nbas

Definition at line 9 of file [m\\_hamindex.F](#).

3.4.3.24 integer, protected m\_hamindex::ndimham =null

Definition at line 9 of file [m\\_hamindex.F](#).

3.4.3.25 integer m\_hamindex::ngpmx

Definition at line 20 of file [m\\_hamindex.F](#).

3.4.3.26 integer, protected m\_hamindex::ngrp =null

Definition at line 8 of file [m\\_hamindex.F](#).

3.4.3.27 integer, protected m\_hamindex::norbmto =null

Definition at line 8 of file [m\\_hamindex.F](#).

3.4.3.28 integer m\_hamindex::norbtx =null

Definition at line 19 of file [m\\_hamindex.F](#).

3.4.3.29 integer m\_hamindex::nqi

Definition at line 20 of file [m\\_hamindex.F](#).

3.4.3.30 integer m\_hamindex::nqnum

Definition at line 20 of file [m\\_hamindex.F](#).

3.4.3.31 integer, protected m\_hamindex::nqtt

Definition at line 9 of file [m\\_hamindex.F](#).

3.4.3.32 integer, parameter, private m\_hamindex::null =-999999 [private]

Definition at line 5 of file [m\\_hamindex.F](#).

3.4.3.33 integer, dimension(:), allocatable, protected m\_hamindex::offl

Definition at line 10 of file [m\\_hamindex.F](#).

3.4.3.34 integer, dimension(:, :, :), allocatable, protected m\_hamindex::offlrev

Definition at line 10 of file [m\\_hamindex.F](#).



3.4.3.35 `real(8), dimension(3,3), protected m_hamindex::plat`

Definition at line 13 of file [m\\_hamindex.F](#).

3.4.3.36 `real(8), dimension(3,3), protected m_hamindex::qlat`

Definition at line 13 of file [m\\_hamindex.F](#).

3.4.3.37 `real(8), dimension(:, :), allocatable, protected m_hamindex::qq`

Definition at line 12 of file [m\\_hamindex.F](#).

3.4.3.38 `real(8), dimension(:, :), allocatable, protected m_hamindex::qtt`

Definition at line 14 of file [m\\_hamindex.F](#).

3.4.3.39 `real(8), dimension(:, :), allocatable, protected m_hamindex::qtti`

Definition at line 14 of file [m\\_hamindex.F](#).

3.4.3.40 `real(8), dimension(:, :), allocatable, protected m_hamindex::shtvg`

Definition at line 12 of file [m\\_hamindex.F](#).

3.4.3.41 `real(8), dimension(:, :, :), allocatable, protected m_hamindex::symops`

Definition at line 12 of file [m\\_hamindex.F](#).

3.4.3.42 `real(8), dimension(:, :, :), allocatable, protected 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](#)

## 3.5 `m_readfermi` Module Reference

### Public Member Functions

- subroutine [readfermi](#) ()

### Public Attributes

- `real(8)`, protected [bandgap](#)
- `real(8)` [ef](#)

#### 3.5.1 Detailed Description

Definition at line 1 of file [genallcf\\_mod.F](#).

### 3.5.2 Member Function/Subroutine Documentation

#### 3.5.2.1 subroutine `m_readefermi::readfermi` ( )

Definition at line 6 of file [genallcf\\_mod.F](#).

Here is the caller graph for this function:

### 3.5.3 Member Data Documentation

#### 3.5.3.1 `real(8)`, protected `m_readefermi::bandgap`

Definition at line 2 of file [genallcf\\_mod.F](#).

#### 3.5.3.2 `real(8)` `m_readefermi::ef`

Definition at line 3 of file [genallcf\\_mod.F](#).

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

- [gwsrc/genallcf\\_mod.F](#)

## 3.6 `m_readq0p` Module Reference

### Public Member Functions

- subroutine [readq0p](#) ( )

### Public Attributes

- `real(8)`, `dimension(:)`, allocatable, protected [wqt](#)
- `real(8)`, `dimension(:, :)`, allocatable, protected [wgt0](#)
- `real(8)`, `dimension(:, :)`, allocatable, protected [q0i](#)
- integer, protected [nq0i](#)
- integer, protected [nq0iadd](#)
- integer, `dimension(:)`, allocatable, protected [ixyz](#)

### 3.6.1 Detailed Description

Definition at line 1 of file [readqg.F](#).

### 3.6.2 Member Function/Subroutine Documentation

#### 3.6.2.1 subroutine `m_readq0p::readq0p` ( )

Definition at line 7 of file [readqg.F](#).

Here is the caller graph for this function:

### 3.6.3 Member Data Documentation

3.6.3.1 integer, dimension(:), allocatable, protected m\_readq0p::ixyz

Definition at line 4 of file [readqg.F](#).

3.6.3.2 integer, protected m\_readq0p::nq0i

Definition at line 3 of file [readqg.F](#).

3.6.3.3 integer, protected m\_readq0p::nq0iadd

Definition at line 3 of file [readqg.F](#).

3.6.3.4 real(8), dimension(:, :), allocatable, protected m\_readq0p::q0i

Definition at line 2 of file [readqg.F](#).

3.6.3.5 real(8), dimension(:, :), allocatable, protected m\_readq0p::wgt0

Definition at line 2 of file [readqg.F](#).

3.6.3.6 real(8), dimension(:), allocatable, protected m\_readq0p::wqt

Definition at line 2 of file [readqg.F](#).

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

- gwsrsrc/[readqg.F](#)

## 3.7 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 primitive vector.  $ginv$  is the inverse of  $plat$  (primitive 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\\_](#)

### 3.7.1 Detailed Description

Return QGcou and QGpsi ==.

Definition at line [55](#) of file [readqg.F](#).

### 3.7.2 Member Function/Subroutine Documentation

**3.7.2.1** 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 [164](#) of file [readqg.F](#).

Here is the call graph for this function:

Here is the caller graph for this function:

**3.7.2.2** 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 primitive vector. `ginv` is the inverse of `plat` (primitive translation vector). Use `kk1, kk2, kk3, nkey(1:3), iqkkk` to get `iqindx`.

Definition at line [344](#) of file [readqg.F](#).

Here is the call graph for this function:

Here is the caller graph for this function:

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

Definition at line [412](#) of file [readqg.F](#).

Here is the caller graph for this function:

3.7.2.4 subroutine `m_readqg::readngmx` ( `character*(*) key`, `integer(4) ngmx` )

Definition at line 72 of file `readqg.F`.

Here is the caller graph for this function:

3.7.2.5 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 93 of file `readqg.F`.

Here is the call graph for this function:

Here is the caller graph for this function:

3.7.2.6 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 130 of file `readqg.F`.

Here is the call graph for this function:

Here is the caller graph for this function:

3.7.2.7 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 391 of file `readqg.F`.

Here is the call graph for this function:

Here is the caller graph for this function:

**3.7.2.8** subroutine `m_readqg::tabkk` ( integer *kk<sub>in</sub>*, integer, dimension(*n*) *kktable*, integer *n*, integer *nout* )

Definition at line 296 of file [readqg.F](#).

Here is the caller graph for this function:

### 3.7.3 Member Data Documentation

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

Definition at line 64 of file [readqg.F](#).

**3.7.3.2** real(8), dimension(3,3), private `m_readqg::ginv` [private]

Definition at line 69 of file [readqg.F](#).

**3.7.3.3** logical, dimension(2), private `m_readqg::init` =.true. [private]

Definition at line 58 of file [readqg.F](#).

**3.7.3.4** integer, dimension(:, :,), pointer, private `m_readqg::iqkkk` [private]

Definition at line 65 of file [readqg.F](#).

**3.7.3.5** integer, dimension(:, :,), allocatable, target, private `m_readqg::iqkkkc` [private]

Definition at line 68 of file [readqg.F](#).

**3.7.3.6** integer, dimension(:, :,), allocatable, target, private `m_readqg::iqkkkp` [private]

Definition at line 67 of file [readqg.F](#).

**3.7.3.7** integer, dimension(:, :), allocatable, target, private `m_readqg::keyc` [private]

Definition at line 68 of file [readqg.F](#).

**3.7.3.8** integer, dimension(:, :), allocatable, target, private `m_readqg::keyp` [private]

Definition at line 67 of file [readqg.F](#).

**3.7.3.9** integer, dimension(:), pointer, private `m_readqg::kk1` [private]

Definition at line 65 of file [readqg.F](#).

3.7.3.10 integer, dimension(:), allocatable, target, private m\_readqg::kk1c [private]

Definition at line 68 of file readqg.F.

3.7.3.11 integer, dimension(:), allocatable, target, private m\_readqg::kk1p [private]

Definition at line 67 of file readqg.F.

3.7.3.12 integer, dimension(:), pointer, private m\_readqg::kk2 [private]

Definition at line 65 of file readqg.F.

3.7.3.13 integer, dimension(:), allocatable, target, private m\_readqg::kk2c [private]

Definition at line 68 of file readqg.F.

3.7.3.14 integer, dimension(:), allocatable, target, private m\_readqg::kk2p [private]

Definition at line 67 of file readqg.F.

3.7.3.15 integer, dimension(:), pointer, private m\_readqg::kk3 [private]

Definition at line 65 of file readqg.F.

3.7.3.16 integer, dimension(:), allocatable, target, private m\_readqg::kk3c [private]

Definition at line 68 of file readqg.F.

3.7.3.17 integer, dimension(:), allocatable, target, private m\_readqg::kk3p [private]

Definition at line 67 of file readqg.F.

3.7.3.18 integer(4), dimension(:), allocatable, private m\_readqg::ngc [private]

Definition at line 61 of file readqg.F.

3.7.3.19 integer(4), target, private m\_readqg::ngcmx [private]

Definition at line 60 of file readqg.F.

3.7.3.20 integer(4), dimension(:), allocatable, private m\_readqg::ngp [private]

Definition at line 61 of file readqg.F.

3.7.3.21 integer(4), target, private m\_readqg::ngpmx [private]

Definition at line 60 of file readqg.F.



3.7.3.22 integer(4), dimension(:, :, :), allocatable, private m\_readqg::ngvecc [private]

Definition at line 61 of file [readqg.F](#).

3.7.3.23 integer(4), dimension(:, :, :), allocatable, private m\_readqg::ngvecp [private]

Definition at line 61 of file [readqg.F](#).

3.7.3.24 integer, dimension(:), pointer, private m\_readqg::nkey [private]

Definition at line 65 of file [readqg.F](#).

3.7.3.25 integer, dimension(3), target, private m\_readqg::nkeyc [private]

Definition at line 66 of file [readqg.F](#).

3.7.3.26 integer, dimension(3), target, private m\_readqg::nkeyp [private]

Definition at line 66 of file [readqg.F](#).

3.7.3.27 integer(4), target, private m\_readqg::nqnumc [private]

Definition at line 60 of file [readqg.F](#).

3.7.3.28 integer(4), target, private m\_readqg::nqnump [private]

Definition at line 60 of file [readqg.F](#).

3.7.3.29 integer, pointer, private m\_readqg::nqtt [private]

Definition at line 62 of file [readqg.F](#).

3.7.3.30 real(8), dimension(:, :, :), allocatable, target, private m\_readqg::qc [private]

Definition at line 57 of file [readqg.F](#).

3.7.3.31 real(8), dimension(:, :, :), allocatable, target, private m\_readqg::qp [private]

Definition at line 57 of file [readqg.F](#).

3.7.3.32 real(8), private m\_readqg::qpgcut\_cou [private]

Definition at line 59 of file [readqg.F](#).

3.7.3.33 real(8), private m\_readqg::qpgcut\_psi [private]

Definition at line 59 of file [readqg.F](#).

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

Definition at line 63 of file [readqg.F](#).

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

- [gwsr/readqg.F](#)

## 3.8 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*, *dwdummy*, *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*, *nwx\_i*, *nt\_max*, *wfacut*, *wtt*, *we\_*, *wfac\_*, *ixss*, *ititpskip*, *iirx*)

### 3.8.1 Detailed Description

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

Definition at line 2 of file [sxcf\\_fal2.sc.F](#).

### 3.8.2 Member Function/Subroutine Documentation

3.8.2.1 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) *dwdummy*, *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  $\text{simga}_{ij}(e_i) = \langle i | \text{Re}[\text{Sigma}](e_i) | j \rangle$

#### Parameters

<i>exchange</i>	<ul style="list-style-type: none"><li>• T : Calculate the exchange self-energy</li><li>• F : Calculate correlated part of the self-energy</li></ul>
<i>zsec</i>	<ul style="list-style-type: none"><li>• <math>S_{ij} = \langle i   \text{Re}[S](e_i)   j \rangle</math></li><li>• Note that <math>S_{ij}</math> itself is not Hermite becasue it includes <math>e_i</math>. <math>i</math> and <math>j</math> are band indexes</li></ul>
<i>coh</i>	dummy
<i>screen</i>	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,itpp)) ) ) !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  $\langle i | \text{Re}[S](e_i) | j \rangle$ .  
( In the version until Jan2006 (fpgw032f8), only the im-axis part was symmetrized.  
But it was not necessary from the beginning because it was done in hqpe.sc.F

(Be careful as for the difference between  
 $\langle i | \text{Re}[S](e_i) | j \rangle$  and transpose(dconjg( $\langle i | \text{Re}[S](e_i) | j \rangle$ )).  
---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 jobsw==2, and  
mode A jobsw==3 (SE\_nn'(e\_n)+SE\_nn'(e\_n'))/2 (Usually used 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 often).

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], Oct. 2007.

[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].

```

---Integration along imaginary axis.---
( Current version for it, wintzsg_npm, do not assume time-reversal when npm=2.)
Integration along the imaginary axis: -----
(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)

---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.
-----

-----
q      =qip(:,iq) = q-vector in SEc(q,t).
itq    = states t at q
ntq    = no. states t
eq     = eigenvalues at q
ef     = fermi level in Rydberg
WVI, WVR: direct access files for W. along im axis (WVI) or along real axis (WVR)
freq_r(nw_i:nw) = frequencies along real axis. freq_r(0)=0d0

qbas   = base reciprocal lattice vectors
ginv   = inverse of qbas s. indxrkf

wk     = weight for each k-point in the FBZ
qbz    = k-points in the 1st BZ

wx     = weights at gaussian points x between (0,1)
ua_    = constant in exp(-ua^2 w'^2) s. wint.f
expa   = exp(-ua^2 w'^2) s. wint.f

irkip(k,R,nq) = gives index in the FBZ with k{IBZ, R=rotation

nqibz  = number of k-points in the irreducible BZ
nqbz   = full BZ
natom  = number of atoms
nctot  = total no. of allowed core states
nbloch = total number of Bloch basis functions
nlmto  = total number of MTO+lo basis functions
ngrp   = no. group elements (rotation matrices)
niw    = no. frequencies along the imaginary axis
nw_i:nw = no. frequencies along the real axis. nw_i=0 or -nw.
zsec(itp,itpp,iq)> = <psi(itp,q(:,iq)) |SEc| psi(iq,q(:,iq)>
-----

```

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:

```
3.8.2.2  subroutine m_sxcpsc::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) nwx_i, 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 1247 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:

- [gwsr/sxcf\\_fal2.sc.F](#)

## 3.9 m\_tetwt Module Reference

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

### Public Member Functions

- subroutine [tetdeallocate](#) ()
- subroutine [gettetwt](#) (q, iq, is, isf, nwgt, frhis, nwhis, nrm,

### Public Attributes

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

### 3.9.1 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](#).

### 3.9.2 Member Function/Subroutine Documentation

3.9.2.1 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*, `real(8)`, `dimension(1:nwhis+1)`, `intent(in)` *frhis*, `integer`, `intent(in)` *nwhis*, `integer`, `intent(in)` *npm* )

Definition at line 22 of file [m\\_tetwt.F](#).

Here is the call graph for this function:

Here is the caller graph for this function:

3.9.2.2 subroutine `m_tetwt::tetdeallocate` ( )

Definition at line 17 of file [m\\_tetwt.F](#).

Here is the caller graph for this function:

### 3.9.3 Member Data Documentation

3.9.3.1 `integer`, `dimension(:, :, :)`, allocatable, protected `m_tetwt::ibjb`

Definition at line 12 of file [m\\_tetwt.F](#).

3.9.3.2 `integer`, `dimension(:, :, :)`, allocatable, protected `m_tetwt::ihw`

Definition at line 12 of file [m\\_tetwt.F](#).

3.9.3.3 `integer`, `dimension(:, :, :)`, allocatable, protected `m_tetwt::jhw`

Definition at line 12 of file [m\\_tetwt.F](#).

3.9.3.4 integer, dimension(:, :, :), allocatable, protected m\_tetwt::n1b

Definition at line 14 of file [m\\_tetwt.F](#).

3.9.3.5 integer, dimension(:, :, :), allocatable, protected m\_tetwt::n2b

Definition at line 14 of file [m\\_tetwt.F](#).

3.9.3.6 integer, dimension(:, :, :), allocatable, protected m\_tetwt::nbnb

Definition at line 14 of file [m\\_tetwt.F](#).

3.9.3.7 integer, protected m\_tetwt::nbnbx

Definition at line 13 of file [m\\_tetwt.F](#).

3.9.3.8 integer, dimension(:, :, :), allocatable, protected m\_tetwt::nhw

Definition at line 12 of file [m\\_tetwt.F](#).

3.9.3.9 integer, protected m\_tetwt::nhwtot

Definition at line 13 of file [m\\_tetwt.F](#).

3.9.3.10 real(8), dimension(:), allocatable, protected 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](#)

## 3.10 m\_zmel Module Reference

Get the matrix element  $zmel = ZO^{-1} \langle MPB \psi | \psi \rangle$ , 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

### 3.10.1 Detailed Description

Get the matrix element  $zmel = ZO^{-1} \langle MPB \psi | \psi \rangle$ , 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](#).

### 3.10.2 Member Function/Subroutine Documentation

- 3.10.2.1 subroutine [m\\_zmel::get\\_zmelt](#) ( logical *exchange*, real(8), dimension(3) *q*, integer *kx*, real(8), dimension(3) *kvec*, integer *iro*, 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:



### 3.10.2.2 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:

## 3.10.3 Member Data Documentation

### 3.10.3.1 real(8), dimension(:, :, :), allocatable, private m\_zmel::cmelt [private]

Definition at line 55 of file [m\\_zmel.F](#).

### 3.10.3.2 complex(8), dimension(:, :, :), allocatable, private m\_zmel::cphim [private]

Definition at line 54 of file [m\\_zmel.F](#).

### 3.10.3.3 complex(8), dimension(:, :, :), allocatable, private m\_zmel::cphiq [private]

Definition at line 54 of file [m\\_zmel.F](#).

### 3.10.3.4 logical, private m\_zmel::init =.true. [private]

Definition at line 53 of file [m\\_zmel.F](#).

### 3.10.3.5 integer, dimension(:), allocatable m\_zmel::itq

Definition at line 43 of file [m\\_zmel.F](#).

### 3.10.3.6 integer, private m\_zmel::kxold =-9999 [private]

Definition at line 56 of file [m\\_zmel.F](#).

### 3.10.3.7 integer, dimension(:, :, :), allocatable m\_zmel::miat

Definition at line 39 of file [m\\_zmel.F](#).

### 3.10.3.8 integer m\_zmel::nband =NULL

Definition at line 42 of file [m\\_zmel.F](#).

### 3.10.3.9 integer m\_zmel::ngcmx =NULL

Definition at line 42 of file [m\\_zmel.F](#).

3.10.3.10 integer m\_zmel::ngpmx =NULL

Definition at line 42 of file [m\\_zmel.F](#).

3.10.3.11 integer m\_zmel::ntq =NULL

Definition at line 42 of file [m\\_zmel.F](#).

3.10.3.12 integer, parameter m\_zmel::null =-99999

Definition at line 37 of file [m\\_zmel.F](#).

3.10.3.13 real(8), dimension(:,:), allocatable m\_zmel::ppbir

Definition at line 44 of file [m\\_zmel.F](#).

3.10.3.14 complex(8), dimension(:,:), allocatable, target m\_zmel::ppovlz

Definition at line 45 of file [m\\_zmel.F](#).

3.10.3.15 real(8), dimension(3), private m\_zmel::q\_bk =1d10 [private]

Definition at line 52 of file [m\\_zmel.F](#).

3.10.3.16 real(8), dimension(3,3), private m\_zmel::qbasinv [private]

Definition at line 52 of file [m\\_zmel.F](#).

3.10.3.17 real(8), dimension(3), private m\_zmel::qk\_bk =1d0 [private]

Definition at line 52 of file [m\\_zmel.F](#).

3.10.3.18 real(8), dimension(:,:), allocatable, private m\_zmel::rmelt [private]

Definition at line 55 of file [m\\_zmel.F](#).

3.10.3.19 real(8), dimension(:,:), allocatable m\_zmel::shtvg

Definition at line 40 of file [m\\_zmel.F](#).

3.10.3.20 real(8), dimension(:,:), allocatable m\_zmel::tiat

Definition at line 40 of file [m\\_zmel.F](#).

3.10.3.21 complex(8), dimension(:,:), allocatable m\_zmel::zmel

Definition at line 49 of file [m\\_zmel.F](#).

---

3.10.3.22 `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/makefile File Reference

#### Variables

- PLATFORM
- doxygen
- cd latex
- make echo fpgw latex refman pdf generated dep
- make echo fpgw latex refman pdf generated and read CallCaller sh echo echo Now generating a file callcaller dat Wait!!It takes minute or so echo If you like to apply this to other programs
- make echo fpgw latex refman pdf generated and read CallCaller sh echo echo Now generating a file callcaller dat Wait!!It takes minute or so echo If you like to apply this to other modify this script echo NOTE

#### 4.1.1 Variable Documentation

##### 4.1.1.1 make echo fpgw latex refman pdf generated dep

Definition at line 66 of file [makefile](#).

##### 4.1.1.2 doxygen

Definition at line 61 of file [makefile](#).

##### 4.1.1.3 cd latex

Definition at line 61 of file [makefile](#).

4.1.1.4 make echo fpgw latex refman pdf generated and read CallCaller sh echo echo Now generating a file callcaller dat Wait !  
It takes minute or so echo If you like to apply this to other modify this script echo NOTE

Definition at line 66 of file [makefile](#).

#### 4.1.1.5 PLATFORM

Definition at line 9 of file [makefile](#).

4.1.1.6 make echo fpgw latex refman pdf generated and read CallCaller sh echo echo Now generating a file callcaller dat Wait !  
It takes minute or so echo If you like to apply this to other programs

Definition at line 66 of file [makefile](#).

## 4.2 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=ifort
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 include make.inc.$(PLATFORM)
00016
00017 BINDIR = $(HOME)/bin
00018
00019 #-----
00020 # src directories
00021 main    = ../main/
00022 gwsrcc  = ../gwsrcc/
00023 tote    = ../tote/
00024 tags    = ../
00025
00026 #maxloc = ../Miyake/maxloc/
00027 # tag directory
00028 #
00029 #progs = hbasfp0 hvccfp0 hx0fp0 hsf0p0 hef hqpe hchknw qg4gw gwinit heftet hmergewv hparainfo hbndout
00030         rdata4gw_v2 convgwin hx0fp0_sc hsf0p0_sc hqpe_sc kino_input_test hecor eout eout2 h_uumatrix hsigmconv
00031 # lmf_exec
00032 #progs = hbasfp0 hvccfp0 hx0fp0 hsf0p0 hef hqpe hchknw qg4gw gwinit heftet hmergewv hparainfo hbndout
00033         rdata4gw_v2 convgwin hx0fp0_sc hsf0p0_sc hqpe_sc kino_input_test hecor eout eout2
00034
00035 # progs = hbasfp0 hvccfp0 hx0fp0 hsf0p0 hef hqpe hchknw qg4gw gwinit heftet hmergewv hbndout rdata4gw_v2
00036         convgwin hx0fp0_sc hsf0p0_sc hqpe_sc kino_input_test hecor eout eout2 h_uumatrix hsigmconv hwmatt hmaxloc huumat
00037         qpwf hpsig hnocc_mlw hx0fp0_mlw hphig
00038
00039 # hmaxloc1D
00040 progs2 = $(progs) $(tags)TAGS
00041 #checkmod
00042
00043 #script = cleargw* dcpu dtote eps* ex* gw* hqpemetal* inf* lmgw* plotg save* tote_lmfh2 xqp mkG*
00044 script = cleargw* dcpu eps* gw* mkG*
00045
00046 ##### You can choose these options. all is default.
00047
00048 all :$(progs2)
00049
00050 clean:
00051         rm -f $(progs)
00052
00053 install:
00054         cp $(progs) $(BINDIR)
```

```

00055         cp $(script) $(BINDIR)
00056
00057 cleanall:
00058     rm -f $(progs2) $(main)*.o $(gwsrc)*.o *.mod $(tote)*.o
00059
00060 doxygen:
00061     cd $(tags);doxygen;cd ../latex;make
00062     echo 'fpgw/latex/refman.pdf generated'
00063
00064 dep:
00065     @echo 'This generate a call-caller data set for fpgw/'
00066     @echo 'HELP --> ../TOOLS/FparserTools/f_calltree.py --help, and read CallCaller.sh'
00067     @echo
00068     @echo '--- Now generating a file 'callcaller.dat' ... Wait!!! It takes 1 minute or so!'
00069     @echo '         If you like to apply this to other programs, modify this script'
00070     @echo ' NOTE: T.Kotani is not sure whether this is reliable enough or not... let me know
something wrong...'
00071     $(tags)/../TOOLS/FparserTools/f_calltree.py $(main)/*.F $(gwsrc)/*.F $(tote)/*.F >callcaller.dat
00072     2>callcaller.err
00073     -egrep -e '^ (ERROR|Error)' callcaller.err
00074     @echo '-----'
00075     @echo '--- If no ERROR is shown above (if ERROR is not in callcaller.err), it is succeeded. ---'
00076     @echo '         Note that Unused files might be used by other mainprogram.'
00077     @echo '--- If ERROR is shown above, look into callcaller.err. Something wrong.'
00078     @echo
00079     @echo ' If you want to make a callcaller-tree picture, try'
00080     @echo ' >GenCCTree.sh callcaller.dotdata'
00081     @echo ' --> Then you get ccmmap.ps.; it is better to use smaller callcaller.dotdata(need to modify
this script to make it).'
00082     @echo ' Note that you need graphviz for GenCCTree.sh. as apt-get install graphviz'
00083
00084 # This is necesaly to compile *.f in right order.
00085 # When you recompile and link, just repeat 'make' (not necessary to repeat 'make init').
00086 # When checkmodule recompile source, you have to repeat 'make'.
00087 init:
00088     rm -f $(main)time_hsf0.sc.m.F
00089     rm -f $(main)time_hx0fp0.sc.m.F
00090     rm -f $(gwsrc)time_sxcf_fal2.sc.F
00091     rm -f $(gwsrc)time_rppov1.F
00092     rm -f $(gwsrc)time_x0kf_v4h.F
00093     rm -f $(gwsrc)time_ppbafp.fal.F
00094     exec ../../TOOLS/checkmodule ../gwsrc/*.F ../main/*.F ../tote/*.F
00095
00096 checkmod:
00097     init
00098     # ../../lm7K/subs/m_hamindex.F
00099     # m_hamindex
00100
00101 ## tete (total energy) #####
00102 ## these are experimental code ###
00103 ECOR = \
00104 $(tote)hecor.o
00105
00106 EO= \
00107 $(tote)eout.o \
00108
00109 EO2= \
00110 $(tote)eout2.o
00111
00112 hecor: $(ECOR) $(GWLIB) $(MPI) $(GWLIB) $(COMM)
00113         $(LK) $(LKFLAGS1) $(ECOR) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00114
00115 eout: $(EO) $(GWLIB) $(MPI)
00116         $(LK) $(LKFLAGS1) $(EO) $(GWLIB) $(MPI) $(LKFLAGS2) -o $@
00117
00118 eout2: $(EO2) $(GWLIB) $(MPI)
00119         $(LK) $(LKFLAGS1) $(EO2) $(GWLIB) $(MPI) $(LKFLAGS2) -o $@
00120 #####
00121
00122
00123 # BNDCONN= \
00124 # $(gwsrc)bndconn.o ### This is not linked but bndconn.o is used in lm/lmgw.
00125 # It is now included in lm/gw/
00126 DERFC=
00127 # $(gwsrc)derfc.o
00128 #         $(gwsrc)dilmach.o \
00129 #         $(gwsrc)ilmach.o
00130 #
00131 # test_genallcf = \
00132 # $(main)test_genallcf.o \
00133 # $(gwsrc)genallcf_dump.o \
00134 # $(GWLIB)
00135
00136
00137 kino_input_test = \
00138 $(main)kino_input_test.o

```

```

00139
00140 convg = \
00141 $(main)convgwin.o
00142
00143 GWINIT = \
00144 $(main)gwinit.m.o
00145
00146 QG = \
00147 $(main)qg4gw.m.o
00148
00149 RDAT_v2 = \
00150 $(main)rdata4gw_v2.m.o
00151
00152 BAS = \
00153 $(main)hbasfp0.m.o
00154
00155 VCC= \
00156 $(main)hvccfp0.m.o
00157
00158 SXC_SC = \
00159 $(main)hsfp0.sc.m.o
00160
00161 SXC = \
00162 $(main)hsfp0.m.o
00163
00164 # WMAT = \
00165 # $(maxloc)hwmato \
00166 # $(maxloc)maxloc0.o \
00167 # $(maxloc)wmat.o
00168
00169 # MLOC = \
00170 # $(maxloc)hmaxloc.o \
00171 # $(maxloc)maxloc0.o \
00172 # $(maxloc)maxloc1.o \
00173 # $(maxloc)maxloc2.o \
00174 # $(maxloc)maxloc3.o
00175
00176 # MLOC1D = \
00177 # $(maxloc)hmaxloc1D.o \
00178 # $(maxloc)maxloc0.o \
00179 # $(maxloc)maxloc1.o \
00180 # $(maxloc)maxloc2.o \
00181 # $(maxloc)maxloc3.o
00182
00183 heftet = \
00184 $(main)heftet.m.o
00185
00186 # hnocc_mlw = \
00187 # $(maxloc)hnocc_mlw.o
00188
00189 hef = \
00190 $(main)hef.m.o
00191
00192
00193 X0_SC = \
00194 $(main)hx0fp0.sc.m.o
00195
00196 X0 = \
00197 $(main)hx0fp0.m.o
00198
00199 # X0mlw = \
00200 # $(maxloc)hx0fp0.m.o \
00201 # $(maxloc)wcf.o \
00202 # $(gwsrctetwt5$(tet5_g) \
00203 # $(gwsrctetwt.o \
00204 # $(gwsrctdiagcv2.o \
00205 # $(tote)rpaq.o \
00206 # $(gwsrctcinvr.o \
00207 # $(gwsrctm_freq.o
00208 #
00209 # UU = \
00210 # $(main)h_uumatrix.m.o \
00211 # $(gwsrctwcf.o \
00212 # $(gwsrctetwt5$(tet5_g) \
00213 # $(gwsrctgintxx.o \
00214 # $(gwsrctpplmat.o \
00215 # $(gwsrctgetgv2.o \
00216 # $(gwsrctx0kf_v4h$(x0kf_g) \
00217 # $(gwsrctrs.o \
00218 # $(gwsrctu_lat_0.o \
00219 # $(gwsrctwronkj.o \
00220 # $(gwsrctmklegw.o \
00221 # $(gwsrctbessl.o \
00222 # $(gwsrctcross.o \
00223 # $(gwsrctdiagcv2.o
00224 #
00225 # UU2 = \

```



```

00226 # $(maxloc)huumat.o \
00227 # $(gwsrc)wcf.o \
00228 # $(gwsrc)tetwt5$(tet5_g) \
00229 # $(gwsrc)gintxx.o \
00230 # $(gwsrc)pplmat.o \
00231 # $(gwsrc)getgv2.o \
00232 # $(gwsrc)rs.o \
00233 # $(gwsrc)u_lat_0.o \
00234 # $(gwsrc)wronkj.o \
00235 # $(gwsrc)mklegw.o \
00236 # $(gwsrc)bessl.o \
00237 # $(gwsrc)cross.o
00238 #
00239 # PSIG = \
00240 # $(maxloc)hpsig.o \
00241 # $(gwsrc)wcf.o \
00242 # $(gwsrc)tetwt5$(tet5_g) \
00243 # $(gwsrc)m_tetwt.o \
00244 # $(gwsrc)gintxx.o \
00245 # $(gwsrc)pplmat.o \
00246 # $(gwsrc)getgv2.o \
00247 # $(gwsrc)rs.o \
00248 # $(gwsrc)u_lat_0.o \
00249 # $(gwsrc)wronkj.o \
00250 # $(gwsrc)mklegw.o \
00251 # $(gwsrc)bessl.o \
00252 # $(gwsrc)cross.o
00253 #
00254 # PHIG = \
00255 # $(maxloc)hphig.o \
00256 # $(gwsrc)wcf.o \
00257 # $(gwsrc)tetwt5$(tet5_g) \
00258 # $(gwsrc)m_tetwt.o \
00259 # $(gwsrc)gintxx.o \
00260 # $(gwsrc)pplmat.o \
00261 # $(gwsrc)getgv2.o \
00262 # $(gwsrc)rs.o \
00263 # $(gwsrc)u_lat_0.o \
00264 # $(gwsrc)wronkj.o \
00265 # $(gwsrc)mklegw.o \
00266 # $(gwsrc)bessl.o \
00267 # $(gwsrc)cross.o
00268
00269 MPI = $(gwsrc)MPI_fpgw2.o
00270
00271 GWLIB = \
00272 $(gwsrc)m_w0w0i.o \
00273 $(gwsrc)getwemax.o \
00274 $(gwsrc)genallcf_dump.o \
00275 $(gwsrc)wse.o \
00276 $(gwsrc)bzints2.o \
00277 $(gwsrc)wintzsg.o \
00278 $(gwsrc)gintxx.o \
00279 $(gwsrc)gwinput_v2.o \
00280 $(gwsrc)pplmat.o \
00281 $(gwsrc)rs.o \
00282 $(gwsrc)conv2gwinput.o \
00283 $(gwsrc)getbzdata1.o \
00284 $(gwsrc)getgv2.o \
00285 $(gwsrc)wcf.o \
00286 $(gwsrc)tetwt5$(tet5_g) \
00287 $(gwsrc)m_tetwt.o \
00288 $(gwsrc)x0kf_v4h$(x0kf_g) \
00289 $(gwsrc)cinvrx.o \
00290 $(gwsrc)zsvd.o \
00291 $(gwsrc)m_zmel.o \
00292 $(gwsrc)m_freq.o \
00293 $(gwsrc)m_hamindex.o \
00294 $(gwsrc)readpomat.o \
00295 $(gwsrc)keyvalue.o \
00296 $(gwsrc)rppovl.o \
00297 $(gwsrc)nocctotg.o \
00298 $(gwsrc)ppbafp.fal$(para_g) \
00299 $(gwsrc)psi2b_v2$(para_g) \
00300 $(gwsrc)psi2b_v3$(para_g) \
00301 $(gwsrc)wfacx.o \
00302 $(gwsrc)sortea.o \
00303 $(gwsrc)rydberg.o \
00304 $(gwsrc)polinta.o \
00305 $(gwsrc)efsimplef.o \
00306 $(gwsrc)extension.o \
00307 $(gwsrc)rangedq.o \
00308 $(gwsrc)nword.o \
00309 $(gwsrc)scg.o \
00310 $(gwsrc)matm.o \
00311 $(gwsrc)rdpp.o \
00312 $(gwsrc)mptauof.o \

```

```

00313 $(gwsrc)genallcf_mod.o \
00314 $(gwsrc)rgwinf_mod.o \
00315 $(gwsrc)rotdlmm.o \
00316 $(gwsrc)iopen.o \
00317 $(gwsrc)cputid.o \
00318 $(gwsrc)rw.o \
00319 $(gwsrc)ext.o \
00320 $(gwsrc)ext2.o \
00321 $(gwsrc)cross.o \
00322 $(gwsrc)mate.o \
00323 $(gwsrc)matel.o \
00324 $(gwsrc)icopy.o \
00325 $(gwsrc)bibl.o \
00326 $(gwsrc)index.o \
00327 $(gwsrc)idxk.o \
00328 $(gwsrc)maxnn.o \
00329 $(gwsrc)reindx.o \
00330 $(gwsrc)iprint.o \
00331 $(gwsrc)bz.o \
00332 $(gwsrc)bzmesh.o \
00333 $(gwsrc)genqbz.o \
00334 $(gwsrc)switches.o \
00335 $(gwsrc)rbwzdata.o \
00336 $(gwsrc)llnew.o \
00337 $(gwsrc)readeigen.o \
00338 $(gwsrc)readqg.o \
00339 $(gwsrc)iqindx.o \
00340 $(gwsrc)alloclist.o \
00341 $(gwsrc)m_pkm4crpa.o \
00342 $(gwsrc)m_anf.o \
00343 $(gwsrc)qpel.sc.o \
00344 $(gwsrc)icomppv2.o \
00345 $(gwsrc)iopenxx.o \
00346 $(gwsrc)qpel.o \
00347 $(gwsrc)mopen.o \
00348 $(gwsrc)checksymbion.o \
00349 $(gwsrc)mkqg.o \
00350 $(gwsrc)m_q0p.o \
00351 $(gwsrc)q0irre.o \
00352 $(gwsrc)basnfp.o \
00353 $(gwsrc)excore.o \
00354 $(gwsrc)mkjp.o \
00355 $(gwsrc)strxq.o \
00356 $(gwsrc)sxcf_fal2.sc$(sxcf_g) \
00357 $(gwsrc)sxcf_fal2$(sxcf_g) \
00358 $(gwsrc)amix.o \
00359 $(gwsrc)dsifa.o \
00360 $(gwsrc)dsisl.o \
00361 $(gwsrc)dsidi.o \
00362 $(gwsrc)diagcv2.o \
00363 $(gwsrc)wronkj.o \
00364 $(gwsrc)rxrx.o \
00365 $(gwsrc)hsmq.o \
00366 $(gwsrc)u_lat_0.o \
00367 $(gwsrc)mklegw.o \
00368 $(gwsrc)bessl.o \
00369 $(gwsrc)lgen.o \
00370 $(gwsrc)hansr5.o \
00371 $(gwsrc)hansr4.o \
00372 $(gwsrc)lattc.o \
00373 $(gwsrc)qdist.o \
00374 $(gwsrc)dltor.o \
00375 $(gwsrc)dpcopy.o \
00376 $(gwsrc)dpadd.o \
00377 $(gwsrc)dpzero.o \
00378 $(gwsrc)ropyln.o \
00379 $(gwsrc)ropcsm.o \
00380 $(gwsrc)rpaq.o \
00381 $(gwsrc)m_readeps.o
00382
00383 QPE_QSGW = \
00384 $(main)hqpe_qsgw.m.o\
00385 $(gwsrc)qpel.qsgw.o
00386
00387 QPE_SC = \
00388 $(main)hqpe.sc.m$(hqpe_g)
00389
00390 QPE = \
00391 $(main)hqpe.m$(hqpe_g)
00392
00393 MERGE = \
00394 $(main)hmergewv.m.o
00395
00396 # PARAINFO = \
00397 # $(main)hparainfo.m.o \
00398 # $(gwsrc)charext.o
00399

```

```

00400 # BNDOUT = \
00401 # $(main)hbndout.m.o \
00402 # $(gwsrc)iqagree.o \
00403 # $(gwsrc)iopenxx.o \
00404 # $(gwsrc)iopen.o \
00405 # $(gwsrc)polinta.o \
00406 # $(gwsrc)rydberg.o \
00407 # $(gwsrc)extension.o \
00408 # $(gwsrc)rangedq.o \
00409 # $(gwsrc)switches.o \
00410 # $(gwsrc)keyvalue.o
00411 #
00412 # $(gwsrc)setpr.o \
00413 # $(gwsrc)sylm.o \
00414 # $(gwsrc)sylmc.o \
00415 # SIGMCONV = \
00416 # $(gwsrc)switches.o \
00417 # $(gwsrc)keyvalue.o \
00418 # $(gwsrc)iopen.o \
00419 # $(main)hsigmconv.m.o
00420
00421 #####
00422
00423 # bndconn.o: $(BNDCONN)
00424 #
00425 ##### dependency for use #####
00426
00427
00428
00429 # hsigmconv: $(SIGMCONV) $(MPI) $(COMM)
00430 # $(LK) $(LKFLAGS1) $(SIGMCONV) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00431
00432
00433 gwinit: $(GWINIT) $(MPI) $(GWLIB)
00434 $(LK) $(LKFLAGS1) $(GWINIT) $(MPI) $(GWLIB) $(LKFLAGS2) -o $@
00435
00436
00437 # qpwf: $(maxloc)qpwf.o $(GWLIB) $(MPI) $(COMM)
00438 # $(LK) $(LKFLAGS1) $(maxloc)qpwf.o $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00439
00440 qg4gw: $(QG) $(MPI) $(GWLIB) $(COMM)
00441 $(LK) $(LKFLAGS1) $(QG) $(MPI) $(GWLIB) $(COMM) $(LKFLAGS2) -o $@
00442
00443 rdata4gw_v2: $(RDAT_v2) $(MPI) $(COMM) $(GWLIB)
00444 $(LK) $(LKFLAGS1) $(RDAT_v2) $(MPI) $(COMM) $(GWLIB) $(LKFLAGS2) -o $@
00445
00446 hbasfp0: $(BAS) $(MPI) $(COMM) $(GWLIB)
00447 $(LK) $(LKFLAGS1) $(BAS) $(MPI) $(COMM) $(GWLIB) $(LKFLAGS2) -o $@
00448
00449 hvccfp0: $(MPI) $(VCC) $(DERFC) $(MPI) $(COMM) $(GWLIB)
00450 $(LK) $(LKFLAGS1) $(VCC) $(DERFC) $(MPI) $(COMM) $(GWLIB) $(LKFLAGS2) -o $@
00451
00452 hx0fp0: $(MPI) $(X0) $(GWLIB) $(MPI) $(COMM)
00453 $(LK) $(LKFLAGS1) $(X0) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00454
00455 # # for maxloc
00456 # hx0fp0_mlw: $(X0mlw) $(GWLIB) $(MPI) $(COMM)
00457 # $(LK) $(LKFLAGS1) $(X0mlw) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00458
00459 # h_uumatrix: $(UU) $(GWLIB) $(MPI) $(COMM)
00460 # $(LK) $(LKFLAGS1) $(UU) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00461
00462 # huumat: $(UU2) $(GWLIB) $(MPI) $(COMM)
00463 # $(LK) $(LKFLAGS1) $(UU2) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00464
00465 # hphig: $(PHIG) $(GWLIB) $(MPI) $(COMM)
00466 # $(LK) $(LKFLAGS1) $(PHIG) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) $(LIBSLA) -o $@
00467
00468 # hpsig: $(PSIG) $(GWLIB) $(MPI) $(COMM)
00469 # $(LK) $(LKFLAGS1) $(PSIG) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00470
00471 hx0fp0_sc: $(MPI) $(X0_SC) $(GWLIB) $(MPI) $(COMM)
00472 $(LK) $(LKFLAGS1) $(X0_SC) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00473
00474 # hwmatt: $(WMAT) $(GWLIB) $(MPI) $(COMM)
00475 # $(LK) $(LKFLAGS1) $(WMAT) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00476
00477 # hmaxloc: $(MLOC) $(GWLIB) $(MPI) $(COMM)
00478 # $(LK) $(LKFLAGS1) $(MLOC) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00479
00480 # hmaxloc1D: $(MLOC1D) $(GWLIB) $(MPI) $(COMM)
00481 # $(LK) $(LKFLAGS1) $(MLOC1D) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00482
00483 hsfp0: $(MPI) $(SXC) $(GWLIB) $(MPI) $(COMM)
00484 $(LK) $(LKFLAGS1) $(SXC) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00485
00486 hsfp0_sc: $(MPI) $(SXC_SC) $(GWLIB) $(MPI) $(COMM)

```

```

00487      $(LK) $(LKFLAGS1) $(SXC_SC)      $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00488
00489 # hnoacc_mlw:      $(hnoacc_mlw) $(GWLIB) $(MPI) $(COMM)
00490 #      $(LK) $(LKFLAGS1) $(hnoacc_mlw) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00491
00492 heftet:      $(heftet) $(GWLIB) $(MPI) $(MPI) $(COMM)
00493      $(LK) $(LKFLAGS1) $(heftet) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00494
00495 hef:      $(hef) $(GWLIB) $(MPI) $(COMM)
00496      $(LK) $(LKFLAGS1) $(hef) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00497
00498
00499 hqpe:      $(QPE) $(MPI) $(COMM) $(GWLIB)
00500      $(LK) $(LKFLAGS1) $(QPE) $(MPI) $(COMM) $(GWLIB) $(LKFLAGS2) -o $@
00501
00502 hqpe_sc:      $(QPE_SC) $(MPI) $(COMM) $(GWLIB)
00503      $(LK) $(LKFLAGS1) $(QPE_SC) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00504
00505 hqpe_qsgw:      $(QPE_QSGW) $(GWLIB) $(MPI) $(COMM)
00506      $(LK) $(LKFLAGS1) $(QPE_QSGW) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00507
00508 hmergewv:      $(MERGE) $(MPI) $(GWLIB) $(COMM)
00509      $(LK) $(LKFLAGS1) $(MERGE) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00510
00511 # hparainfo:      $(PARAINFO) $(GWLIB) $(MPI) $(COMM)
00512 #      $(LK) $(LKFLAGS1) $(PARAINFO) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00513
00514 # hbndout:      $(BNDOUT) $(MPI) $(COMM)
00515 #      $(LK) $(LKFLAGS1) $(BNDOUT) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00516
00517 convgwin:      $(convg)
00518      $(LK) $(LKFLAGS1) $(convg) $(LKFLAGS2) -o $@
00519
00520 kino_input_test:      $(kino_input_test) $(GWLIB) $(MPI) $(COMM)
00521      $(LK) $(LKFLAGS1) $(kino_input_test) $(GWLIB) $(MPI) $(COMM) $(LKFLAGS2) -o $@
00522
00523 ##### test
00524 #
00525 # test_genallcf:      $(test_genallcf)
00526 #      $(LK) $(LKFLAGS1) $(test_genallcf) $(LKFLAGS2) -o $@
00527
00528
00529 $(tags)TAGS: $(progs)
00530      cd $(tags);etags ./**/*.F ./**/*.F
00531
00532
00533 # --- Make rules ---
00534 .SUFFIXES:
00535 .SUFFIXES: .F .o
00536 #.SUFFIXES: .f .o .c1_o .c2_o .c3_o .c4_o .F
00537
00538 .F.o:
00539      $(FC) $(FFLAGS) $*.F -c -o $*.o
00540 #      etags $*.f -o $(tags)'echo $*.f | sed 's/..\\/' | sed 's/\\/-/g''.tags
00541
00542 #.F.o:
00543 #      $(FC) $(FFLAGS) $*.F -c -o $*.o
00544 #      etags $*.f -o $(tags)'echo $*.f | sed 's/..\\/' | sed 's/\\/-/g''.tags
00545
00546 #.f.o:
00547 #      $(FC) $(FFLAGS) $*.f -c -o $*.o
00548 #      etags $*.f -o $(tags)'echo $*.f | sed 's/..\\/' | sed 's/\\/-/g''.tags
00549
00550 .f.c1_o:
00551      $(FC) $(FFLAGS_c1) $*.f -c -o $*.c1_o
00552      etags $*.f -o $(tags)'echo $*.f | sed 's/..\\/' | sed 's/\\/-/g''.tags
00553
00554 .f.c2_o:
00555      $(FC) $(FFLAGS_c2) $*.f -c -o $*.c2_o
00556      etags $*.f -o $(tags)'echo $*.f | sed 's/..\\/' | sed 's/\\/-/g''.tags
00557
00558 .f.c3_o:
00559      $(FC) $(FFLAGS_c3) $*.f -c -o $*.c3_o
00560      etags $*.f -o $(tags)'echo $*.f | sed 's/..\\/' | sed 's/\\/-/g''.tags
00561
00562 .f.c4_o:
00563      $(FC) $(FFLAGS_c4) $*.f -c -o $*.c4_o
00564      etags $*.f -o $(tags)'echo $*.f | sed 's/..\\/' | sed 's/\\/-/g''.tags
00565
00566
00567 check:
00568      (cd ../TESTinstallGW;./testgw.py --enforce --all)
00569
00570 # test for f90 dependency
00571 #../main/hvccfp0.m.o      :      ../main/hx0fp0.m.o
00572 #
00573 #../main/hvccfp0.m.o      :      ../main/hbasfp0.m.o

```

```

00574
00575 include moduleddepends.inc
00576
00577
00578 #####
00579 ##### You can comment out these blocks to commnet out memory and time check (verbose output)
00580 addtime=script/addtime.awk
00581 septhen=script/then_separate.awk
00582 alloclist=script/add_alloclist.awk
00583 $(main)hsfp0.sc.m.o: $(main)hsfp0.sc.m.F
00584     gawk -f $(addtime) -vSTART=1 $(main)hsfp0.sc.m.F | gawk -f $(septhen) | gawk -f $(alloclist) >
00585     $(main)time_hsf0.sc.m.F
00586     $(FC) $(FFLAGS) $(main)time_hsf0.sc.m.F -c -o $*.o
00587 $(main)hx0fp0.sc.m.o: $(main)hx0fp0.sc.m.F
00588     gawk -f $(addtime) -vSTART=1 $(main)hx0fp0.sc.m.F | gawk -f $(septhen) | gawk -f $(alloclist) >
00589     $(main)time_hx0fp0.sc.m.F
00590     $(FC) $(FFLAGS) $(main)time_hx0fp0.sc.m.F -c -o $*.o
00591 $(gwsrc)sxcf_fal2.sc$(sxcf_g): $(gwsrc)sxcf_fal2.sc.F
00592     gawk -f $(addtime) -vSTART=100 $(gwsrc)sxcf_fal2.sc.F | gawk -f $(septhen) | gawk -f $(alloclist) >
00593     $(gwsrc)time_sxcf_fal2.sc.F
00594     $(FC) $(FFLAGS) $(gwsrc)time_sxcf_fal2.sc.F -c -o $*.o
00595 #$(gwsrc)rppovl.o: $(gwsrc)rppovl.F
00596 #     gawk -f $(addtime) -vSTART=200 $(gwsrc)rppovl.F | gawk -f $(septhen) | gawk -f $(alloclist) >
00597 #     $(gwsrc)time_rppovl.F
00598 #     $(FC) $(FFLAGS) $(gwsrc)time_rppovl.F -c -o $*.o
00599 $(gwsrc)x0kf_v4h$(x0kf_g): $(gwsrc)x0kf_v4h.F
00600     gawk -f $(addtime) -vSTART=100 $(gwsrc)x0kf_v4h.F | gawk -f $(septhen) | gawk -f $(alloclist) >
00601     $(gwsrc)time_x0kf_v4h.F
00602     $(FC) $(FFLAGS) $(gwsrc)time_x0kf_v4h.F -c -o $*.o
00603 $(gwsrc)ppbafp.fal$(para_g): $(gwsrc)ppbafp.fal.F
00604     gawk -f $(addtime) -vSTART=300 $(gwsrc)ppbafp.fal.F | gawk -f $(septhen) | gawk -f $(alloclist) >
00605     $(gwsrc)time_ppbafp.fal.F
00606     $(FC) $(FFLAGS) $(gwsrc)time_ppbafp.fal.F -c -o $*.o
00606 #$(gwsrc)ppbafp.fal$(para_g): $(gwsrc)ppbafp.fal.F
00607 #     gawk -f $(addtime) -vSTART=300 $(gwsrc)ppbafp.fal.F | gawk -f $(septhen) | gawk -f $(alloclist) >
00608 #     $(gwsrc)time_ppbafp.fal.F
00609 #     $(FC) $(FFLAGS) $(gwsrc)time_ppbafp.fal.F -c -o $*.o
00610 #####
00611
00612 $(gwsrc)wintzsg.o : $(gwsrc)wintzsg.F
00613     $(FC) $(FFLAGS) $(gwsrc)wintzsg.F -c -o $*.o
00614
00615
00616 # DO NOT DELETE

```

## 4.3 gwsrc/genallcf\_mod.F File Reference

### Data Types

- module [m\\_readfermi](#)
- module [m\\_genallcf\\_v3](#)

*get basic settings of crystal structure and nlm info*

### Functions/Subroutines

- subroutine [idxlnmc](#) (nindxv, nindxc, nl, nn, nnv, nnc, nlnmx, nlnmxv, nlnmxc, nclass, il, in, im, ilnm, ilv, inv, imv, ilnmv, ilc, inc, imc, ilnmc)
- integer function [noflmt0](#) (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, nl, nnc, nclass, natom, icore, ncore, nctot)

### 4.3.1 Function/Subroutine Documentation

4.3.1.1 subroutine `idxlmc` ( `dimension(0:nl-1,nclass) nindxv`, `dimension(0:nl-1,nclass) nindxc`, `nl`, `nn`, `nnv`, `nnc`, `nlnmx`, `nlnmzv`, `nlnmxc`, `nclass`, `dimension(nlnmx,nclass) il`, `dimension(nlnmx,nclass) in`, `dimension(nlnmx,nclass) im`, `dimension(nn,nl*nl,nclass) ilnm`, `dimension(nlnmxv,nclass) ilv`, `dimension(nlnmxv,nclass) inv`, `dimension(nlnmxv,nclass) imv`, `dimension(nnv,nl*nl,nclass) ilnmv`, `dimension(nlnmxc,nclass) ilc`, `dimension(nlnmxc,nclass) inc`, `dimension(nlnmxc,nclass) imc`, `dimension(nnc,nl*nl,nclass) ilnmc` )

Definition at line 392 of file [genallcf\\_mod.F](#).

Here is the caller graph for this function:

4.3.1.2 subroutine `incor` ( `dimension(0:nl-1,nnc,nclass) ncwf`, `dimension(0:nl-1,nclass) nindxc`, `dimension(natom) iclass`, `nl`, `nnc`, `nclass`, `natom`, `dimension(nl*nl*nnc,nclass) icore`, `dimension(nclass) ncore`, `nctot` )

Definition at line 568 of file [genallcf\\_mod.F](#).

Here is the caller graph for this function:

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

Definition at line 526 of file [genallcf\\_mod.F](#).

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

Definition at line 475 of file [genallcf\\_mod.F](#).

4.3.1.5 integer function `noflmt0` ( `dimension(0:nl-1,nclass) nindx`, `dimension(natom) iclass`, `nl`, `nclass`, `natom` )

Definition at line 462 of file [genallcf\\_mod.F](#).

Here is the caller graph for this function:

4.3.1.6 integer function `nofln` ( `dimension(0:nl-1) nindx`, `nl` )

Definition at line 504 of file [genallcf\\_mod.F](#).

4.3.1.7 integer function `noflnm` ( `dimension(0:nl-1) nindx`, `nl` )

Definition at line 515 of file [genallcf\\_mod.F](#).

## 4.4 `genallcf_mod.F`

```
00001 module m_readfermi
00002 real(8),protected:: bandgap
```

```

00003      real(8)::ef
00004      contains
00005
00006      subroutine readefermi()
00007      implicit none
00008      integer:: ifief,ifile_handle
00009      ifief=ifile_handle()
00010      open(ifief,file='EFERMI')
00011      read(ifief,*) ef,bandgap
00012      close(ifief)
00013      write(6,"(a,f12.6)")' --- READIN ef from EFERMI. ef=',ef
00014      end subroutine
00015      end module m_readefermi
00016
00017 !> get basic settings of crystal structure and nlm info
00018 !! - genallcf_v3(nwin,efin,incwfx) set data
00019 !! - This is old routine. Confusing. We need to clean up.
00020 module m_genallcf_v3
00021 !!-----
00022 !! - structure
00023 !! - o          plat,alat,natom,nclass,pos,
00024 !! - o          ngrp, symgg,
00025 !! - o          invg, ef,
00026 !! - l,n and dimensions
00027 !! - o          clabl, nspin,nl,nn,nnv,nn,
00028 !! - o          nindx, nindxc, nindxc, iclass,
00029 !! - d          nlmtol,nlnx,nlnxv,nlnxc,nlnmx,nlnmxv,nlnmxc,
00030 !! - o          z,
00031 !! - l,n,m indices for Phi (atomic basis)
00032 !! - o          il, in, im, ilnm, nlnm,
00033 !! - o          ilv,inv,imv, ilnmv, nlnmv,
00034 !! - o          ilc,inc,imc, ilnmc, nlnmc,
00035 !! - core
00036 !! - o          ncwf, ecore, konf, icore, ncore,nctot,
00037 !! - frequency
00038 !! -          niw,diw,nw,dw,delta,deltaw,esmr, freq)
00039 !!          symgrp
00040 !!          ,nocc, nunocc, occv, unoccv, occc, unoccc
00041      implicit none
00042      character(120),protected:: symgrp
00043      character(6),allocatable,protected:: clabl(:)
00044      integer,allocatable,protected:: iclass(:),
00045      & nindxv(:,:),nindxc(:,:),ncwf(:,:),
00046      o invg(:), il(:,:), in(:,:), im(:,:), ilnm(:), nlnm(:),
00047      o ilv(:),inv(:),imv(:), ilnmv(:), nlnmv(:),
00048      o ilc(:),inc(:),imc(:), ilnmc(:), nlnmc(:),
00049      o nindx(:,:),konf(:,:),icore(:,:),ncore(:),
00050      & occv(:,:),unoccv(:,:),
00051      & ,occc(:,:),unoccc(:,:),
00052      o nocc(:,:),nunocc(:,:), iantiferro(:)
00053      integer,protected::
00054      o nclass,natom,nspin,nl,nn,nnv,nn, ngrp,
00055      o nlmtol,nlnx,nlnxv,nlnxc,nlnmx,nlnmxv,nlnmxc, nctot!,nw
00056      real(8), allocatable,protected::
00057      o plat(:,:),pos(:,:),z(:), symgg(:,:), !w(igrp) freq(:),
00058      real(8),protected:: alat,deltaw !ef,,diw,dw
00059      logical,protected:: done_genallcf_v3=.false.
00060      character(8),allocatable,protected:: spid(:)
00061 !! unprotected --> need to be protected
00062      real(8), allocatable:: ecore(:,:)
00063      real(8):: delta
00064      integer:: niw
00065      real(8):: esmr
00066 c-----
00067      contains
00068
00069      subroutine genallcf_v3(incwfx)
00070      !!> Readin GWIN_V2 and LMTO(crystal) data and allocate all required.
00071      !!r Return iclass=ibas.
00072      !! efin,incwfx, are used as switches.
00073      !! input: efin,incwfx,
00074      !!          GWIN_V2, LMTO
00075      !! output: All the output are given in the declare section above.
00076      !! -----
00077      implicit none
00078      integer(4)::iflmtol,ifinin,incwfx,ifec,i,j,
00079      & lmx, lmx2,nlmtol2,nprodxc,nlnaxc,nlnaxv,nprodxc,ifi,ig,is
00080      & ,iopen,iclose,nprodxc,nlnax
00081      & ,noflmtol,maxnn
00082      integer(4):: infwfx
00083      integer(4):: nl,n2,n3,imagw,lcutmx,n,ic
00084      logical:: nocore
00085      real(8)::efin
00086      real(8),allocatable::tolbas(:)
00087      character(120):: symgrpt
00088      real(8), allocatable:: ecoret(:,:,:)
00089      integer(4),allocatable::ncwf2(:,:,:)

```

```

00090      integer:: ia,l,m,icl,isp,lt,nt,nsp,nr,ncorex,ifix
00091      real(8)::a,b,zz,  efdummy,dw,diw
00092      integer:: nwdummy
00093 c      allocate(nclass,natom,nsp,nl,nnv,nnnc, ngrp,
00094 c      o  nlmto,nlnx,nlnxv,nlnxc,nlnmx,nlnmxv,nlnmxc, nctot, niw,nw)
00095      if(done_genallcf_v3) call rx('genallcf_v3 is already called')
00096      done_genallcf_v3=.true.
00097
00098 c      allocate(alat,ef, diw,dw,delta,deltaw,esmr,  symgrp)
00099      iflmto  = iopen('LMT0',1,0,0)
00100      if (iflmto < 0) call rx( 'unit file for GWIN_V2 < 0')
00101
00102 c--- readin these by rgwinf_v3
00103 c      character*120 symgrp
00104 c      integer(4)::nclass,natom,nsp,nl,nnv,nnnc
00105 c      real(8)::alat
00106 c      integer(4),allocatable::
00107 c      &  iclass(:)
00108 c      &  ,nindxv(:,:),nindxc(:,:)
00109 c      &  ,occv(:,:),unoccv(:,:),)
00110 c      &  ,occc(:,:),unoccc(:,:),)
00111 c      &  ,ncwf(:,:),)
00112 c      real(8),allocatable:: plat(:,:),pos(:,:),z(:)
00113 c      character*6,allocatable:: clabl(:)
00114 c      write(6,*)' goto rgwinf'
00115 c      call rgwinf_v3 (iflmto,ifinin,nwin,efin,incwfx) !these are inputs
00116 c      write(6,*)' end of rgwinf_v3'
00117 c-----
00118 c--- rgwinf ---
00119      ifi = iflmto
00120 c      nw  = nwin
00121 c      ef  = efin
00122      read(ifi,*) ; read(ifi,*)
00123      read(ifi,*)symgrpt      !SYMMETRY
00124      j = 0
00125      symgrp=' '//trim(adjustl(symgrpt))
00126      write(6,*)' symgrp=', symgrp
00127      read(ifi,*)
00128      read(ifi,*)
00129      read(ifi,*)
00130      read(ifi,*)alat      !lattice constant
00131      allocate(plat(3,3))      !primitive lattice vectors
00132      read(ifi,*)
00133      read(ifi,*)plat(1:3,1)
00134      read(ifi,*)plat(1:3,2)
00135      read(ifi,*)plat(1:3,3)
00136      read(ifi,*)
00137      read(ifi,*) natom      !Number of atoms
00138 !!
00139      nclass = natom !We set nclass = natom through the GW calculations
00140      write(6,*)'genalloc: alat natom=',alat,natom
00141      allocate(pos(3,natom))      !positions of atoms
00142      read(ifi,*)
00143      do n = 1,natom
00144          read(ifi,*) pos(1,n),pos(2,n),pos(3,n)
00145      end do
00146      read(ifi,*)
00147      read(ifi,*)
00148      read(ifi,*)
00149      read(ifi,*)nspn      !spin (1=paramagnetic 2=ferromagnetic)
00150      read(ifi,*)
00151      read(ifi,*)nl      !max. no. valence and core l
00152      read(ifi,*)
00153      read(ifi,*)nnv,nnnc !max. no. valence and core n
00154      write(6,*)' nspn nl nnv nnc =',nspn,nl,nnv,nnnc
00155 c-----
00156      if(nnv==1) nnv=2 ! for backward compatibility!takao apr 2002
00157      ! nnv=2 corresponds to phi and phidot
00158      ! nnv=3 corresponds to
00159 c-----
00160      read(ifi,*)
00161      read(ifi,*) !nrx is not readin
00162      read(ifi,*)
00163      allocate(clabl(nclass),z(nclass)) !class-label, z
00164      do ic = 1,nclass
00165          read(ifi,*) clabl(ic),z(ic) !,nrofi is not readin
00166      end do
00167
00168      allocate(iclass(natom)) !atom and its class.
00169      do n = 1,natom      !!We set nclass = natom through the GW calculations
00170          iclass(n)=n
00171      end do
00172
00173      allocate(nindxv(nl,nclass), nindxc(nl,nclass),
00174      &  occv(nl,nnv,nclass),unoccv(nl,nnv,nclass),
00175      &  occc(nl,nnnc,nclass),unoccc(nl,nnnc,nclass))
00176      allocate(ncwf2(nl,nnnc,nclass),ncwf(nl,nnnc,nclass))

```



```

00177      allocate(tolbas(0:2*(nl-1)))
00178      ifix=ifi
00179      call rgwinaf(ifi,ifinin,nl,nnv,nn,nnc,nclass, !ifi can be changed.
00180 c> BZ
00181      o          nl,n2,n3,efdumy,
00182 c> frequencies
00183      o          niw,diw,nwdumy,dw,delta,deltaw,esmr,imagw,
00184 c> coulomb
00185      c      o          tolvc,alp,alptx,h,ng,
00186 c> product basis
00187      o          tolbas,lcutmx,nindxv,nindxc,
00188      o          occv,unoccv, occc,unoccc,
00189 c> core
00190      o          ncwf,ncwf2 )
00191 c-----
00192      allocate(iantiferro(1:natom),spid(1:natom))
00193      read(ifix,*)
00194      read(ifix,*)iantiferro(1:natom) !may2015
00195      read(ifix,*)
00196      read(ifix,*)spid(1:natom)
00197
00198      inquire(file='NoCore',exist=nocore)
00199      if(nocore) then
00200          occc=0      ! call iclear(nl*nn*nnc*nclass, w(ioccc))
00201          unoccc=0    ! call iclear(nl*nn*nnc*nclass, w(iunoccc))
00202          ncwf =0     ! call iclear(nl*nn*nnc*nclass, w(ncwf))
00203      elseif( incwfx== -1 ) then
00204          write(6,*)' ### incwf=-1 Use ForSxc for core'
00205          ncwf = ncwf2 !call icopy(nl*nn*nnc*nclass,w(ncwf2),w(ncwf))
00206      elseif( incwfx== -2 ) then
00207          write(6,*)' ### incwf=-2 Use NOT(ForSxc) for core and Pro-basis '
00208          call notbit(nl*nn*nnc*nclass, ncwf2)
00209          ncwf = ncwf2 ! call icopy (nl*nn*nnc*nclass, w(ncwf2),w(ncwf))
00210          occc= ncwf ! call icopy (nl*nn*nnc*nclass, w(ncwf),w(ioccc))
00211          unoccc= 0   ! call iclear(nl*nn*nnc*nclass, w(iunoccc))
00212      elseif( incwfx== -3 ) then
00213          call ibiton(nclass,nl,nn,nindxc, occc, ncwf)
00214          unoccc= 0    ! call iclear(nl*nn*nnc*nclass, w(iunoccc))
00215          write(6,*)' ### incwf=-3  occ=1 unocc=0 incwf=1 for all core '
00216      elseif( incwfx== -4 ) then
00217          write(6,*)' ### incwf=-4  occ=0 and unocc=0 for all core '
00218          occc=0 !call iclear(nl*nn*nnc*nclass, w(ioccc))
00219          unoccc=0 !call iclear(nl*nn*nnc*nclass, w(iunoccc))
00220          ncwf=0 !call iclear(nl*nn*nnc*nclass, w(ncwf))
00221      elseif(incwfx=0) then
00222          write(6,*)' ### Use unocc occ ForX0 for core'
00223      else
00224          call rx( ' ### proper incwf is not given for genallcf2:rgwinf ')
00225      endif
00226      deallocate(ncwf2)
00227 c... End of rgwinf section -----
00228
00229
00230 c> dimensions and constants
00231      lmx      = 2*(nl-1)
00232      lmx2     = (lmx+1)**2
00233      nlmt0    = noflmt0(nindxv,iclass,nl,nclass,natom)
00234      nlmt02   = nlmt0*nlmt0
00235      nn       = maxnn(nindxv,nindxc,nl,nclass)
00236
00237 c>> combine nocc,nunocc,nindx
00238      allocate(nindx(nl,nclass))
00239      allocate(nocc(nl,nn,nclass),nunocc(nl,nn,nclass))
00240      call reindx(occv,unoccv,nindxv,
00241      i          occc,unoccc,nindxc,
00242      d          nl,nn,nnv,nn,nnc,nclass,
00243      o          nocc,nunocc,nindx)
00244      call maxdim(occv,unoccc,nindxc,nl,nn,nclass,
00245      o          nprodx,nlnxc,nlnmx,nlnax)
00246      call maxdim(occv,unoccv,nindxv,nl,nnv,nclass,
00247      o          nprodxv,nlnxv,nlnmxv,nlnaxv)
00248      call maxdim(nocc,nunocc,nindx,nl,nn,nclass,
00249      o          nprodx,nlnx,nlnmx,nlnax)
00250
00251 c      nlnx4    = nlnx**4
00252 c      nphi     = nrx*nl*nn*nclass
00253 c      pi       = 4d0*datan(1d0)
00254 c      tpia     = 2d0*pi/alat
00255
00256 c$$$c> frequency mesh
00257 c$$$c      call defdr (ifreq,nw)
00258 c$$$      write(6,*)' nw from rgwinaf=',nw
00259 c$$$      if(nw>0) then
00260 c$$$          allocate(freq(nw))
00261 c$$$          call genfreq (nw,dw,0.d0,
00262 c$$$      o          freq )
00263 c$$$      endif

```

```

00264
00265 c> index for allowed core states
00266 c      call defi      (iicore,nl*nl*nnc*nclass)
00267 c      call defi      (incore,nclass)
00268      allocate(iicore(nl**2*nnc,nclass),ncore(nclass))
00269      icore=9999999
00270      ncore=9999999
00271      call incor(ncwf,nindxc,iclass,
00272              d      nl,nnc,nclass,natom,
00273              o      icore,ncore,ncotot )
00274 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00275 c      write(6,*)' nnc=',nnc,nl,nclass,natom
00276 c      write(6,*)' ncwf ',ncwf
00277 c      write(6,*)' nindxc ',nindxc
00278 c      write(6,*)' iclass ',iclass
00279 c      write(6,*)' --- icore=',icore
00280 c      write(6,*)' --- ncore nctot=',ncore,ncotot
00281 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00282
00283 c> core energies
00284      ifec      = iopen('ECORE',1,0,0)
00285      allocate(konf(nl,nclass),ecore(ncotot,2))
00286      konf=0
00287      allocate(ecoret(0:nl-1,nnc,2,nclass))
00288      ecoret=0d0
00289      do ic = 1,nclass
00290          write(6,*) ' read Ecore : ic=',ic
00291          read (ifec,*)
00292          read (ifec,*)
00293          read (ifec,*)
00294          read (ifec,*) !zz,icl,nr ,a,b,ns
00295          read (ifec,*)
00296          read (ifec,*) (konf(l+1,ic),l=0,nl-1)
00297          read (ifec,*)
00298          do l = 0,nl-1
00299              ncorex = konf(l+1,ic)-l-1
00300              if (ncorex .gt. nnc) call rx( 'ECORE: wrong nnc')
00301              do n = 1,ncorex
00302                  read (ifec,*) lt,nt,(ecoret(l,n,isp,ic),isp=1,ns
00303                      if (ns
00304 c      write(6,*)(' read ecore=',3i4,2d13.5)")l,n,ic,ecoret(l,n,1:ns
00305                      if (lt .ne. 1) call rx( 'rcore: wrong l')
00306                      if (nt .ne. n) call rx( 'rcore: wrong n')
00307                  end do
00308              end do
00309          end do
00310          i = 0
00311          do ia = 1,nclass
00312              ic = iclass(ia)
00313              do l = 0,nl-1
00314                  do n = 1,nnc
00315                      do m = -1,1
00316                          if (ncwf(l+1,n,ic) .eq. 1) then
00317                              i = i + 1
00318                              if (i > nctot) call rx( 'genalloc_mod: wrong nctot')
00319                              ecore(i,l:ns
00320                                  write(6,*)(' ecore=',4i4,2d13.5)")i, l,n,ic,ecore(i,l:ns
00321                              endif
00322                          enddo
00323                      enddo
00324                  enddo
00325              enddo
00326          deallocate(ecoret)
00327 c> index for core and LMT0 basis
00328 c      call defi      (iil,nlnmx*nclass)
00329 c      call defi      (iin,nlnmx*nclass)
00330 c      call defi      (iim,nlnmx*nclass)
00331 c      call defi      (iilm,nn*nl*nl*nclass)
00332 c      call defi      (iilv,nlnmxv*nclass)
00333 c      call defi      (iinv,nlnmxv*nclass)
00334 c      call defi      (iimv,nlnmxv*nclass)
00335 c      call defi      (iilmv,nnv*nl*nl*nclass)
00336 c      call defi      (iilc,nlnmxc*nclass)
00337 c      call defi      (iinc,nlnmxc*nclass)
00338 c      call defi      (iimc,nlnmxc*nclass)
00339 c      call defi      (iilmc,nnc*nl*nl*nclass)
00340      allocate(
00341      & iil(nlnmx,nclass),
00342      & iin(nlnmx,nclass),
00343      & iim(nlnmx,nclass),
00344      & iilm(nn*nl*nl*nclass),
00345      & iilv(nlnmxv*nclass),
00346      & iinv(nlnmxv*nclass),
00347      & iimv(nlnmxv*nclass),
00348      & iilmv(nnv*nl*nl*nclass),
00349      & iilc(nlnmxc*nclass),
00350      & inc(nlnmxc*nclass),

```

```

00351      & imc(nlnmxc*nclass),
00352      & ilnmc(nnc*nl*nl*nclass)
00353      & )
00354      call idxlnmc( nindxv,nindxc,
00355      d            nl,nn,nnv,nnc,nlnmx,nlnmxv,nlnmxc,nclass,
00356      o            il,in,im,ilnm,
00357      o            ilv,inv,imv,ilnmv,
00358      o            ilc,inc,imc,ilnmc)
00359      allocate(nlnmv(nclass),nlnmc(nclass),nlnm(nclass))
00360      call nlnma(nindxv,nl,nclass,
00361      o          nlnmv )
00362      call nlnma(nindxc,nl,nclass,
00363      o          nlnmc )
00364      call nlnma(nindx,nl,nclass,
00365      o          nlnm )
00366      i=2 !see previous definition of symgrp
00367      if(symgrp(i+1:i+13)/= 'UseSYMOPSfile') then
00368        call rx( " Not: UseSYMOPSfile in LMT0 file")
00369      endif
00370      write(6,*) ' symgrp==UseSYMOPSfile'
00371      ifi = 6661
00372      open (ifi, file='SYMOPS')
00373      read(ifi,*) ngrp
00374      allocate(symgg(3,3,ngrp))
00375      do ig = 1,ngrp
00376        read(ifi,*)
00377        do i=1,3
00378          read(ifi,"(3d24.16)") symgg(i,1:3,ig)
00379        enddo
00380      enddo
00381      close(ifi)
00382      allocate(invgr(ngrp))
00383      call invgrp(symgg,ngrp,
00384      o          invg)
00385      is = iclose('LMT0')
00386      is = iclose('ECORE')
00387      call cputid(0)
00388      write(6,*) 'genallcf_v3'
00389      end subroutine genallcf_v3
00390      end module
00391
00392      subroutine idxlnmc(nindxv,nindxc,
00393      d            nl,nn,nnv,nnc,nlnmx,nlnmxv,nlnmxc,nclass,
00394      o            il,in,im,ilnm,
00395      o            ilv,inv,imv,ilnmv,
00396      o            ilc,inc,imc,ilnmc)
00397      c 92.jan.07
00398      c 92.03.17 include core states
00399      c indexing of core states and LMT0 basis functions for all classes,
00400      c follows that in TB-LMTO program
00401      c il,in,im = 1,n,m
00402      c ilnm(n,lm) = index of n,l,m
00403      c lm = l+1 + 1 + m + 1
00404      c NOTE: the indexing starts with core first and then valence on top
00405      c of core (not the same as index generated from nindx)
00406      implicit real*8(a-h,o-z)
00407      dimension nindxv(0:nl-1,nclass),nindxc(0:nl-1,nclass)
00408      dimension ilnm(nn,nl*nl,nclass),
00409      o          ilnmv(nnv,nl*nl,nclass),
00410      o          ilnmc(nnc,nl*nl,nclass),
00411      o          in(nlnmx,nclass),il(nlnmx,nclass),im(nlnmx,nclass),
00412      o          inv(nlnmxv,nclass),ilv(nlnmxv,nclass),imv(nlnmxv,nclass),
00413      o          inc(nlnmxc,nclass),ilc(nlnmxc,nclass),imc(nlnmxc,nclass)
00414      do ic = 1,nclass
00415        ind = 0
00416      c core
00417        do l = 0,nl-1
00418          lm = l+1
00419          do n = 1,nindxc(l,ic)
00420            m = 1,2*l+1
00421            ind = ind + 1
00422            if (ind .gt. nlnmx) call rx( 'idxlnmc: ind > nlnmx')
00423            lm = l2 + m
00424            il(ind,ic)= 1
00425            in(ind,ic)= n
00426            im(ind,ic)= m - 1 - 1
00427            ilnm(n,lm,ic) = ind
00428            ilc(ind,ic)= 1
00429            inc(ind,ic)= n
00430            imc(ind,ic)= m - 1 - 1
00431            ilnmc(n,lm,ic)= ind
00432          end do
00433        end do
00434      end do
00435      c valence
00436      indv = 0
00437      do l = 0,nl-1

```

```

00438         l2          = l+1
00439         ncore       = nindx(1,ic)
00440         do          n = 1,nindxv(1,ic)
00441             if (ncore+n .gt. nn) call rx( 'idxlnmc: ncore+n > nn')
00442             do      m = 1,2*1+1
00443                 ind  = ind + 1
00444                 indv = indv + 1
00445                 if (ind .gt. nlnmx) call rx( 'idxlnmc: ind > nlnmx')
00446                 lm   = l2 + m
00447                 il(ind,ic)= 1
00448                 in(ind,ic)= ncore + n
00449                 im(ind,ic)= m - 1 - 1
00450                 ilnm(ncore+n,lm,ic) = ind
00451                 ilv(indv,ic)= 1
00452                 inv(indv,ic)= n
00453                 imv(indv,ic)= m - 1 - 1
00454                 ilnmv(n,lm,ic) = indv
00455             end do
00456         end do
00457     end do
00458 end do
00459 return
00460 end
00461
00462 integer function noflmto(nindx,iclass,nl,nclass,natom)
00463 c total number of LMT0 basis functions
00464 implicit real*8(a-h,o-z)
00465 dimension nindx(0:nl-1,nclass),iclass(natom)
00466 noflmto = 0
00467 do 1 i = 1,natom
00468     ic = iclass(i)
00469     do 1 l = 0,nl-1
00470         noflmto = noflmto + (2*1+1)*nindx(1,ic)
00471     1 continue
00472     return
00473 end
00474
00475 integer function nalwln (nocc,nunocc,nindx,nl,nn)
00476 c gives the number of allowed product radial phi
00477 c nocc(1,n) = 0,1 ==> unoccupied, occupied
00478 c nunocc(1,n) = 1,0 ==> unoccupied,occupied
00479 c nalwln = number of allowed phi(l1,n1) phi(l2,n2)
00480 implicit real*8(a-h,o-z)
00481 parameter(lmax=6,nmx=10)
00482 dimension nocc(0:nl-1,nn),nunocc(0:nl-1,nn),
00483 i nindx(0:nl-1)
00484 dimension icheck(0:lmax,nmx,0:lmax,nmx)
00485 if (nl-1 .gt. lmax) call rx( 'nalwln: increase lmax')
00486 if (nn .gt. nmxx) call rx( 'nalwln: increase nmxx')
00487 icheck=0
00488 nalwln = 0
00489 do 10 l1 = 0,nl-1
00490     do 10 n1 = 1,nindx(l1)
00491         if(nocc(l1,n1) .eq. 0)goto 10
00492         do 20 l2 = 0,nl-1
00493             do 20 n2 = 1,nindx(l2)
00494                 if(nunocc(l2,n2) .eq. 0)goto 20
00495                 if((l1.ne.l2 .or. n1.ne.n2) .and. icheck(l2,n2,l1,n1).ne.0)
00496                     . goto 20
00497                 nalwln = nalwln + 1
00498                 icheck(l1,n1,l2,n2) = nalwln
00499             20 continue
00500         10 continue
00501     return
00502 end
00503
00504 integer function nofln(nindx,nl)
00505 c count the number of l,n
00506 implicit real*8(a-h,o-z)
00507 dimension nindx(0:nl-1)
00508 nofln = 0
00509 do 1 l = 0,nl-1
00510     nofln = nofln + nindx(1)
00511 end do
00512 return
00513 end
00514 c-----
00515 integer function noflnm(nindx,nl)
00516 c number of l,n,m
00517 implicit real*8(a-h,o-z)
00518 dimension nindx(0:nl-1)
00519 noflnm = 0
00520 do 1 l = 0,nl-1
00521     noflnm = noflnm + nindx(1)*(2*1+1)
00522 1 continue
00523 return
00524 end

```

```

00525
00526     integer function nallow (nocc,nunocc,nindx,nl,nn)
00527 c gives the number of allowed product basis
00528 c nocc(n,l) = 0,1 ==> unoccupied, occupied
00529 c nallow = number of allowed product basis
00530     implicit real*8(a-h,o-z)
00531     parameter(lmax=6,nnx=10)
00532     dimension nocc(0:nl-1,nn),nunocc(0:nl-1,nn),
00533             i             nindx(0:nl-1)
00534     dimension icheck(0:lmax,nnx,0:lmax,nnx)
00535     if(nl-1.gt. lmax) call rx( 'nallow: increase lmax')
00536     if(nn .gt. nnx) call rx( 'nallow: increase nnx')
00537     icheck=0
00538     do      l1 = 0,nl-1
00539     do      nl = 1,nindx(l1)
00540     do      l2 = 0,nl-1
00541     do      n2 = 1,nindx(l2)
00542         icheck(l1,nl,l2,n2) = nocc(l1,nl)*nunocc(l2,n2)
00543         if (l1 .ne. l2 .or. nl .ne. n2) then
00544             if (icheck(l1,nl,l2,n2)*icheck(l2,n2,l1,nl) .ne. 0)
00545                 . icheck(l1,nl,l2,n2) = 0
00546             endif
00547         end do
00548     end do
00549     end do
00550     end do
00551     nallow = 0
00552     do 10  l1 = 0,nl-1
00553     do 10  nl = 1,nindx(l1)
00554     do 10  ml = 1,2*l1+1
00555     do 10  l2 = 0,nl-1
00556     do 10  n2 = 1,nindx(l2)
00557     do 10  m2 = 1,2*l2+1
00558 c     if (nocc(l1,nl) .eq. 0)goto 10
00559 c     if (nunocc(l2,n2) .eq. 0)goto 10
00560         if (icheck(l1,nl,l2,n2) .eq. 0) goto 10
00561 c temporary
00562         if (l1 .eq. l2 .and. nl.eq.n2 .and. ml.lt.m2)goto 10
00563         nallow = nallow + 1
00564     10 continue
00565     return
00566     end
00567
00568     subroutine incor (ncwf,nindxc,iclass,
00569             d             nl,nnc,nclass,natom,
00570             o             icore,ncore,nctot)
00571 c 92.03.18
00572 c sorts out allowed core states and count the number of core states
00573 c ncwf(1,n,cl) = 1 ==> allowed, 0 ==> not allowed
00574 c nindxc(1,cl) = no. core states/l,class
00575 c nl,nnc = max. no. l,n
00576 c icore(i,cl) = index for allowed core states
00577 c ncore(cl) = no. allowed core states
00578 c nctot = total no. allowed core states
00579     implicit real*8 (a-h,o-z)
00580     dimension ncwf(0:nl-1,nnc,nclass),nindxc(0:nl-1,nclass),
00581             i             iclass(natom)
00582     dimension icore(nl*nl*nnc,nclass),ncore(nclass)
00583     ncx = nl*nl*nnc
00584     do      ic = 1,nclass
00585     i       = 0
00586     j       = 0
00587     do      l = 0,nl-1
00588     do      n = 1,nindxc(l,ic)
00589     do      m = -l,l
00590     j       = j + 1
00591     if (ncwf(l,n,ic) .eq. 1) then
00592         i       = i + 1
00593         if (i .gt. ncx) call rx( 'incore: wrong ncx')
00594         icore(i,ic)= j
00595     endif
00596     end do
00597     end do
00598     end do
00599     ncore(ic) = i
00600     end do
00601 c total no. allowed core states
00602     nctot = 0
00603     do      i = 1,natom
00604     ic      = iclass(i)
00605     nctot = nctot + ncore(ic)
00606     end do
00607     return
00608     end

```

## 4.5 gwsr/m\_anf.F File Reference

### Data Types

- module `m_anf`

*Antiferro condition module. We have line AFcond at the bottom of 'LMTO' file. Currently(feb2016), only laf is used (thus AF symmetry is not used yet for hx0fp0\_sc) To access laf, need to call `anfcond()` in advance.*

## 4.6 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,protected:: laf !! - laf: antiferro switch
00008     integer,allocatable,protected:: 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,*) 'Read AFcond section in LMTO file, call anfcond in m_anf.F:'
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         if(trim(aaa)=='number of atoms (natom)') then
00030             read(ilmto,*) natom
00031             read(ilmto,*)
00032             allocate(iantiferro(natom),ibasf(natom))
00033         endif
00034         if(aaa(1:6)=='AFcond') then
00035             read(ilmto,*) iantiferro(1:natom)
00036             ibasf=-999
00037             do ibas=1,natom
00038                 do ibasx=ibas+1,natom
00039                     if(abs(iantiferro(ibas))/=0 .and. iantiferro(ibas)+iantiferro(ibasx)==0) then
00040                         ibasf(ibas)=ibasx
00041                         exit
00042                     endif
00043                 enddo
00044                 if(ibasf(ibas)/=-999) write(6,"(a,2i5)") ' AF pair: ibas ibasf(ibas)=',ibas,ibasf(ibas)
00045             enddo
00046         endif
00047     enddo
00048 1011 continue
00049     close(ilmto)
00050     if(sum(abs(iantiferro))==0) then
00051         laf=.false. !no AF case
00052         return
00053     endif
00054     !! Antiferro case -----
00055     laf=.true.
00056     if(laf) write(6,"(a,100i4)") ' Antiferromode=',iantiferro
00057     end subroutine anfcond
00058     end module
00059
```

## 4.7 gwsr/m\_freq.F File Reference

## Data Types

- module `m_freq`

*Frequency mesh generator.*

## 4.8 m\_freq.F

```
00001 !>Frequency mesh generator
00002 !! - OUTPUT
00003 !! - fhris :histogram bins to accumulate 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,protected:: frhis(:),freq_r(:),freq_i(:),wiw(:)
00011   integer,protected:: nwhis,npm,nw_i,nw
00012
00013 c      real(8),allocatable,protected:: frhis0(:)
00014 c      integer,protected:: nwhis0
00015 contains
00016 !> Get data set for m_freq. All arguments are input.
00017 !! - This read GWinput(dw,omg_c) and TimeReversal()
00018 !! - All arguments are input
00019 subroutine getfreq(epsmode,realomega,imagomega,tetra,omg2max,wemax,niw,ua,mpi__root)
00020   use m_keyvalue,only:getkeyvalue
00021
00022   implicit none
00023   integer,intent(in):: niw !,nw_input
00024   logical,intent(in):: realomega,imagomega,tetra,mpi__root,epsmode
00025   real(8),intent(in):: omg2max,ua
00026
00027   real(8),allocatable:: freqx(:),wx(:),expa(:)
00028   logical:: timereversal,onceww
00029   integer:: nw2,iw,ihis
00030   real(8)::omg_c,dw,omg2,wemax
00031   real(8), allocatable :: freqr2(:) ,frhis_tmp(:)
00032   real(8):: pi = 4d0*datan(1d0), aa,bb,ratio,oratio,daa
00033   integer::nee,noo,ifif,ifile_handle
00034
00035   logical,save:: done=.false.
00036   if(done) call rx('gerfreq is already done') !sanity check
00037   done =.true.
00038   nw=-99999 !for sanity check
00039
00040 c      nw = nw_input
00041 !! Histogram bin divisions
00042 !! We first accumulate Imaginary parts.
00043 !! Then it is K-K transformed to obtain real part.
00044
00045 c      call getkeyvalue("GWinput","dw",dw )
00046 c      call getkeyvalue("GWinput","omg_c",omg_c )
00047 c      write(6,('(dw, omg_c= ',2f13.5)) dw, omg_c
00048   call getkeyvalue("GWinput","HistBin_ratio",oratio, default=1.03d0)
00049   call getkeyvalue("GWinput","HistBin_dw",dw, default=1d-5) !a.u.
00050   aa = oratio-1d0
00051   bb = dw/aa
00052   iw = 0d0
00053   do
00054     iw=iw+1
00055     if( bb*( exp(aa*(iw-1)) - 1d0 ) >omg2max+1d-6) exit
00056   enddo
00057   nwhis = iw+2 !+2 for margin. Necessary?
00058   allocate(frhis(1:nwhis+1))
00059   do iw = 1,nwhis+1
00060     frhis(iw) = bb*( exp(aa*(iw-1)) - 1d0 )
00061   enddo
00062   write(6,('(dw, omg_ratio, nwhis= ',d9.2,f13.5,i6)) dw, aa,nwhis
00063
00064 !! Determine nw. Is this correct?
00065   do iw=3,nwhis
00066     omg2 = (frhis(iw-2)+frhis(iw-1))/2d0
00067     if (omg2 > wemax/2d0 ) then !>dw*(nw_input-3)) then !omg is in unit of Hartree
00068       nw=iw
00069       exit
00070     endif
00071   enddo
00072 !! document need to be fixed...
00073 c      nw=nw2-1      ! nw+1 is how many points of real omega we use
00074                   ! for dressed coulomb line W(iw=0:nw) iw=0 corresponds omg=0
```

```

00075      ! maximum nw=nw2-1 because nwhis=nw2-1
00076 !! document need to be fixed...
00077      !nw is chosen from condition that frhis_m(nw-3)<dw*(nw_input-3) <frhis_m(nw-2).
00078      !Here frhis_m(iw)= (freqr2(iw)+freqr2(iw+1))/2d0
00079      !nw was constructed such that omg=dw*(nw-2)> all relevant frequensies needed
00080      ! for correlation Coulomb Wc(omg),
00081      ! and one more point omg=dw*(nw-1) needed for extrapolation.
00082      ! Now, frhis_m(nw-1)> all relevent frequensies for Wc(omg)
00083      ! and one more point omg=frhis_m(nw) needed for extrapolation
00084      ! used in subroutine alagr3z in  sxcf.f.
00085
00086 !! Determine freq_r
00087      if(epsmode) then
00088          nw = nwhis-1
00089      endif
00090      allocate(freq_r(0:nw))
00091      freq_r(0)=0d0
00092      do iw=1,nw
00093          freq_r(iw)=(frhis(iw)+frhis(iw+1))/2d0
00094      enddo
00095
00096 !! Timereversal=F is implimented only for tetra=T and sergeyv=T
00097 !! nw_i and npm
00098      npm=1
00099      nw_i=0
00100      if(.not.timereversal()) then
00101          write(6, "('TimeReversal off mode'")
00102          npm=2
00103          nw_i=-nw
00104          if(.not.tetra) call rx( ' tetra=T for timereversal=off')
00105      endif
00106      write(6,*)'Timereversal=',timereversal()
00107
00108 !! Write freq_r
00109      if(realomega .and. mpi__root) then
00110          ifif=ifile_handle()
00111          open(unit=ifif,file='freq_r') !write number of frequency points nwp and frequensies in 'freq_r'
00112      file
00113          write(ifif,"(2i8,' !(a.u.=2Ry)')") nw+1, nw_i
00114          do iw= nw_i,-1
00115              write(ifif,"(d23.15,2x,i6)") -freq_r(-iw),iw
00116          enddo
00117          do iw= 0,nw
00118              write(ifif,"(d23.15,2x,i6)") freq_r(iw),iw
00119          enddo
00120          close(ifif)
00121      endif
00122 !! Determine freq_i : gaussian frequencies x between (0,1) and w=(1-x)/x
00123      if (imagomega) then
00124          write(6,*)' freqimg: niw =',niw
00125          allocate( freq_i(niw) ,freqx(niw),wx(niw),expa(niw) )
00126          call freq01(niw,ua,
00127              o      freqx,freq_i,wx,expa)
00128          allocate(wiw(niw))
00129          do iw=1,niw
00130              wiw(iw)=wx(iw)/(2d0*pi*freqx(iw)*freqx(iw))
00131          enddo
00132          deallocate(freqx,wx,expa)
00133      endif
00134
00135 !! Plot frhis
00136      if(oncew(1)) then
00137          write(6,*)' we set frhis nwhis noo-->nee=',nwhis,noo,nee
00138          write(6,*)' --- Frequency bins to accumulate Im part (a.u.) are ---- '
00139          do ihis= 1, nwhis !min(10,nwhis)
00140              write(6,"(' ihis Init End=', i5,2f18.11)") ihis,frhis(ihis),frhis(ihis+1)
00141          enddo
00142      endif
00143      end subroutine getfreq
00144      end module m_freq

```

## 4.9 gwsrc/m\_hamindex.F File Reference

### Data Types

- module [m\\_hamindex](#)

*This is in `Im7K/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`. `norbt` is given in `gwsrc/readeigen.F` `init_readeigen2`.*



## 4.10 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   integer,parameter,private:: null=-999999
00006   logical,private:: debug=.false.
00007
00008   integer,protected:: ngrp=null, lxx=null, kxx=null,norbmtto=null
00009   integer,protected:: nbas,nqtt,ndimham=null
00010   integer,allocatable,protected:: ltab(:),ktab(:),offl(:),ispec(:), iclasst(:),offlrev(:,:),ibastab(:)
00011 )
00012   integer,allocatable,protected:: iqimap(:),iqmap(:),igmap(:),invgx(:),miat(:,:),ibasindex(:)
00013   !,ngvecp(:,:),ngvecprev(:,:),:)
00012   real(8),allocatable,protected:: symops(:,:),ag(:,:),tiat(:,:),shtvg(:,:), dlmm(:,:),:,),qq(:,:)
00013   real(8),protected:: plat(3,3),qlat(3,3)
00014   real(8),allocatable,protected:: qtt(:,:),qtti(:,:)
00015   integer,allocatable,protected:: igv2(:,:),napwk(:),igv2rev(:,:),:)
00016   integer,protected:: napwmx=null,lxxa=null
00017
00018
00019   integer:: norbtx=null
00020   integer:: nqi, nqnum,ngpmx,imx=null
00021   contains
00022
00023 !> get index ikt such that for qin(:)=qq(:,ikt)
00024   integer function getikt(qin) !return
00025   integer::i
00026   real(8):: qin(3)
00027 c   if(debug) print *,'nkt=',nkt
00028   do i=1, nqnum !*2 !nkt
00029     if(debug) print *,i,qin, qq(:,i)
00030     if(sum(abs(qin-qq(:,i)))<1d-8) then
00031       getikt=i
00032       return
00033     endif
00034   enddo
00035   print *,' getikt: xxx error nqnum qin=',nqnum,qin
00036   do i=1, nqnum !*2 !nkt
00037     write(*, "('i qq=',i3,f11.5)")i, qq(:,i)
00038   enddo
00039   call rx( ' getikt can not find ikt for given q')
00040   end function
00041
00042 !> write info for wave rotation.
00043   subroutine writehamindex()
00044   integer(4):: ifi
00045   logical::pmtton
00046   logical,save:: done=.false.
00047   if(done) call rx('writehamindex is already done')
00048   done=.true.
00049   ifi=1789
00050   open(ifi,file='HAMindex',form='unformatted')
00051   write(ifi)ngrp,nbas,kxx,lxx,nqtt,nqi,nqnum,imx,ngpmx,norbmtto
00052   write(ifi)symops,ag,invgx,miat,tiat,shtvg,qtt,qtti,iqmap,igmap,iqimap
00053   write(ifi)lxxa
00054   write(ifi)dlmm
00055   write(ifi)ibastab,ltab,ktab,offl,offlrev !for rotation of MTO. recovered sep2012 for EIBZ for hsfp0
00056   write(ifi)qq !,ngvecp,ngvecprev
00057   write(ifi)plat,qlat,napwmx
00058   if(napwmx/=0) then !for APW rotation used in rotwvigg
00059     write(ifi) igv2,napwk,igv2rev
00060   endif
00061   close(ifi)
00062   end subroutine writehamindex
00063
00064 !> read info for wave rotation.
00065   subroutine readhamindex()
00066   integer(4):: ifi,nkt
00067   logical::pmtton
00068   logical,save:: done=.false.
00069   if(done) call rx('readhamindex is already done')
00070   done=.true.
00071   ifi=1789
00072   open(ifi,file='HAMindex',form='unformatted')
00073   read(ifi)ngrp,nbas,kxx,lxx,nqtt,nqi,nqnum,imx,ngpmx,norbmtto
00074   allocate(symops(3,3,ngrp),ag(3,ngrp),qtt(3,nqtt),qtti(3,nqi))
00075   allocate(invqx(ngrp),miat(nbas,ngrp),tiat(3,nbas,ngrp),shtvg(3,ngrp))
00076   allocate(iqmap(nqtt),igmap(nqtt),iqimap(nqtt))
00077   write(6,*) 'ngrp=',ngrp
00078   read(ifi)symops,ag,invqx,miat,tiat,shtvg,qtt,qtti,iqmap,igmap,iqimap
00079   allocate( ltab(norbmtto),ktab(norbmtto),offl(norbmtto),ibastab(norbmtto) )
00080   allocate( offlrev(nbas,0:lxx,kxx))
00081   read(ifi) lxxa
00082   allocate( dlmm(-lxxa:lxxa, -lxxa:lxxa, 0:lxxa, ngrp))
```

```

00083      read(ifi) dlmm
00084      read(ifi) ibastab, ltab, ktab, offl, offlrev
00085 c      allocate( ngvecprev(-imx:imx, -imx:imx, -imx:imx, nqnum) )
00086 c      allocate( ngvecp(3, ngpmx, nqnum) )
00087      allocate( qq(3, nqnum) ) !this was qq(3, nqnum*2) until Aug2012 when shorbz had been used.
00088      read(ifi) qq !, ngvecp, ngvecprev
00089      read(ifi) plat, qlat, napwmx
00090      if(napwmx/=0) then !for APW rotation used in rotwvigg
00091          nkt=nqtt
00092          allocate( igv2(3, napwmx, nkt) )
00093          allocate( napwk(nkt) )
00094          allocate( igv2rev(-imx:imx, -imx:imx, -imx:imx, nkt) )
00095          read(ifi) igv2, napwk, igv2rev
00096      endif
00097      close(ifi)
00098      done=.true.
00099      end subroutine readhamindex
00100      end module
00101
00102

```

## 4.11 gwsrc/m\_tetwt.F File Reference

### Data Types

- module `m_tetwt`

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

## 4.12 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 !! - ww(k(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, protected :: whw(:)
00012      integer, allocatable, protected :: ihw(:, :, :), nhw(:, :, :), jhw(:, :, :), ibjb(:, :, : : )
00013      integer, protected :: nbnbx, nhwtot
00014      integer, allocatable, protected :: n1b(:, :, :), n2b(:, :, :), nbnb(:, : )
00015 !!
00016      contains !! -----
00017      subroutine tetdeallocate()
00018      deallocate(ihw, nhw, jhw, whw, ibjb, n1b, n2b, nbnb)
00019      end subroutine
00020
00021 !! routine -----
00022      subroutine gettetwt(q, iq, is, isf, nwgt, frhis, nwhis, npm,
00023      i qbas, ginv, ef, ngibz_mtet, nband, ekxx1, ekxx2, nctot, ecore,
00024      i ngbz, qbz, ngbzw, qbz, ntetf, idtetf, iblbz,
00025      i nbmx, ebm, mtet, eibzmode) !nov2016
00026 !! INPUT DATA: read only
00027 !! ngibz_mtet: is only for mtet=(/1,1,1/) --->(we usually use only this case)
00028 !!
00029 !! output data in returned in the module variables above.
00030
00031 !! we assume read_bzdata is called already
00032 c      use m_read_bzdata, only: qbas, ginv, ntetf, idtetf, iblbz !, qbz, ngbzw, qbz, ngibz
00033
00034 c      use m_read_eigen, only: readeval !we assume init_readeval is called already
00035 c      use m_genallcf_v3, only: ecore, nctot !we assume genallcf_v3 called already.
00036 c      use m_read_bzdata, only: ngbz, qbas, ginv, ngbzw, nteti, ntetf, idtetf, qbz, iblbz, ngibz, qbz
00037 c      use m_freq, only: !we assume getfreq is called already.
00038 c      & frhis, nwhis, npm !output of getfreq
00039 c      use m_zmel, only: nband
00040 c      use m_readfermi, only: readfermi, ef
00041
00042      implicit none
00043      integer, intent(in) :: is, isf, iq, nwgt(:), ngibz_mtet, ngbz, ngbzw, nband, npm, nwhis, nctot, nbmx, mtet(3)
00044      integer, intent(in) :: ntetf, idtetf(0:3, ntetf), iblbz(ngbzw)
00045      real(8), intent(in) :: q(3), qbas(3, 3), ginv(3, 3), ef, qbz(3, ngbz), qbz(3, ngbzw), ebm
00046      real(8), intent(in) :: ekxx1(nband, ngbz), ekxx2(nband, ngbz) !qbz(:, : )
00047      real(8), intent(in) :: frhis(1:nwhis+1), ecore(nctot, 2)

```

```

00048
00049     real(4),allocatable :: demin(:,:,:),demax(:,:,:))
00050     logical,allocatable :: iwgt(:,:,:))
00051     integer,allocatable:: nbnbtt(:,:),noccxvv(:) ! & idtetf(:,:),iblbz(:)
00052     logical :: eibzmode,tetra,tmpwwk=.false.,debug,eibz4x0
00053     integer::kx,ncc,job,jpm,noccxv(2)=-9999,ik,jhwtot,ib1,ib2,ibib,noccx,noccxv,verbose,ifief,
ifile_handle
00054     real(8),allocatable:: ecore_(:,:)
00055     if(nctot==0) then
00056         allocate(ecore_(1,2)) !this is dummry
00057     else
00058         allocate(ecore_(nctot,2))
00059         ecore_=ecore
00060     endif
00061
00062     tetra=.true.
00063 c     eibzmode = eibz4x0()
00064     debug=.false.
00065     if(verbose()>=100) debug=.true.
00066
00067 c     if(.not.allocated(nbnb))
00068         allocate( nbnb(nqbz,npm))
00069         allocate( nbnbtt(nqbz,npm)) !,ekxx1(nband,nqbz),ekxx2(nband,nqbz))
00070
00071 !!=====tetraini block tetra==.true.=====lini
00072 c     if(tetra) then
00073         write(6, "(' tetra mode nqbz nband ispin q=' ,2i7,i2,3f13.6)") nqbz,nband,is,q
00074
00075 !! move to upper level nov2016
00076 c$$$!!     ekxx1 for rk
00077 c$$$!!     ekxx2 for q+rk See tetwt4
00078 c$$$     do kx = 1, nqbz
00079 c$$$$cccccccccccccccc
00080 c$$$c     write(6, "(' kkkkk kx ',i4,3f9.4,3x,3f9.4)") kx,qbz(:,kx),qbzw(:,kx)
00081 c$$$$cccccccccccccccc
00082 c$$$     call readeval(qbz(:,kx), is, ekxx1(1:nband, kx) )
00083 c$$$     call readeval(q+qbz(:,kx), isf, ekxx2(1:nband, kx) )
00084 c$$$     enddo
00085
00086 c     takao-feb/2002 i replaced tetwt4(1d30) with tetwt5(job=0) ----
00087 c     ... get pairs(nlb n2b) with non-zero tetrahedron wieghts.
00088 c     the pairs are not dependent on the energy otemega
00089 c     in the denominator of the dielectric function.
00090     write(6, "(' -- First tetwt5 is to get size of array --')")
00091     job = 0
00092     if(npm==1) then
00093         ncc=0
00094     else
00095         ncc=nctot
00096     endif
00097     allocate( demin(nband+nctot,nband+ncc,nqbz,npm),
00098 & demax(nband+nctot,nband+ncc,nqbz,npm) )
00099     allocate( iwgt(nband+nctot,nband+ncc,nqbz,npm) )
00100 !     wgt, demin, demax may require too much memory in epsilon mode.
00101 !     We will have to remove these memory allocations in future.
00102 !     tetwt5x_dtet2 can be very slow because of these poor memory allocation.
00103 c     if(nctot==0) then
00104 c         deallocate(ecore)
00105 c         allocate(ecore_(1,2)) !this is dummry
00106 c     endif
00107     allocate(ibjb(1,1,1,1),ihw(1,1,1),jhw(1,1,1),nhw(1,1,1),whw(1)) !dummy
00108 c--- efermi
00109 c     ifief=ifile_handle()
00110 c     open(ifief,file='EFERMI')
00111 c     read(ifief,*) ef
00112 c     close(ifief)
00113 c     call readfermi() !comment out,since ef is passed nov2016
00114 ccccccccccccccccc
00115 c     print *, 'nqbz,nqbwz,nteti,ntetf,nqibz_mtet=',nqbz,nqbwz,nteti,ntetf,nqibz_mtet
00116
00117     call tetwt5x_dtet4(npm,ncc,
00118 i q, ekxx1, ekxx2, qbas,ginv,ef,
00119 d ntetf,nqbwz, nband,nqbz,
00120 i nctot,ecore_(1,is),idtetf,qbwz,iblbz,
00121 i job,
00122 o iwgt,nbnb, !job=0
00123 o demin,demax, !job=0
00124 i frhis, nwhis, ! job=1 not-used
00125 i nbnbx,ibjb,nhwtot, ! job=1 not-used
00126 i ihw,nhw,jhw, ! job=1 not-used
00127 o whw, ! job=1 not-used
00128 i iq,is,isf,nqibz_mtet, eibzmode,nwgt,
00129 i nbmx,ebmx,mtet) !nov2016
00130
00131     deallocate(ibjb,ihw,jhw,nhw,whw) !dummy
00132     nbnbx = maxval(nbnb(1:nqbz,1:npm)) !nbnbx = nbnbxx
00133     if(debug) write(6,*) ' nbnbx=',nbnbx

```

```

00134     allocate( n1b(nbnbx,nqbz,npm)
00135     & ,n2b(nbnbx,nqbz,npm) )
00136     n1b=0; n2b=0
00137     do jpm=1,npm
00138         call rsvwtk00_4(jpm, iwgt(1,1,1,jpm),nqbz,nband,nctot,ncc, nbnbx,
00139         o n1b(1,1,jpm), n2b(1,1,jpm), noccxvx(jpm), nbnbtt(1,jpm))
00140     enddo
00141     if(debug) then
00142         do kx = 1, nqbz
00143             do jpm = 1, npm
00144                 write(6, "('jpm kx minval n1b n2b=' ,4i5)") jpm,kx,
00145                 & minval(n1b(1:nbnb(kx,jpm),kx,jpm)),
00146                 & minval(n2b(1:nbnb(kx,jpm),kx,jpm))
00147             enddo
00148         enddo
00149     endif
00150     if(sum(abs(nbnb-nbnbtt))/=0) then
00151         do ik=1,nqbz
00152             write(6,*)
00153             write(6,*)"nbnb =" ,nbnb(ik,:)
00154             write(6,*)"nbnbtt=",nbnbtt(ik,:)
00155         enddo
00156         call rx( 'hx0fp0:sum(nbnb-nbnbtt)/=0' )
00157     endif
00158     noccxv = maxval(noccxvx)
00159     noccx = nctot + noccxv
00160     write(6,*) ' Tetra mode: nctot noccxv= ',nctot,noccxv
00161     deallocate(iwgt)
00162 c     endif
00163 c=====end of tetraini block=====1end
00164
00165 !! TetrahedronWeight_5 block. tetwt5 icx==,4,6,11 =====4ini
00166 c     if(icx==11) then !sf 21May02
00167 c         --- method(tetwt5) for the tetrahedron weight
00168 !         Histogram secctions are specified by frhis(1:nwp)
00169 !         The 1st bin is [frhis(1), frhis(2)] ...
00170 !         The last bin is [frhis(nw), frhis(nwp)].
00171 !         nwp=nw+1; frhis(1)=0
00172 !         takao-feb/2002
00173         if(abs(frhis(1))>1d-12) call rx( ' hx0fp0: we assume frhis(1)=0d0' )
00174         write(6,*) ' -----nbnbx nqbz= ',nbnbx,nqbz
00175 !!         ... make index sets
00176         allocate(ihw(nbnbx,nqbz,npm),nhw(nbnbx,nqbz,npm),jhw(nbnbx,nqbz,npm))
00177         ihw=0; nhw=0; jhw=0
00178         jhwtot = 1
00179         do jpm =1,npm
00180             do ik = 1,nqbz
00181                 do ibib = 1,nbnb(ik,jpm)
00182                     call hisrange( frhis, nwhis,
00183                     i demin(n1b(ibib,ik,jpm),n2b(ibib,ik,jpm),ik,jpm),
00184                     i demax(n1b(ibib,ik,jpm),n2b(ibib,ik,jpm),ik,jpm),
00185                     o ihw(ibib,ik,jpm),nhw(ibib,ik,jpm))
00186                     jhw(ibib,ik,jpm)= jhwtot
00187                     jhwtot = jhwtot + nhw(ibib,ik,jpm)
00188                 enddo
00189             enddo
00190         enddo
00191         nhwtot = jhwtot-1
00192         write(6,*) ' nhwtot=',nhwtot
00193         deallocate(demin,demax)
00194         allocate( whw(nhwtot), ! histo-weight
00195         & ibjb(nctot+nband,nband+ncc,nqbz,npm) )
00196         whw=0d0
00197         ibjb = 0
00198         do jpm=1,npm
00199             do ik = 1,nqbz
00200                 do ibib = 1,nbnb(ik,jpm)
00201                     ib1 = n1b(ibib,ik,jpm)
00202                     ib2 = n2b(ibib,ik,jpm)
00203                     ibjb(ib1,ib2,ik,jpm) = ibib
00204                 enddo
00205             enddo
00206         enddo
00207 !!         ... Generate the histogram weights whw
00208         job=1
00209         write(6,*) 'goto tetwt5x_dtet4 job=',job
00210         allocate(demin(1,1,1,1),demax(1,1,1,1),iwgt(1,1,1,1)) !dummy
00211         call tetwt5x_dtet4( npm,ncc,
00212         i q, ekxx1, ekxx2, qbas,ginv,ef,
00213         d ntetf,nqbwz, nband,nqbz,
00214         i nctot,ecore_(1,is),idtetf,qbwz,ib1bz,
00215         i job,
00216         o iwgt,nbnb, ! job=0
00217         o demin,demax, ! job=0
00218         i frhis,nwhis, ! job=1
00219         i nbnbx,ibjb,nhwtot, ! job=1
00220         i ihw,nhw,jhw, ! job=1

```

```

00221      o whw,                      ! job=1
00222      i iq,is,isf,nqibz_mtet, eibzmode,nwgt,
00223      i   nbmx,ebmx,mtet) !nov2016
00224      deallocate(demin,demax,iwgt,nbnbtt)
00225 !! =====TetrahedronWeight_5 block end =====
00226      end subroutine gettetwt
00227      end module

```

## 4.13 gwsr/m\_zmel.F File Reference

### Data Types

- module [m\\_zmel](#)

*Get the matrix element  $zmel = ZO^{-1} \langle MPB \psi | \psi \rangle$ , 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)

#### 4.13.1 Function/Subroutine Documentation

##### 4.13.1.1 subroutine timeshowx ( character\*(\*) info )

Definition at line 382 of file [m\\_zmel.F](#).

Here is the caller graph for this function:

## 4.14 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,nnn, ngrp,
00011         i nlmt0,nlnx,nlnxv,nlnxc,nlnmx,nlnmxv,nlnmxc, niw,
00012         i alat,delta,deltaw,esmr,symgrp,iclass,nlnmv, !,diw,dw
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 routine.
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 routine.
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 m_keyvalue,only: getkeyvalue

```

```

00035      implicit none
00036 !! -----
00037      integer,parameter:: NULL=-99999
00038 !! These are set by mptauof in main routine. 'call mptauof'
00039      integer,allocatable :: mlat(:, :)
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 routine
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,nmmx,nqmx, nctot,ncc)
00061 !! Get <phiq(q,ncc+nqmx,ispq) | phim(q-rkvec,nctot+nmmx,ispm) MPB(rkvec,ngb)> Z0^-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,nmmx,nqmx,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,nmmx,ispm,nctot, !middle-phi for phi_{q-rkvec}
00078      & q,nqini,nqmx,ispq,ncc ) !end-phi for phi_q
00079      end subroutine get_zmelt
00080 !! -----
00081 cold ntqxx-->nqmx
00082 cold nbmax -->nmmx
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 nqmx.
00085 !! nstate = nctot+nmmx
00086 !! allocate(zmeltt(ngb, nstate, nqmx))
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:nmmx | ncc + nqini:nqmx
00092 !      middle state | end state
00093 !
00094 !!--- For dielectric function, we use irot=1 kvec=rkvec=q. We calculate \chi(q).
00095 !!      q rkvec | q + rkvec
00096 !      nkmin:nkmax | nkqmin:nkqmx
00097 !      (we fix nkmin=1)
00098 !
00099 !      or
00100 !      nt0=nkmax-nkmin+1 | ntp0=nkqmx-nkqmin+1
00101 !      1:nt0 | 1:ntp0
00102 !      occ | unocc
00103 !      (cphi_k | cphi_kq !in x0kf)
00104 !      middle state | end state
00105 !!      rkvec= rk(:,k)-qq ! <phi(q+rk,nqmx)|phi(rk,nctot+nmmx) MPB(q,ngb )>
00106 !!      kvec = rk(:,k)-qq ! k
00107 !!
00108 !! NOTE: dimension
00109 !! nmtot = nctot+ nmmx-nmini+1
00110 !! nqtot = ncc + nqmx-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,nmmx,ispm,nctot, !middle for phi_{q-rkvec}
00116      & q,nqini,nqmx,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,ngtot
00127 real(8),allocatable :: dreallzzmel(:,:,:), dimagzzmel(:,:,:), 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::tolq=1d-8
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_rdp) call rx('m_zmel: not yet call rdp')
00141 if(.not.done_read_bzdata) call rx('m_zmel: not yet call read_bzdata')
00142
00143 if(init) then
00144 call minv33(qbas,qbasinv)
00145 allocate( cphiq(nlmt0,nband), cphim(nlmt0,nband))
00146 init=.false.
00147 endif
00148
00149 if(sum(abs(q-q_bk))>tolq) then
00150 call readcphi(q, nlmt0,ispq, quu, cphim )
00151 cphiq(1:nlmt0,1:ntq) = cphim(1:nlmt0,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-q_bk))>tolq) then
00162 call readcphi(qk, nlmt0,ispq, 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,ngpr)
00168 tr = tiat(:,invr)
00169 iatomp= miat(:,invr)
00170 symope= symgg(:,irot)
00171 shtv = matmul(symope,shtvg(:,invr))
00172 !! ppb= <Phi(SLn,r) Phi(SLn',r) B(S,i,Rr)>
00173 !! Note spin-dependence. Look for ix==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 ngtot = ncc + nqmax -nqini+1 ! = phi_end
00188 allocate( zzzmel(nbloch,nmtot,ngtot))
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,nlmt0,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,ngtot,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,nlmt0,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 ispg,ispm,ginv,
00230 i ngc,ngcmx, ngpmx,nband,itq,
00231 i symope, shvtv, qbas, qbasinv,qibz,qbz,nqibz,nqibz,
00232 i drealzzzmel, dimagzzzmel, nbloch, nctot,ncc,itps,
00233 o rmelt,cmelt)
00234 if(debug) write(6,*) ' sxcf_fall: 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,ixq !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$$$ read(ifpomat) qrr(:,ikpo),nn_,no,ixq !readin reduction matrix pomat
00280 c$$$ 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$$$              call rx( 'x0kf_v2h: P0mat err no/=ngb')
00302 c$$$          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$$$          rmelt = drealm(zmel)
00312 c$$$          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 c          if(allocated(zzzmel))deallocate(zzzmel) !rmel,cmel)
00324 c          if(debug) write(6,*) ' sxcf: goto wtt'
00325 c          if(debug) write(6, "('sum of rmelt cmelt=',4d23.16)") sum(rmelt),sum(cmelt)
00326 c          !! === End of zmel ; we now have matrix element zmel= rmelt + img* cmelt ===
00327 c          !TIME0_1301
00328
00329 c          !! Multiplied by ppovlz and reformat
00330 c          if(exchange) then
00331 c              if(debug) write(*,*) 'exchange mode 0000 ngb nmtot ngtot',ngb,nmtot,ngtot
00332 c              allocate( zmel(ngb, nmtot, ngtot))
00333 c              zmel = dcmplx(rmelt,cmelt)
00334 c              if(debug) write(*,*) 'exchange mode 1111'
00335 c              deallocate(rmelt,cmelt)
00336 c              if(debug) then
00337 c                  do it = 1,nmtot
00338 c                      write(6, "('wwwWSC ',i5,2f10.4)") it,sum(abs(zmel(:,it,1)))
00339 c                  enddo
00340 c                  write(*,*) 'eeeeeeeeeeeeee end of wwwWSC',nctot,nmmax
00341 c                  write(6,*) 'sumcheck ppovlz=',sum(abs(ppovlz(:,:)))
00342 c              endif
00343 c          !! OUTPUT zmel for exchange
00344 c          allocate(zmel(nmtot,ngtot,ngb))
00345
00346 c          if(verbose())>39) then
00347 c              write(*,*) 'info: USE GEMM FOR SUM (zmel=ppovlz) in sxcf_fal2.sc.F'
00348 c              write(*,*) 'zgemmsize',ngtot*nmtot,ngb,ngb
00349 c              write(*,*) 'size ,zmel',size(zmel,dim=1),size(zmel,dim=2),size(zmel,dim=3)
00350 c              write(*,*) 'size ,ppovlz',size(ppovlz,dim=1),size(ppovlz,dim=2)
00351 c              write(*,*) 'size ,zmel',size(zmel,dim=1),size(zmel,dim=2),size(zmel,dim=3)
00352 c          endif
00353 c          call flush(6)
00354 c          call zgemm('T','N',ngtot*nmtot,ngb,ngb,(1d0,0d0),
00355 c          .          zmel,ngb,ppovlz,ngb,(0d0,0d0),zmel,ngtot*nmtot )
00356 c          deallocate(zmel)
00357 c          else
00358 c          !! Correlation case. Get zmel
00359 c          if(debug) write(*,*) 'correlation mode 0000'
00360 c          nstate = nctot + nmmax ! = nstate for the case of correlation
00361 c          allocate(zmel(ngb, nmtot, ngtot))
00362 c          zmel= dcmplx(rmelt,-cmelt) !zmel= <itp|it,ib>
00363 c          deallocate(rmelt,cmelt)
00364 c          !! zmel(igb,it*itp) = C(ppovlz)*N(zmel(:,it*itp))
00365 c          !! C means Hermitian conjugate, N means normal
00366 c          !! http://www.netlib.org/lapack/lapack-3.1.1/html/zgemm.f.html
00367 c          !! OUTPUT
00368 c          allocate( zmel(ngb, nmtot, ngtot) )
00369
00370 c          if(debug) write(6, "('4 zzzppp222aaa ",3d13.5)') sum(abs(zmel)),sum(zmel)
00371 c          call zgemm('C','N',ngb, nmtot*ngtot,ngb,(1d0,0d0),
00372 c          .          ppovlz, ngb, zmel,ngb, (0d0,0d0),zmel,ngb)
00373 c          deallocate(zmel)
00374 c          if(debug) write(*,*) 'zz000 nmtot,ngb,nstate ',nmtot,ngb,ngtot
00375 c          if(debug) write(*,*) 'zz000 sumchk zmel ',sum(abs(zmel(1:ngb,1:nmtot,1:ngtot)))
00376 c          if(debug) write(*,*) 'correlation mode end'
00377 c          !TIME1_1301 "matmul_zmelp_povlz"
00378 c          endif
00379 c          end subroutine get_zmel2
00380 c          end module m_zmel
00381

```

```

00382      subroutine timeshowx(info)
00383      character(*) :: info
00384      write(*,'(a,$)')info
00385      call cputid(0)
00386      end
00387

```

## 4.15 gwsrsrc/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](#) (rkpr, rkmr, 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)

### 4.15.1 Function/Subroutine Documentation

#### 4.15.1.1 real(8) function [fac2m](#) ( i )

Definition at line [629](#) of file [mkjp.F](#).

Here is the caller graph for this function:

#### 4.15.1.2 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 [643](#) of file [mkjp.F](#).

Here is the call graph for this function:

Here is the caller graph for this function:

#### 4.15.1.3 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 [782](#) of file [mkjp.F](#).

Here is the caller graph for this function:

4.15.1.4 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, nx, 0:lxx) *sgbb* )

Definition at line 681 of file [mkjp.F](#).

Here is the call graph for this function:

Here is the caller graph for this function:

4.15.1.5 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) *rojpb*, complex(8), dimension(ngc, nx, (lxx+1)\*\*2) *sgpb*, complex(8), dimension(ngc, nx, (lxx+1)\*\*2) *fouvb* )

Definition at line 430 of file [mkjp.F](#).

Here is the call graph for this function:

Here is the caller graph for this function:

4.15.1.6 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 762 of file [mkjp.F](#).

Here is the call graph for this function:

Here is the caller graph for this function:

4.15.1.7 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 822 of file [mkjp.F](#).

Here is the caller graph for this function:

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

Definition at line 864 of file [mkjp.F](#).

Here is the caller graph for this function:

4.15.1.9 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*) *rojpb*, real(8), dimension(*nxx*, 0:lxx, *nbas*) *rojpb*, 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*) *fouvbb*, 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:

## 4.16 mkjp.F

```
00001      subroutine vcoulq_4(q,nbloch, ngc,
00002      &                      nbas, lx,lxx, nx,nxx,
00003      &                      alat, qlat, vol, ngvecc,
00004      &                      strx,rojpb,rojb, sgbb,sgpb, fouvbb,    !sgpp,fouvbb,
00005      i                      nblochpmx,bas,rmax,
00006      i                      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 rojpb 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 rojpb = <j_aL(r) | P(q+G)_aL > where
00030      c          |P(q+G)_aL> : the projection of exp(i (q+G) r) to aL channel.
00031      c          |j_aL>      : \def r^l/(2l+1)!! Y_L. The spherical bessel functions near r=0. Energy-dependence
00032      c          is omitted.
00033      c
00034      use m_lldata,only: ll
00035      implicit none
00036      integer(4) :: nbloch, nblochpmx, nbas,
00037      &          lxx,lx(nbas), nxx, nx(0:lxx,nbas)
00038      real(8)    :: egtpi,vol,q(3),fpi
```

```

00038
00039 ci structure con
00040     complex(8) :: strx((lxx+1)**2, nbas, (lxx+1)**2,nbas)
00041 ci |q+G|**2
00042     integer(4) :: ngc, ngvecc(3,ngc)
00043     real(8) :: qlat(3,3),alat,absqg2(ngc),qg(3)
00044
00045 ci rho-type onsite integral
00046     complex(8) :: rojp(ngc, (lxx+1)**2, nbas)
00047     real(8) :: rojb(nxx, 0:lxx, nbas)
00048 ci sigma-type onsite integral
00049     real(8) :: sgbb(nxx, nxx, 0:lxx, nbas)
00050     complex(8) :: sgpb(ngc, nxx, (lxx+1)**2, nbas)
00051 c & ,sgpp(ngc, ngc, (lxx+1)**2, nbas)
00052 ci Fourier
00053     complex(8) ::
00054     & fouvb(ngc, nxx, (lxx+1)**2, nbas)
00055 Co
00056     & ,vcoul(nblochpmx, nblochpmx)
00057 c & ,fouv(ngc, ngc, (lxx+1)**2, nbas)
00058
00059 cinternals
00060     integer(4) :: ibl1, ibl2, ig1, ig2, ibas, ibas1, ibas2,
00061     & l, m, n, nl, l1, m1, lml, n2, l2, m2, lm2, ipl1, ipl2
00062     integer(4) :: ibasbl(nbloch), nbl(nbloch), lbl(nbloch),
00063     & mbl(nbloch), lmb1(nbloch)
00064     real(8) :: pi, fpivol, tpiba
00065     complex(8) :: rojpstrx((lxx+1)**2,nbas)
00066
00067 c check
00068     complex(8),allocatable :: hh(:, :), oo(:, :), zz(:, :)
00069     real(8),allocatable :: eb(:)
00070
00071     complex(8),allocatable :: matp(:, :), matp2(:, :)
00072     complex(8) :: xxx
00073     integer(4) :: nblochngc, nev, nmix, ix
00074     logical :: ptest=.false. ! See ptest in basnfp.f
00075
00076 c-----
00077     real(8), allocatable :: cy(:), yl(:)
00078     complex(8),allocatable :: pjyl_(:, :), phase(:, :)
00079     complex(8) :: img=(0d0,1d0)
00080     real(8):: bas(3,nbas), r2s, rmax(nbas)
00081     integer(4):: lm
00082 c$$$#ifdef COMMONLL
00083 c$$$ integer(4)::ll(51**2)
00084 c$$$ common/llblock/ll
00085 c$$$#else
00086 c$$$ integer(4) :: ll
00087 c$$$ external ll
00088 c$$$#endif
00089     real(8):: fkk(0:lxx), fkj(0:lxx), fjk(0:lxx), fjj(0:lxx), sigx(0:lxx), radsig(0:lxx)
00090     complex(8):: fouv(ig1,ig2), fouv(ig2,ig1), sgpp(ig1,ig2)
00091
00092     integer(4):: nr,nr(nbas),ir,ig
00093     real(8):: eee, intl(nrx),int2x(nrx),phi(0:lxx),psi(0:lxx)
00094     & ,aa(nbas),bb(nbas),rkpr(nrx,0:lxx,nbas),rkmr(nrx,0:lxx,nbas)
00095     & ,rofi(nrx,nbas)
00096     real(8), allocatable:: ajr(:, :, :, :), al(:, :, :,:)
00097     logical :: debug=.false.
00098 c-----
00099     write(6,(' vcoul_4: nblochpmx nbloch ngc=",3i6)') nblochpmx,nbloch,ngc
00100 c print *, ' sum fouv=' ,sum(fouv(:, :, :, 1))
00101 c print *, ' sum fouv=' ,sum(fouv(:, :, :, 1))
00102     pi = 4d0*datan(1d0)
00103     fpi = 4*pi
00104     fpivol = 4*pi*vol
00105
00106 c---for sgpp fouv
00107     allocate( !ajr(1:nr,0:lx,ngc),al(1:nr,0:lx,ngc),rkpr(nr,0:lx),rkmr(nr,0:lx),
00108     & pjyl_((lxx+1)**2,ngc),phase(ngc,nbas) )
00109     allocate(cy((lxx+1)**2),yl((lxx+1)**2))
00110     call sylmnc(cy,lxx)
00111
00112 c=====
00113     vcoul = 0d0
00114 c-gvec
00115     tpiba = 2*pi/alat
00116     do ig1 = 1,ngc
00117         qg(1:3) = tpiba * (q(1:3)+ matmul(qlat, ngvecc(1:3,ig1)))
00118         absqg2(ig1) = sum(qg(1:3)**2)
00119 c---for sgpp fouv -----
00120         do ibas=1,nbas
00121             phase(ig1,ibas) = exp( img*sum(qg(1:3)*bas(1:3,ibas))*alat )
00122         enddo
00123         call sylm(qg/sqrt(absqg2(ig1)),yl,lxx,r2s) !spherical factor Y( q+G )
00124         do lm =1,(lxx+1)**2

```

```

00125         l = l1(lm)
00126         pjl_l(lm,igl) = fpi*img**1 *cy(lm)*yl(lm) * sqrt(absqg2(igl))**1 !*phase
00127         ! <jlyl | exp i q+G r> projection of exp(i q+G r) to jl yl on MT
00128     enddo
00129 c-----
00130     enddo
00131 c
00132
00133
00134 c-- index (mx,nx,lx,ibas) order.
00135     ibl1 = 0
00136     do ibas= 1, nbas
00137         do l = 0, lx(ibas)
00138 c         write(6,(' l ibas nx =" ,3i5)') l,nx(l,ibas),ibas
00139             do n = 1, nx(l,ibas)
00140                 do m = -1, 1
00141                     ibl1 = ibl1 + 1
00142                     ibasbl(ibl1) = ibas
00143                     nbl(ibl1) = n
00144                     lbl(ibl1) = l
00145                     mbl(ibl1) = m
00146                     lbl(ibl1) = l**2 + l+1 +m
00147 c         write(6,*)ibl1,n,l,m,lbl(ibl1)
00148             enddo
00149         enddo
00150     enddo
00151     enddo
00152     if(ibl1/= nbloch) then
00153         write(6,*)' ibl1 nbloch',ibl1, nbloch
00154 cstop2rx 2013.08.09 kino stop ' vcoulq: error ibl1/= nbloch'
00155         call rx( ' vcoulq: error ibl1/= nbloch')
00156     endif
00157
00158
00159 c-- <B|v|B> block
00160 c     write(6,*)' vcoulq: bvb block xxx rojbsum='
00161 c     write(6,*) sum(rojb(:, :, 1))
00162 c     write(6,*) sum(rojb(:, :, 2))
00163 c     write(6,*) sum(rojb(:, :, 3))
00164 c     write(6,*) sum(rojb(:, :, 4))
00165 c     write(6,*)' vcoulq: bvb block xxx sgbbsum='
00166 c     write(6,*) sum(sgbb(:, :, 1))
00167 c     write(6,*) sum(sgbb(:, :, 2))
00168 c     write(6,*) sum(sgbb(:, :, 3))
00169 c     write(6,*) sum(sgbb(:, :, 4))
00170     do ibl1= 1, nbloch
00171         ibas1= ibasbl(ibl1)
00172         n1 = nbl(ibl1)
00173         l1 = lbl(ibl1)
00174         m1 = mbl(ibl1)
00175         lm1 = lbl(ibl1)
00176         do ibl2= 1, ibl1
00177             ibas2= ibasbl(ibl2)
00178             n2 = nbl(ibl2)
00179             l2 = lbl(ibl2)
00180             m2 = mbl(ibl2)
00181             lm2 = lbl(ibl2)
00182             vcoul(ibl1,ibl2) =
00183 & rojb(n1, l1, ibas1) *strx(lm1,ibas1,lm2,ibas2)
00184 & *rojb(n2, l2, ibas2)
00185             if(ibas1==ibas2 .and. lm1==lm2) then
00186                 vcoul(ibl1,ibl2) = vcoul(ibl1,ibl2) + sgbb(n1,n2,l1, ibas1)
00187                 ! sigma-type contribution. onsite coulomb
00188             endif
00189         enddo
00190     enddo
00191
00192 ccccccccccccccccccccccccccccccc
00193 c     goto 1112
00194 ccccccccccccccccccccccccccccccc
00195
00196 c <P_G|v|B>
00197     if(debug) write(6,*)' vcoulq_4: pgvb block 1111'
00198     do ibl2= 1, nbloch
00199         ibas2= ibasbl(ibl2)
00200         n2 = nbl(ibl2)
00201         l2 = lbl(ibl2)
00202         m2 = mbl(ibl2)
00203         lm2 = lbl(ibl2)
00204         do igl = 1,ngc
00205             ip11 = nbloch + igl
00206             vcoul(ip11,ibl2) = fouvb(igl, n2, lm2, ibas2)
00207
00208             do ibas1= 1, nbas
00209                 do lm1 = 1, (lx(ibas1)+1)**2
00210                     vcoul(ip11,ibl2) = vcoul(ip11,ibl2) -
00211 & dconjg(rojp(igl, lm1, ibas1)) *strx(lm1,ibas1,lm2,ibas2)

```

```

00212      &      *rojb(n2, 12, ibas2)
00213              if (ibas1==ibas2 .and.lm1==lm2) then
00214                  vcoul(ip11,ibl2) = vcoul(ip11,ibl2) -
00215      &      sgpbl(ig1, n2, lm2, ibas2)
00216      endif
00217      enddo
00218      enddo
00219      enddo
00220      enddo
00221
00222      if(debug) write(6,*)' vcoulq_4: ajr allocate'
00223      C... prepare funciton ajr and al.
00224      C... ajr:spherical bessel, al: integral of (sperical bseel)*(rkp rkm)
00225      c-----
00226      allocate( ajr(nrx,0:lx,nbas,ngc), al(nrx,0:lx,nbas) )
00227      if(debug) write(6,*)' vcoulq_4: end ajr allocate'
00228      do ig1 = 1,ngc
00229          do ibas= 1,nbas
00230              if(debug) write(6, "('ccc: ',10i15)")ig1,ibas
00231              do ir = 1,nr(ibas)
00232                  call bessl(absqg2(ig1)*rofi(ir,ibas)**2,lxx,phi,psi)
00233                  do l = 0,lx(ibas)
00234
00235                      if(debug.and.ig==162.and.ibas==8) then
00236                          write(6, "('ccc: ',10i15)")ig1,ibas,ir,l
00237                          write(6,*)"ccc:", phi(l)
00238                          write(6,*)"ccc:", rofi(ir,ibas)
00239                      endif
00240
00241                      ajr(ir,l,ibas,ig1) = phi(l)* rofi(ir,ibas) ** (l +1 )
00242                      ! ajr = j_l(sqrt(e) r) * r / (sqrt(e))**l
00243                  enddo
00244              enddo
00245          enddo
00246      enddo
00247      c-----
00248
00249      c <P_G|v|P_G>
00250      if(debug) write(6,*)' vcoulq_4: pgvpg block'
00251      do ig1 = 1,ngc
00252          ip11 = nbloch + ig1
00253          rojpstrx = 0d0
00254          do ibas1= 1, nbas
00255              do lm1 = 1, (lx(ibas1)+1)**2
00256                  do ibas2= 1, nbas
00257                      do lm2 = 1, (lx(ibas2)+1)**2
00258                          rojpstrx(lm2, ibas2) = rojpstrx(lm2, ibas2)+
00259      &      dconjg(rojp(ig1, lm1, ibas1)) *strx(lm1,ibas1,lm2,ibas2)
00260                      enddo
00261                  enddo
00262              enddo
00263          enddo
00264
00265      c-----
00266      do ibas=1,nbas
00267          do l = 0,lx(ibas)
00268              call intn_smpxxx( rkpr(1,l,ibas), ajr(1,l,ibas,ig1),intl
00269      &      ,aa(ibas),bb(ibas),rofi(1,ibas),nr(ibas),0)
00270              call intn_smpxxx( rkmr(1,l,ibas), ajr(1,l,ibas,ig1),int2x
00271      &      ,aa(ibas),bb(ibas),rofi(1,ibas),nr(ibas),0)
00272              al(1, l,ibas) = 0d0
00273              al(2:nr(ibas),l,ibas) =
00274      &      rkmr(2:nr(ibas),l,ibas) *( intl(1)-intl(2:nr(ibas)) )
00275      &      + rkpr(2:nr(ibas),l,ibas) * int2x(2:nr(ibas))
00276          enddo
00277      enddo
00278      c-----
00279
00280      do ig2 = 1,ig1
00281          ip12 = nbloch + ig2
00282          if(ig1==ig2) vcoul(ip11,ip12) = fpivol/(absqg2(ig1) -eee) !eee is negative
00283          do ibas2= 1, nbas
00284      c... for fouvp and sgpp -----
00285              call wronkj( absqg2(ig1), absqg2(ig2), rmax(ibas2),lx(ibas2),
00286      o      fkk,fkj,fjk,fjj)
00287
00288              if(eee==0d0) then
00289                  call sigintpp( absqg2(ig1)**.5, absqg2(ig2)**.5, lx(ibas2), rmax(ibas2),
00290      o      sigx)
00291              else
00292                  do l = 0,lx(ibas2)
00293                      call gintxx(al(1,l,ibas2), ajr(1,l,ibas2,ig2)
00294      &      ,aa(ibas2),bb(ibas2),nr(ibas2), sigx(l))
00295                  enddo
00296              endif
00297              do l = 0,lx(ibas2)
00298                  radsig(l) = fpi/(2*l+1) * sigx(l)

```

```

00299             enddo
00300
00301 c-----
00302             do lm2 = 1, (lx(ibas2)+1)**2
00303                 l= ll(lm2)
00304 c...fouvvp sgpp-----
00305                 fouvvp_ig1_ig2 = fpi/(absqg2(ig1)-eee) *dconjg(pjyl_(lm2,ig1)*phase(ig1,ibas2))
00306             &      * (-fjj(l)) * pjyl_(lm2,ig2)*phase(ig2,ibas2)
00307                 fouvvp_ig2_ig1 = fpi/(absqg2(ig2)-eee) *dconjg(pjyl_(lm2,ig2)*phase(ig2,ibas2))
00308             &      * (-fjj(l)) * pjyl_(lm2,ig1)*phase(ig1,ibas2)
00309                 sgpp_ig1_ig2 = dconjg(pjyl_(lm2,ig1)*phase(ig1,ibas2))*radsig(l)
00310             &      * pjyl_(lm2,ig2)*phase(ig2,ibas2)
00311 c-----
00312                 vcoul(ipl1,ipl2) = vcoul(ipl1,ipl2)
00313             &      + rojpstrx(lm2,ibas2)*rojp(ig2, lm2, ibas2)
00314 c             &      - dconjg( fouvvp(ig2, ig1, lm2, ibas2)) !BugFix Mar5-01 It was dcmplx.
00315 c             &      - fouvvp(ig1, ig2, lm2, ibas2)
00316 c             &      + sgpp(ig1, ig2, lm2, ibas2)
00317             &      - dconjg( fouvvp_ig2_ig1 )
00318             &      - fouvvp_ig1_ig2
00319             &      + sgpp_ig1_ig2
00320             enddo
00321         enddo
00322     enddo
00323 enddo
00324 ccccccccccccccccccccccccccccccccccc
00325 c 1112 continue
00326 ccccccccccccccccccccccccccccccccccc
00327
00328
00329 c-- Right-Upper part of vcoul.
00330 if(debug) write(6,*)' vcoulq_4: right-upper'
00331 do ipl1=1, nbloch+ngc
00332     do ipl2=1, ipl1-1
00333         vcoul(ipl2,ipl1) = dconjg(vcoul(ipl1,ipl2))
00334     enddo
00335 enddo
00336
00337 ccccccccccccccccccccccccccccccccccc
00338 c test.xxxxxxxxxx
00339 c$$$      do ibl2= 1, nbloch
00340 c$$$          ibas2= ibasbl(ibl2)
00341 c$$$          n2 = nbl (ibl2)
00342 c$$$          l2 = lbl (ibl2)
00343 c$$$          m2 = mbl (ibl2)
00344 c$$$          lm2 = lmb1(ibl2)
00345 c$$$          if(l2==1.and.ibas2>2) then
00346 c$$$              vcoul(nbloch+1:nbloch+ngc, ibl2) = 0d0
00347 c$$$              vcoul(ibl2, nbloch+1:nbloch+ngc) = 0d0
00348 c$$$          endif
00349 c$$$      enddo
00350 ccccccccccccccccccccccccccccccccccc
00351
00352 c vcoul is in a.u. You have to multiply e-2=2 if you want to it in Ry,
00353 c     vcoul = 2d0*vcoul ! in Ry unit.
00354 c
00355
00356 c check write
00357     do ix = 1,nbloch+ngc,20
00358         write(6, "(' Diagonal Vcoul =',i5,2d18.10)") ix,vcoul(ix,ix)
00359     enddo
00360 if( allocated(y1) ) deallocate(y1)
00361 if( allocated(cy) ) deallocate(cy)
00362 if( allocated(phase)) deallocate(phase)
00363 if( allocated(pjyl_)) deallocate(pjyl_)
00364 if(.not.ptest) return
00365
00366
00367
00368 ccccccccccccccccccccccccccccccccccc
00369 c! Below ia a plane-wave test.
00370 c--- check! Coulomb by plane wave expansion.
00371 write(6,*) ' --- plane wave Coulomb matrix check 1---- '
00372 write(197,*) ' --- off diagonal ---- '
00373 nblochngc = nbloch+ngc
00374 allocate(matp(nblochngc),matp2(nblochngc))
00375 do ig1 = 1,ngc
00376     matp = 0d0
00377     do ibl2= 1, nbloch
00378         ibas2= ibasbl(ibl2)
00379         n2 = nbl(ibl2)
00380         l2 = lbl(ibl2)
00381         m2 = mbl(ibl2)
00382         lm2 = lmb1(ibl2)
00383         matp(ibl2) = fouvvp(ig1, n2, lm2, ibas2)*absqg2(ig1)/fpi
00384     enddo
00385     matp(nbloch+ig1) = 1d0

```



```

00386         ig2=ig1
00387 c      do ig2 = 1,ngc !off diagonal
00388         matp2 = 0d0
00389         do ibl2= 1, nbloch
00390             ibas2= ibasbl(ibl2)
00391             n2    = nbl(ibl2)
00392             l2    = lbl(ibl2)
00393             m2    = mbl(ibl2)
00394             lm2   = lmb1(ibl2)
00395             matp2(ibl2) = fouvb(ig2, n2, lm2, ibas2)*absqg2(ig2)/fpi
00396         enddo
00397         matp2(nbloch+ig2) = 1d0
00398         xxx= sum(
00399             & matmul(matp(1:nblochngc),vcoul(1:nblochngc,1:nblochngc))
00400             & *dconjg(matp2(1:nblochngc)) )
00401         if(ig1/=ig2) then !off diagonal
00402             if(abs(xxx)>1d-1 ) then
00403                 write(197,'(2i5, 2d13.6)') ig1,ig2, xxx
00404                 write(197,'("      matpp ", 2d13.6)')
00405             & vcoul(nbloch+ig1,nbloch+ig2)
00406             write(197,*)
00407             endif
00408         else
00409             write(196,'(2i5," exact=",3d13.6,"q ngsum=",3f8.4,i5)')
00410             & ig1,ig2,fpi*vol/absqg2(ig1)
00411             & , fpi*vol/absqg2(ig2),absqg2(ig1), q(1:3)
00412             & , sum(ngvecc(1:3,ig1)**2)
00413             write(196,'("      cal   =", 2d13.6)') xxx
00414             write(196,'("      vcoud=", 2d13.6)')
00415             & vcoul(nbloch+ig1,nbloch+ig2)
00416             write(196,*)
00417             endif
00418 c      enddo !off diagonal
00419         enddo
00420 c      deallocate(matp,matp2)
00421         stop ' *** ptest end *** See fort.196 and 197'
00422 c      ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00423         end
00424
00425
00426
00427
00428
00429 c=====
00430 subroutine mkjp_4( q,ngc,ngvecc, alat, qlat, lxx,lx,nxx,nx,
00431 i
00432 i eee,rofi,rkpr,rkmr,
00433 o rojp,sgpb,fouvb)
00434 C- Make integrals in each MT. and the Fourier matrix.
00435 Cr the integrals rojp, fouvb,fouvp
00436 Cr are for J_L(r)= j_l(sqrt(e) r)/sqrt(e)*l Y_L,
00437 Cr which behaves as r^1/(2l+1)!! near r=0.
00438 Cr
00439 Cr oniste integral is based on
00440 Cr 1/|r-r'| = \sum 4 pi /(2k+1) \frac{r_<^k}{r_>^{k+1}} } Y_L(r) Y_L(r')
00441 Cr See PRB34 5512(1986) for sigma type integral
00442 Cr
00443 use m_lldata,only: ll
00444 implicit none
00445 integer(4) :: ngc,ngvecc(3,ngc), lxx, lx, nxx,nx(0:lxx),nr,nrx
00446 real(8) :: q(3),bas(3), rprodx(nrx,nxx,0:lxx),a,b,rmax,alat,
00447 & qlat(3,3)
00448 ci rho-type onsite integral
00449 complex(8) :: rojp(ngc, (lxx+1)**2)
00450 ci sigma-type onsite integral
00451 complex(8) :: sgpb(ngc, nxx, (lxx+1)**2)
00452 c & sgpp(ngc, ngc, (lxx+1)**2)
00453 real(8),allocatable::cy(:),yl(:)
00454 ci Fourier
00455 complex(8) ::
00456 & fouvb(ngc, nxx, (lxx+1)**2)
00457 c & fouvp(ngc, ngc, (lxx+1)**2)
00458 c internal
00459 integer(4) :: nlx,ig1,ig2,l,n,ir,n1,n2,lm !, ibas
00460 c$$$#ifdef COMMONLL
00461 c$$$ integer(4)::ll(51**2)
00462 c$$$ common/llblock/ll
00463 c$$$#else
00464 c$$$ integer(4) :: ll
00465 c$$$ external ll
00466 c$$$#endif
00467 real(8) :: pi,fpi,tpiba, qg1(3),
00468 & fkk(0:lx),fkj(0:lx),fjk(0:lx),fjj(0:lx),absqg1,absqg2,
00469 & fac,radint,radsigo(0:lx),radsig(0:lx),phi(0:lx),psi(0:lx)
00470 & ,r2s,sig,sig1,sig2,sigx(0:lx),sig0(0:lx) ,qg2(3)
00471 complex(8) :: img =(0d0,1d0),phase
00472 complex(8),allocatable :: pjyl(:,)

```

```

00473      real(8),allocatable :: ajr(:, :, :), al(:, :, :), rkpr(:, :), rkmr(:, :),
00474      &  qg(:, :), absqg(:)
00475
00476
00477      real(8):: rofi(nrx), rkpr(nrx, 0:lx), rkmr(nrx, 0:lx), eee
00478      logical :: debug=.false.
00479 c rkpr(nr, 0:lx), rkmr(nr, 0:lx),
00480 c-----
00481      if(debug) print *, ' mkjp_4:'
00482      nlx = (lx+1)**2
00483      allocate(ajr(1:nr, 0:lx, ngc), al(1:nr, 0:lx, ngc),
00484      &  qg(3, ngc), absqg(ngc),
00485      &  pjyl((lx+1)**2, ngc) )
00486
00487      pi = 4d0*datan(1d0)
00488      fpi = 4*pi
00489      tpiba = 2*pi/alat
00490      allocate(cy((lx+1)**2), yl((lx+1)**2))
00491      call sylmnc(cy, lx)
00492      print *, ' mkjp_4: end of sylmnc'
00493 C... q+G and <J_L | exp(i q+G r)> J_L= j_l/sqrt(e)**1 Y_L
00494      do igl = 1, ngc
00495          qg(1:3, igl) = tpiba * (q(1:3) + matmul(qlat, ngvecc(1:3, igl)))
00496          qg1(1:3) = qg(1:3, igl)
00497          absqg(igl) = sqrt(sum(qg1(1:3)**2))
00498          absqg1 = absqg(igl)
00499          phase = exp( img*sum(qg1(1:3)*bas(1:3))*alat )
00500          call sylm(qg1/absqg1, yl, lx, r2s) !spherical factor Y( q+G )
00501          do lm = 1, nlx
00502              l = ll(lm)
00503              pjyl(lm, igl) = fpi*img**l *cy(lm)*yl(lm) *phase *absqg1**l
00504              ! <jlyl | exp i q+G r> projection of exp(i q+G r) to jl yl on MT
00505          enddo
00506      enddo
00507
00508 cc rofi and aj = r**l / (2l+1)!! \times r. Spherical Bessel at e=0.
00509 c      rofi(1) = 0d0
00510 c      do ir = 1, nr
00511 c          rofi(ir) = b*( exp(a*(ir-1)) - 1d0)
00512 c      enddo
00513 c      do l = 0, lx
00514 c          rkpr(1:nr, l) = rofi(1:nr)**(l +1 )
00515 c          rkmr(2:nr, l) = rofi(2:nr)**(-l-1 +1 )
00516 c          rkmr(1, l) = rkmr(2, l)
00517 c      enddo
00518
00519 c rojp
00520      if(debug) print *, ' mkjp_4: rojp'
00521      do igl = 1, ngc
00522          call wrnkj( absqg(igl)**2, eee, rmax, lx,
00523          o      fkk, fkj, fjk, fj)
00524          do lm = 1, nlx
00525              l = ll(lm)
00526              rojp(igl, lm) = (-fj(1))* pjyl(lm, igl)
00527          enddo
00528      enddo
00529
00530 c ajr
00531      do igl = 1, ngc
00532          do ir = 1, nr
00533              call bessl(absqg(igl)**2*rofi(ir)**2, lx, phi, psi)
00534              do l = 0, lx
00535                  ajr(ir, l, igl) = phi(l)* rofi(ir) ** (l +1 )
00536                  ! ajr = j_l(sqrt(e) r) * r / (sqrt(e))**l
00537              enddo
00538 cccccccccccccccccccccccccccccccccc
00539 c      write(116, '(i3, 10d13.6)') ir, rofi(ir), ajr(ir, 0:lx, igl)
00540 cccccccccccccccccccccccccccccccccc
00541      enddo
00542 cccccccccccccccccccccccccccccccccc
00543 c      write(6, *) igl, sum(ajr(1:nr, 0:lx, igl))
00544 cccccccccccccccccccccccccccccccccc
00545      enddo
00546
00547 c-----
00548      if(eee==0d0) then
00549 c          print *, ' mkjp_4: use sigintAn1 eee=0(r0c=infty) mode'
00550          do igl = 1, ngc
00551              call sigintan1( absqg(igl), lx, rofi, nr
00552              o      , al(1:nr, 0:lx, igl) )
00553          enddo
00554 c      else
00555 c We need to impliment a version of sigintAn1 to treat eee/=0 case...
00556      endif
00557
00558 c-----
00559 c sgpb

```

```

00560      do igl = 1,ngc
00561      do lm = 1,nlx
00562      l = ll(lm)
00563      do n = 1,nx(l)                                ! r jl          , r B(r)
00564      if(eee==0d0) then
00565      call gintxx(al(1,l,igl),rprodx(1,n,l),a,b,nr, sig )
00566      cccccccccccccccccccccc
00567      c      write(6, '( ' sgpb= ' ,3i5,2d14.6)') igl,n,lm, sgpb(igl,n,lm)
00568      cccccccccccccccccccccc
00569      else !for a while, we use this version of sgpb
00570      call sigint_4(rkpr(1,l),rkmr(1,l), lx,a,b,nr, ajr(1,l,igl),rprodx(1,n,l)
00571      &      , rofi, sig)
00572      endif
00573      sgpb(igl,n,lm) = dconjg(pjyl(lm,igl))* sig/(2*l+1)*fpi
00574      cccccccccccccccccccccc
00575      c      write(6, '( ' sgpb= ' ,3i5,2d14.6)') igl,n,lm, sgpb(igl,n,lm)
00576      c      write(6,*)
00577      cccccccccccccccccccccc
00578      enddo
00579      enddo
00580      enddo
00581      cccccccccccccccccccccc
00582      c      stop 'test end===== '
00583      cccccccccccccccccccccc
00584
00585      c-----
00586      c sgpp block----->removed
00587      c-----
00588
00589      c Fourier
00590      c fouvb
00591      if(debug) print *, ' mkjp_4: Four'
00592      fouvb=0d0
00593      do igl = 1,ngc
00594      do lm = 1,nlx
00595      l = ll(lm)
00596      do n = 1,nx(l)
00597      cccccccccccccccccccccccccccccccccccccccccccccccc
00598      c      print *, ' igl lm l n= ',igl,lm,l,n
00599      cccccccccccccccccccccccccccccccccccccccccccccccc
00600      call gintxx(ajr(1,l,igl), rprodx(1,n,l), a,b,nr,
00601      o      radint )
00602      cccccccccccccccccccccccccccccccccccccccccccccccc
00603      c      print *, ' radint=',radint
00604      cccccccccccccccccccccccccccccccccccccccccccccccc
00605      fouvb(igl, n, lm) =
00606      &      fpi/(absqg(igl)*2-eee) *dconjg(pjyl(lm,igl))*radint !eee is supposed to be negative
00607
00608      enddo
00609      enddo
00610      enddo
00611      cccccccccccccccccccccc
00612      c      write(6,*) ' fourvb sum= ',sum (fouvb)
00613      cccccccccccccccccccccc
00614
00615      c-----
00616      c fouvp block --->removed
00617      c-----
00618
00619      deallocate(ajr,al, qg,absqg, pjyl)
00620      if (allocated( cy )) deallocate(cy)
00621      if (allocated( yl )) deallocate(yl)
00622      end
00623
00624
00625
00626
00627
00628
00629      real(8) function fac2m(i)
00630      cC A table of (2l-1)!!
00631      c      data fac2l /1,1,3,15,105,945,10395,135135,2027025,34459425/
00632      logical,save:: init=.true.
00633      real(8),save:: fac2mm(0:100)
00634      if(init) then
00635      fac2mm(0)=1d0
00636      do l=1,100
00637      fac2mm(l)=fac2mm(l-1)*(2*l-1)
00638      enddo
00639      endif
00640      fac2m=fac2mm(i)
00641      end
00642      c=====
00643      subroutine genjh(eee,nr,a,b,lx, nrx,lxx,
00644      o      rofi,rkpr,rkmr)
00645      C-- Generate radial mesh rofi, spherical bessel, and hankel functions
00646      Cr rkpr, rkmr are real fucntions --

```

```

00647 ci eee=E= -kappa**2 <0
00648 cr      rkpr = (2l+1)!! * j_l(i sqrt(abs(E)) r) * r / (i sqrt(abs(E)))**l
00649 cr      rkmr = (2l-1)!! * h_l(i sqrt(abs(E)) r) * r * i*(i sqrt(abs(E)))**(l+1)
00650 cr rkpr reduced to be r**l*r      at E \to 0
00651 cr rkmr reduced to be r**(-l-1)*r at E \to 0
00652 C-----
00653     implicit none
00654     integer(4):: nr,lx, nrx,lxx,ir,l
00655     real(8):: a,b,eee,psi(0:lx),phi(0:lx)
00656     real(8):: rofi(nrx),rkpr(nrx,0:lxx),rkmr(nrx,0:lxx),fac2m
00657     rofi(1) = 0d0
00658     do ir = 1, nr
00659         rofi(ir) = b*( exp(a*(ir-1)) - 1d0)
00660     enddo
00661     if(eee==0d0) then
00662         do l = 0,lx
00663             rkpr(1:nr,l) = rofi(1:nr)**(l+1)
00664             rkmr(2:nr,l) = rofi(2:nr)**(-l-1+1)
00665             rkmr(1,l) = rkmr(2,l)
00666         enddo
00667     else
00668         do ir = 1, nr
00669             call bessl(eee*rofi(ir)**2,lx,phi(0:lx),psi(0:lx))
00670             do l = 0,lx
00671                 !fac2m(l)= (2l-1)!!
00672                 print *, ' phi=',l,phi(l),phi(l)*fac2m(l+1)
00673                 print *, ' psi=',l,psi(l),psi(l)/fac2m(l)
00674                 rkpr(ir,l) = phi(l)* rofi(ir)**(l+1) *fac2m(l+1)
00675                 if(ir/=1) rkmr(ir,l) = psi(l)* rofi(ir) **(-l ) /fac2m(l)
00676             enddo
00677         enddo
00678         rkmr(1,0:lx) = rkmr(2,0:lx)
00679     endif
00680 end
00681 C=====
00681 subroutine mkjb_4( lxx,lx,nxx,nx,
00682 i      a,b,nr,nrx,rprodx,
00683 i      rofi,rkpr,rkmr,
00684 o      rojb,sbbb)
00685 C--make integrals in each MT. and the Fourier matrix.
00686     implicit none
00687     integer(4) :: lxx, lx, nx, nx(0:lxx),nr,nrx
00688     real(8) :: q(3), rprodx(nrx,nxx,0:lxx),a,b
00689 ci rho-type onsite integral
00690     real(8) :: rojb(nxx, 0:lxx)
00691 ci sigma-type onsite integral
00692     real(8) :: sbbb(nxx, nx, 0:lxx)
00693 c internal
00694     integer(4) :: l,n,ir,nl,n2,l1
00695     real(8) ::
00696     & fac, xxx,fpi,pi,sig
00697     real(8) :: rofi(nrx),rkpr(nrx,0:lxx),rkmr(nrx,0:lxx)
00698     pi = 4d0*datan(1d0)
00699     fpi = 4*pi
00700 c     real(8),allocatable :: rkpr(:, :),rkmr(:, :)
00701 c
00702 c     allocate(rkpr(nr,0:lx),rkmr(nr,0:lx))
00703 C-----
00704 c rofi and aj = r**l / (2l+1)!! \times r. Spherical Bessel at e=0.
00705 ccccccccccccccccccccccccccccccccccc
00706 c     do l = 0,lx
00707 c         do n = 1,nx(1)
00708 c             do nl = 1,nx(1)
00709 c                 call gintxx(rprodx(1:nr,n,1), rprodx(1:nr,nl,1), a,b,nr,
00710 c                     xxx )
00711 c                 write(6,*) ' check rprodx = ',l,n,n-nl,xxx
00712 c             enddo
00713 c         enddo
00714 c     enddo
00715 c     stop 'xxx'
00716 ccccccccccccccccccccccccccccccccccc
00717
00718 c     rofi(1) = 0d0
00719 c     do ir = 1, nr
00720 c         rofi(ir) = b*( exp(a*(ir-1)) - 1d0)
00721 c     enddo
00722 c     do l = 0,lx
00723 c         rkpr(1:nr,l) = rofi(1:nr)**(l+1)
00724 c         rkmr(2:nr,l) = rofi(2:nr)**(-l-1) *rofi(2:nr)
00725 c         rkmr(1,l) = rkmr(2,l)
00726 c     enddo
00727
00728 C... initialize
00729     rojb=0d0
00730     sbbb=0d0
00731 c rojb
00732     fac = 1d0
00733     do l = 0,lx

```

```

00734         fac = fac/(2*1+1)
00735         do n = 1,nx(1)
00736             call gintxx(rkpr(1,1), rprodx(1,n,1), a,b,nr,
00737                 o rojb(n,1) )
00738         enddo
00739         rojb(1:nx(1),1) = fac*rojb(1:nx(1),1)
00740     enddo
00741 c sgbb
00742     do l = 0,lx
00743         do n1 = 1,nx(1)
00744             do n2 = 1,nx(1)
00745                 call sigint_4(rkpr(1,1),rkmr(1,1),lx,a,b,nr,rprodx(1,n1,1),rprodx(1,n2,1)
00746                     & , rofi,sig )
00747                 sgbb(n1, n2, 1)=sig/(2*1+1)*fpi
00748             enddo
00749         enddo
00750     enddo
00751 c     write(6,*) ' rojbsum=', sum(rojb(:,:)), sum(abs(rojb(:,:)))
00752 c     write(6,*) ' sgbbsum=', sum(sgbb(:,:)), sum(abs(sgbb(:,:)))
00753 c     write(6,*) ' sigint 1 1 0=',sgbb(1, 1, 0) !/(16d0*datan(1d0))
00754 c     write(6,*) ' sigint 1 1 0=',sgbb(1, 1, 0)
00755 c     sgbb(1, 1, 0) =0d0
00756 c     deallocate(rkpr,rkmr)
00757 c     end
00758
00759
00760
00761 c-----
00762 subroutine sigint_4(rkp,rkm,kmx,a,b,nr,phil,phi2,rofi, sig)
00763 implicit none
00764 integer(4) :: nr,kmx,k,ir
00765 real(8):: a,b, al(nr),a2(nr),b1(nr),rkp(nr),rkm(nr),
00766 & int1x(nr),int2x(nr), phil(nr), phi2(nr),rofi(nr),sig
00767 real(8),parameter:: fpi = 4d0*3.14159265358979323846d0
00768 c
00769     al(1) = 0d0; al(2:nr) = rkp(2:nr)
00770     a2(1) = 0d0; a2(2:nr) = rkm(2:nr)
00771     b1(1:nr) = phil(1:nr)
00772     call intn_smpxxx(al,b1,int1x,a,b,rofi,nr,0)
00773     call intn_smpxxx(a2,b1,int2x,a,b,rofi,nr,0)
00774 c
00775     al(1) = 0d0; al(2:nr) =
00776 & rkm(2:nr) * ( int1x(1)-int1x(2:nr) )+ rkp(2:nr) * int2x(2:nr)
00777     b1(1:nr) = phi2(1:nr)
00778     call gintxx(al,b1,a,b,nr, sig )
00779     end
00780
00781 c-----
00782 subroutine intn_smpxxx(g1,g2,int,a,b,rofi,nr,lr0)
00783 c-- integral of two wave function. used in ppdf
00784 c
00785 c int(r) = \int_(r)^(rmax) u1(r') u2(r') dr'
00786 c
00787 c lr0 dummy index, now not used.
00788 c simpson rule ,and with higher rule for odd deviation.
00789 c -----
00790     IMPLICIT none
00791     integer nr,ir,lr0
00792     double precision g1(nr),g2(nr),int(nr),a,b,rofi(nr),w1,w2,w3
00793     & ,ooth,foth
00794     data ooth,foth/0.3333333333333333,1.3333333333333333/
00795     data w1,w2,w3/0.4166666666666666,0.6666666666666666,
00796     & -0.08333333333333333/
00797     if(mod(nr,2).eq.0)
00798 Cstop2rx 2013.08.09 kino & stop ' INTN: nr should be odd for simpson integration rule'
00799     & call rx( ' INTN: nr should be odd for simpson integration rule')
00800 c
00801     int(1)=0.0d0
00802     DO 10 ir = 3,nr,2
00803         int(ir)=int(ir-2)
00804         & + ooth*g1(ir-2)*g2(ir-2)*( a*(b+rofi(ir-2)) )
00805         & + foth*g1(ir-1)*g2(ir-1)*( a*(b+rofi(ir-1)) )
00806         & + ooth*g1(ir)*g2(ir)*( a*(b+rofi(ir)) )
00807     10 CONTINUE
00808
00809 c At the value for odd points, use the same interpolation above
00810     do 20 ir = 2,nr-1,2
00811         int(ir)=int(ir-1)
00812         & + w1*g1(ir-1)*g2(ir-1)*( a*(b+rofi(ir-1)) )
00813         & + w2*g1(ir) *g2(ir)* ( a*(b+rofi(ir)) )
00814         & + w3*g1(ir+1)*g2(ir+1)*( a*(b+rofi(ir+1)) )
00815     20 continue
00816     do ir=1,nr
00817         int(ir)=int(nr)-int(ir)
00818     enddo
00819     END
00820

```

```

00821 c-----
00822     subroutine sigintanl( absqg, lx, rofi, nr,
00823         o          alint)
00824 c alint(r') = r' * \int_0^a r^2 {r_{<}}^1 / (r_{>})^{1+1} *
00825 c          j_l(absqg r)/absqg**1
00826     implicit none
00827     integer(4) :: nr,l,ir,lx
00828     real(8):: alint(nr,0:lx), rofi(nr),absqg
00829     real(8)::
00830     & ak(0:lx),aj(0:lx),dk(0:lx),dj(0:lx),
00831     & aknr(0:lx),ajnr(0:lx),dknr(0:lx),djnr(0:lx),
00832     & phi(0:lx),psi(0:lx)
00833 c---
00834 c     print *, ' sigintAnl: absqg=',absqg
00835 c     if(absqg<1d-10) then
00836 c         if(absqg<1d-6) then !23jan2004 1d-10 ok?
00837 cstop2rx 2013.08.09 kino          stop "sigintAnl: absqg=0 is not supported yet. Improve here."
00838 c         call rx( "sigintAnl: absqg=0 is not supported yet. Improve here.")
00839 c This part for absqg=0 has not been checked yet!
00840 c         call bessl(0d0,lx,phi,psi)
00841 c         do ir = 1,nr
00842 c             do l = 0,lx
00843 c                 alint(ir,l) = .5d0* rofi(nr)**2      * rofi(ir)**1      * phi(l)
00844 c             &          +(1d0/(2d0*1+3d0)-.5d0) * rofi(ir)**(1+2) * phi(l)
00845 c             enddo
00846 c         enddo
00847 c     else
00848 c         call radkj(absqg**2, rofi(nr),lx,aknr,ajnr,dknr,djnr,0)
00849 c         alint(1,:) = 0d0
00850 c         do ir = 2,nr
00851 c             call radkj(absqg**2, rofi(ir),lx,ak,aj,dk,dj,0)
00852 c             do l = 0,lx
00853 c                 alint(ir,l) = ( (2*l+1)* aj(l)
00854 c             &          -(l+1)* ajnr(l)+ rofi(nr)*djnr(l) )*(rofi(ir)/rofi(nr))**1)
00855 c             &          /absqg**2
00856 c             &          *rofi(ir)
00857 c             enddo
00858 c         enddo
00859 c     endif
00860 c     print *, ' sigintAnl: end'
00861 c     end
00862
00863 c-----
00864     subroutine sigintpp( absqg1, absqg2, lx, rmax,
00865         o          sig)
00866 c sig(l) = \int_0^a r^2 {r_{<}}^1 / (r_{>})^{1+1} *
00867 c          j_l(absqg1 r)/absqg1**1
00868 c          j_l(absqg2 r)/absqg2**1
00869 c e1\ne0 e2\ne0
00870     implicit none
00871     integer(4) :: l,lx
00872     real(8):: rmax,sig(0:lx), absqg1,absqg2, e1,e2,
00873     & ak1(0:lx),aj1(0:lx),dk1(0:lx),dj1(0:lx),
00874     & ak2(0:lx),aj2(0:lx),dk2(0:lx),dj2(0:lx),
00875     & fkk(0:lx),fkj(0:lx),fjk(0:lx),fjj(0:lx)
00876 c---
00877     e1 = absqg1**2
00878     e2 = absqg2**2
00879 c
00880 c     print *, " sigintpp",e1,e2
00881 c
00882 c     call wronkj( e1,e2, rmax,lx, fkk,fkj,fjk,fjj )
00883 c     call radkj( e1, rmax,lx, ak1,aj1,dk1,dj1,0)
00884 c     call radkj( e2, rmax,lx, ak2,aj2,dk2,dj2,0)
00885 c
00886 c     do l = 0,lx
00887 c         sig(l)= ( -1*(l+1)*rmax*aj1(l)*aj2(l)
00888 c             &          + rmax**3 * dj1(l)*dj2(l)
00889 c             &          + 0.5d0*rmax**2* (aj1(l)*dj2(l)+aj2(l)*dj1(l))
00890 c             &          - fjj(l)*(2*l+1)*(e1+e2)/2d0
00891 c             &          ) /(e1*e2)
00892 c     enddo
00893 c     end
00894

```

## 4.17 gwsrc/mkqg.F File Reference

### Functions/Subroutines

- subroutine [mkqg2](#) (alat, plat, symops, ngrp, nnn, iq0pin, QpGcut\_psi, QpGcut\_Cou, ifiqg, ifiqgc, gamma-cellctrl, Inq0iadd)

## 4.17.1 Function/Subroutine Documentation

4.17.1.1 subroutine mkqg2 ( real(8) *alat*, real(8), dimension(3,3) *plat*, real(8), dimension(3,3,ngrp) *symops*, integer *ngrp*, integer, dimension(3) *nnn*, integer *iq0pin*, real(8) *QpGcut\_psi*, real(8) *QpGcut\_Cou*, integer *ifiqq*, integer *ifiqgc*, integer *gammacellctrl*, logical *lnq0iadd* )

Definition at line 1 of file `mkqg.F`.

Here is the caller graph for this function:

## 4.18 mkqg.F

```
00001      subroutine mkqg2(alat,plat,symops,ngrp,nnn,iq0pin,
00002      &      qpgcut_psi, qpgcut_cou, ifiqq, ifiqgc,gammacellctrl,lnq0iadd)
00003      !! 'call getbzdatal' gives all following data
00004      use m_get_bzdatal,only: getbzdatal,
00005      &      nqbw, nqibz, nqbw,ntetf,nteti,nqbm,
00006      &      qbz,wbz,qibz,wibz,
00007      &      qbw, !qbasmc,
00008      &      idtetf, iblbz, idtet,
00009      &      irk, nstar, nstbz,
00010      &      qbm, qbw
00011      use m_keyvalue,only:getkeyvalue
00012      !! 'call getallq0p' give following data
00013      use m_q0p,only: getallq0p,
00014      &      q0i,wt,nq0i,nq0itru, nq0iadd
00015      !! == Make required q and G in the expansion of GW. ==
00016      !! |q+G| < QpGcut_psi for eigenfunction psi.
00017      !! |q+G| < QpGcut_Cou for coulomb interaction
00018      !!
00019      !! OUTPUT
00020      !! file handle= ifiqq, which contains q and G points for eigenfunction psi. --> QGpsi
00021      !! file handle= ifiqgc, which contains q and G points for Coulomb --> QGcou
00022      !!
00023      !! QGpsi(ifiqq), QGcou(ifiqgc), Q0P are written.
00024      !! See the end of console output.
00025      !! -----
00026      implicit none
00027      integer,parameter:: nqibz_r=0
00028      real(8)::qibz_r(3,1) !dummy
00029
00030      integer ::nnn(3),ifiqq,ifiqgc,ngcxx,
00031      &      ngrp,i,j,iq,iq0,ngp,ngpmx,ngc,ngcmx,nqnum,iq0pin,
00032      &      nline,nlinemax,ifsyml,iqq,is,nk,ix,nqnumx,il,ifkpt
00033      real(8) :: plat(3,3),qlat(3,3),q(3),dummy,qp(3),
00034      &      qpgcut_psi, qpgcut_cou,qpgcut,alpv(3),q0smean,sumt,alp,
00035      &      volum,voltot,pi,q0(3),qlat0(3,3), alat,trip1,
00036      &      symops(3,3,ngrp),xx,qqx(3),alpm
00037      integer,allocatable:: ngvecp(:,:), ngvecc(:,:),
00038      &      ngpn(:),ngcn(:),ngvect(:,:),ngcx(:), nqq(:)
00039      real(8),allocatable ::
00040      &      qq(:,:),qql(:,:),qq2(:,:),qqm(:,:)
00041      real(8) :: vol,ginv(3,3),aaa,dq(3) !,www
00042      integer :: mxkp,ifiqibz,iqibz,ifigwin,mtet(3),nm1,nm2,nm3
00043      logical ::tetrai,tetra,tetra_hsf0
00044      integer :: ifbz
00045      integer(4):: bzc=1
00046      logical :: readgwinput
00047      integer:: nqnum,ifiqmtet,verbose,q0pchoice,nn1,nn2,ifiqibz,iqbz !,auxfunq0p
00048      real(8)::aaij,bbij
00049      logical:: qbzreg
00050
00051      logical :: qreduce ,qreduce0
00052      real(8),allocatable:: qsave(:,:)
00053      integer:: imx,ifinin,il,imx0
00054      integer,allocatable :: ngvecprev(:,:),ngveccrev(:,:),)
00055
00056      real(8):: ddq(3)
00057      logical :: offmesh=.false. ,offmeshg=.false.
00058      logical :: regmesh=.false. ,regmeshg=.false. , timereversal
00059
00060      logical :: caca,debug=.true. !,newaniso
00061      integer:: imxc,nnn3(3),imx0c,imxl1(1,1)
00062      real(8):: deltaq,delta5,delta8,deltaq_scale!=1d0/3.0**5d0
00063
00064      integer:: nqi,ifix,ig,iq0i,lm
00065      real(8),allocatable:: wti(:),qi(:,:)
```

```

00066 integer:: ifidml!,iclose,iopen !,ifiwqfac
00067 integer:: llxxx,lm1,lm2
00068 real(8),allocatable:: funa(:,:),wsumau(:),y1l(:,:)
00069 real(8)::volinv,wtrue0,qg(3),alpgg2,qg2,tpiba
00070 character*99:: q0pf !nov2012
00071 integer:: dummyia(1,1),iimx,irradd,nmax
00072 real(8):: epstol=1d-8,tolq=1d-8,qx(3),qxx(3)
00073 logical :: newoffsetg !july2014
00074 real(8),allocatable:: wt0(:)
00075 integer,allocatable:: irr(:)
00076 real(8):: dq_(3),qlatbz(3,3)
00077 integer:: gammacellctrl,nnng(3),ifile_handle,ifi0,itet
00078 real(8):: imat33(3,3)
00079 logical:: lnq0iadd
00080
00081 c-----
00082 print *, ' mkqg2: '
00083 greduce0 = greduce()
00084 newoffsetg=.true. !newaniso()
00085 if(iq0pin == 101) then
00086     iq0pin=1
00087     newoffsetg=.false. !for old oldset Gamma case
00088 endif
00089
00090 !! I (apr2016takao) think iq0pin==3 is used little now.
00091 !! band case --- iq0pin == 3 ==>read syml file. E.g for Imag-part calcualtion along a symmetry line.
00092 !!      nqq(is),qq1(1:3,is),qq2(1:3,is),is =1,nline
00093 if(iq0pin == 3) then
00094     greduce0=.false.
00095     nlinemax = 50
00096     allocate(nqq(nlinemax),qq1(1:3,nlinemax),qq2(1:3,nlinemax))
00097     ifsyml = ifile_handle()
00098     open(ifsyml,file='SYML')
00099     nline = 0
00100     do
00101         nline = nline + 1
00102         read(ifsyml,*,err=601,end=601)
00103         & nqq(nline),qq1(1:3,nline),qq2(1:3,nline)
00104     enddo
00105 601 continue
00106     close(ifsyml)
00107     nline = nline - 1
00108     write(6,"(/' Symmetry lines:/' points',12x,'start',22x,'end')")
00109     do is=1,nline
00110         write(6,"(i6,2x,3f8.4,2x,3f8.4)")
00111         & nqq(is),(qq1(i,is),i=1,3),(qq2(i,is),i=1,3)
00112     enddo
00113     nqnumx = sum(nqq(1:nline))
00114     allocate( qq(1:3,nqnumx),irr(nqnumx) )
00115     iqq = 0
00116     do is = 1,nline
00117         nk = nqq(is)
00118         do iq=1,nk
00119             xx = 0d0
00120             if(nk>1) xx=(iq-1d0)/(nk-1d0)
00121             qgx = xx*qq2(1:3,is)+(1d0-xx)*qq1(1:3,is)
00122             iqq = iqq + 1
00123             qq(1:3,iqq) = qgx
00124             write (6,"(' q=' ,3f7.3)") qq(1:3,iqq)
00125         enddo
00126     enddo
00127     nqnum = iqq
00128     write (6,"(' Total number of q-points:',i5/)") nqnum
00129     call minv33tp(plat,qlat) !it was dinv33(plat,1,qlat) by Ferdi
00130     goto 2001
00131 endif
00132
00133 !! we usually use negative delta (tetrahedron).
00134 call getkeyvalue("GWinput","delta",aaa)
00135 if(aaa<0d0) then
00136     print * , 'GWinput delta<0: tetrahedron method for x0'
00137     tetraf=.true.
00138 else
00139     print * , 'GWinput delta>0: not use tetrahedron method for x0'
00140     tetraf=.false.
00141 endif
00142
00143 !! plat,qlat,ginv
00144 voltot = abs(alat**3*tripl(plat,plat(1,2),plat(1,3)))
00145 call minv33tp(plat,qlat)
00146 call minv33(qlat,ginv)
00147 imat33=0d0
00148 imat33(1,1)=1d0
00149 imat33(2,2)=1d0
00150 imat33(3,3)=1d0
00151 if(sum(abs(matmul(transpose(qlat),plat)-imat33))>tolq) call rx('qlat*plat err')
00152 if(sum(abs(matmul(ginv,qlat)-imat33))>tolq) call rx('ginv=qlat^-1 err')

```



```

00153      write(6,*)'=== plat ==='
00154      write(6,"(3d23.15)") plat
00155      write(6,*)'=== qlat ==='
00156      write(6,"(3d23.15)") qlat
00157 c      write(6,*)'=== ginv=== '
00158 c      write(6,"(3f9.4)") ginv
00159 !! We now use mtet=(1,1,1). If we like to recover this, examine code again.
00160      call getkeyvalue("GWinput","multitet",mtet,3,default=(/1,1,1/))
00161
00162 !! For gammacellctrl==2, we only consider tetrahedron method within the Gammacell.
00163 !! The Gammacell is a part of BZ made from three vectors following qlatbz=(qlat(:,1)/n1q,...)
00164 !! Then the Gamma point is in the middle of micro_qlat = (qlat(:,1)/n1q,qlat(:,2)/n2q,...)
00165 !! To get qbz which is in the Gamma cell, we use shift in the getbzdatal for gammacellctrl=2.
00166 !! Tetrahedron method is applied for such qbz.
00167      if(gammacellctrl==2) then
00168          do i=1,3
00169              qlatbz(:,i) = qlat(:,i)/nnn(i) !qlat for Gamma cell
00170          enddo
00171          call getkeyvalue("GWinput","GammaDivnln2n3",nnng,3)
00172          nnn = nnng !division of Gamma cell
00173          dq_ = -matmul(qlatbz(1:3,1:3),(/.5d0,.5d0,.5d0/))
00174 !This shift vector is to make the Gamma point centered in the Gamma cell.
00175          tetrai=.false.
00176          call minv33(qlatbz,ginv)
00177          write(6,*)'=== Gammacell qlatgz ==='
00178          write(6,"(3d23.15)") qlatbz
00179          write(6,*)'=== Gammacell ginv ==='
00180          write(6,"(3f9.4)") ginv
00181 cccccccccccccccccccccccccc
00182 c      qreduce0=.false.
00183 cccccccccccccccccccccccccc
00184      else
00185          qlatbz(:,i) = qlat(:,i)
00186          tetrai = .true. !used in heftet tetra_hsf0()
00187          dq_ = 0d0
00188          if(.not.qbzreg()) dq_ = -matmul(qlat(1:3,1:3),(/.5d0/nnn(1),.5d0/nnn(2),.5d0/nnn(3)/))
00189          !This dq_ is off-gamma mesh, used when qbzreg=F
00190      endif
00191 cccccccccccccccccccccccccccc
00192 c      dq_=0d0
00193 cccccccccccccccccccccccccccc
00194      if(sum(abs(dq_))>tolq) write(6,(' Shift vector (skip Gamma) by dq_="',3f9.4)')dq_
00195
00196 !! Get BZ data by 'call getbzdatal'
00197 !! See following data after 'use getbzdatal' at the top of this routine.
00198 !! In the case of gammacellctrl=2, we only calculate quantities in the Gamma cell.
00199 !! Thus we have special meanings of ngbz. GWinput --> GammaDivnln2n3 4 4 4
00200      call getbzdatal(qlatbz,nnn, !plat bzcasc,
00201          & symops,ngrp,tetrai,tetra,mtet,gammacellctrl) !all are inputs. output: See use.
00202
00203 !! Write BZDATA
00204      print *, ' Writing BZDATA...'
00205      ifbz = ifile_handle()
00206      open (ifbz, file='BZDATA')
00207      write(ifbz,"(10i10)") ngbz,nqibz, ngbzw, ntetf, nteti,ngrp !,nqibz_r
00208      write(ifbz,"(10i10)") nnn(1:3) !n1q,n2q,n3q
00209      write(ifbz,"(3d24.16)") qlat,ginv!,qbasmc
00210      do iqibz = 1,nqibz
00211          write(ifbz,"(4d24.16,i9)") qibz(1:3,iqibz),wibz(iqibz),nstar(iqibz)
00212 c      write(6,"('bbbbbbbbb ',4d24.16,i9)") qibz(1:3,iqibz),wibz(iqibz),nstar(iqibz)
00213      write(ifbz,"(100i8)") irk(iqibz,1:ngrp)
00214      enddo
00215 c      write(ifbz,"(i10)") nqibz_r
00216 c      do iqibz = 1,nqibz_r
00217 c          write(ifbz,"(3d24.16)") qibz_r(1:3,iqibz)
00218 c      enddo
00219 c      do iqbz = 1,ngbz
00220 c          write(ifbz,"(4d24.16,i10)") qbz(1:3,iqbz),wbz(iqbz),nstbz(iqbz)
00221 c      enddo
00222      if(ntetf>0) then
00223          write(ifbz,"(4i10)") (idtetf(0:3,itet),itet=1,ntetf)
00224          write(ifbz,"(i9,3d24.16)") (iblbz(iqbz), qbz(1:3,iqbz),iqbz=1,ngbzw)
00225      endif
00226      if(nteti>0) write(ifbz,"(5i10)") (idteti(0:4,iteti),itet=1,nteti)
00227      write(ifbz,"(3d24.16,' !dq_')") dq_
00228      close(ifbz)
00229 !! Write QIBZ
00230      write(6,*)' qibz are written in QIBZ file...'
00231      ifiqibz = ifile_handle()
00232      open (ifiqibz, file='QIBZ') !write q-points in IBZ.
00233      write(ifiqibz,"(i10)") nqibz
00234      do iqibz = 1,nqibz
00235          write(ifiqibz,"(3d24.16,3x,d24.16)") qibz(1:3,iqibz),wibz(iqibz)
00236      enddo
00237      close(ifiqibz)
00238 !! Write QBZ
00239      ifiqbz = ifile_handle()

```

```

00240     open (ifqibz, file='QBZ') !write q-points in IBZ.
00241     write(ifqibz,"(i10)") nqbz
00242     do iqbz = 1,nqbz
00243         write(ifqibz,"(3d24.16,3x,d24.16)") qbz(1:3,iqbz)
00244     enddo
00245     close(ifqibz)
00246 !! Write KPNTin1BZ.mkqg.chk (files *.chk is only for check.).
00247     ifkpt = ifile_handle()
00248     open(ifkpt,file='KPTin1BZ.mkqg.chk')
00249     write(ifkpt,*)"  qbz --> shoten(qbz)"
00250     do
00251         il = 1,nqbz
00252         call shorbz(qbz(1,il),qp,q1at,plat)
00253         write (ifkpt,"(1x,i7,4f10.5,' ',3f10.5)")
00254         & il,qbz(1,il),qbz(2,il),qbz(3,il),wbz(il),qp
00255     enddo
00256     close (ifkpt)
00257     write(6,*) ' --- TOTAL num of q =',nqbz
00258     write(6,*)
00259     write(6,(' ' ngrp = ',i3))ngrp
00260     write(6,(' ' qibz=",i6,3f12.5'))(i,qibz(1:3,i),i=1,min(10,nqibz))
00261     write(6,*)" ... QIBZ is written in QIBZ file ..."
00262 !! alpha is for auxially function for offset Gamma method.
00263     call getkeyvalue("GWinput","alpha_offG",alp,default=-1d60)
00264     alp(:)=alp
00265     if(alp===-1d60) then
00266         call getkeyvalue("GWinput","alpha_offG_vec",alp,3,default=(-1d50,0d0,0d0/))
00267         if(alp(1)===-1d50) then
00268             call rx( ' mkqg: No alpha_offG nor alpha_offG_vec given in GWinput')
00269         endif
00270     endif
00271     print *
00272     print *, ' alp=',alp
00273     print *
00274     alpm = minval(alp)
00275     if(alpm<=0d0) call rx( 'alpha_offG or alpha_offG_vec <=0')
00276 !! Determine G vectors for q points set by getgv2
00277     if(iq0pin==1) then ! --- get q0x (offsetted q=0 point) -----
00278 !! I now think this QpGwut is large enough.
00279     qpqcut = sqrt(25d0/alpm) !a.u. !exp( -alp*QpGcut**2) !alp * QpGcut**2 = 22
00280     allocate( ngcx(nqbz) )
00281     ngcx=1
00282     do iq = 1, nqbz
00283         q = qbz(1:3,iq)
00284         call getgv2(alat,plat,q1at,q, qpqcut, 1, ngcx(iq), dummyia)
00285     enddo
00286     ngcxc = maxval(ngcx)
00287     allocate( ngvect(3,ngcxc,nqbz) )
00288     print *, ' goto getgv2: ngcxc=',ngcxc
00289     do iq = 1, nqbz
00290         q = qbz(1:3,iq)
00291         call getgv2( alat,plat,q1at, q, qpqcut, 2,
00292         & ngcx(iq), ngvect(1:3,1:ngcx(iq),iq) )
00293     enddo
00294     endif
00295 !! getallq0p all inputs
00296 !! Q0P is offset Gamma or k point given in GWinput
00297 !! see use m_q0p => q0i,wt,nq0i,nq0itrue are outputs
00298 !! we now have q0i(:,nq0i+1,nq0i+nq0iadd).
00299 !! q0i(:,1:nq0i+nq0iadd) contains all q0x(:,i)= q1at(:,i)/nnn(i)/2d0*deltaq_scale() for i=1,3.
00300 !! lnq0iadd=.true.
00301 c
00302 call getallq0p(iq0pin,newoffsetg,alat,plat,q1at,nnn,alp,alp, !apr2016
00303 i ngcxc,ngcx,nqbz,nqibz,nstbz,qbz,qibz,symops,ngrp,ngvect,lnq0iadd)
00304 c
00305 print *, 'size q0i=',size(q0i),ubound(q0i),lbound(q0i)
00306 do i=nq0i+1,nq0i+nq0iadd
00307     write(6,(' ' q0iadd= ', i3, 3f10.5)") i,q0i(:,i)
00308 enddo
00309 !! Four kinds of mesh points. Q0P means offset Gamma (slightly different from Gamma).
00310 !! Which we need?
00311 !! 1. regular
00312 !! 2. offregular (not including Gamma)
00313 !! 3. regular + Q0P
00314 !! 4. offregular + Q0P
00315 if(iq0pin==2) then !this is just for dielectric case
00316     regmesh = qbzreg()
00317 else
00318     regmesh = .true.
00319 endif
00320 regmeshg = qbzreg() !Gamma mesh based on regular mesh
00321 offmesh = .not.qbzreg() !we fix bzcass=1 now. apr2015.
00322 offmeshg = .not.qbzreg() !Gamma mesh based on off-regular mesh
00323 print *, ' regmesh offmeshg=', regmesh,regmeshg !regular, regular+shifted
00324 print *, ' offmesh offmeshg=', offmesh,offmeshg !offregmesh, offregular+shifted
00325
00326 !! We check wether all q0i \in qbz or not. <--- Takao think this block is not necessary now.

```

```

00327      call minv33(qlat,ginv)
00328      nqnum = nqbz
00329      allocate( qq(1:3,nqnum),irr(nqnum) )
00330      qq(1:3,1:nqbz) = qbz(1:3,1:nqbz)
00331      do iq0i=1,nq0i+nq0iadd
00332          do iq=1,nqbz
00333              if(sum(abs(q0i(:,iq0i)-qq(:,iq)))<tolq) goto 2112
00334              call rangedq( matmul(ginv,q0i(:,iq0i)-qq(:,iq)), qx)
00335              if(sum(abs(qx))< tolq) goto 2112
00336          enddo
00337          goto 2111
00338      2112      continue
00339          qq(:,iq) = q0i(:,iq0i) !replaced with equivalent q0i.
00340      enddo
00341      print *, ' --- We find all q0i in qbz. Skip greduce.'
00342      goto 2001
00343      2111      continue
00344
00345      !! Accumulate all required q points
00346      deallocate(qq,irr)
00347      nqnum = nqbz + nqbz*(nq0i+nq0iadd)
00348      nqnum = nqnum + 1 !add Gamma
00349      nqnum = nqnum + nq0i + nq0iadd !add Gamma + q0i
00350      allocate( qq(1:3,nqnum),irr(nqnum) )
00351      ix = 0
00352      if(regmesh) then
00353          qq(1:3,1:nqbz) = qbz(1:3,1:nqbz)
00354          ix = ix+ nqbz
00355      endif
00356      !! - Off Regular mesh.
00357      if(offmesh) then
00358          do iq = 1, nqbz
00359              ix = ix+1
00360              qq(1:3,ix) = qbz(1:3,iq) - dq_
00361          enddo
00362      endif
00363      nnn = ix !nlq*n2q*n3q! if(offmesh) nnn = 2*nlq*n2q*n3q
00364      c print *, ' nnn=',nnn !This is the number to calcualte Vxc
00365      !! - Shifted mesh
00366      if(regmeshg) then
00367          do iq00 = 1, nq0i+ nq0iadd
00368              do iq = 1, nqbz
00369                  ix = ix+1
00370                  qq(1:3,ix) = qbz(1:3,iq) + q0i(1:3,iq00)
00371              enddo
00372          enddo
00373      endif
00374      if(offmeshg) then
00375          do iq00 = 1, nq0i+ nq0iadd
00376              do iq = 1, nqbz
00377                  ix = ix+1
00378                  qq(1:3,ix) = qbz(1:3,iq) - dq_ + q0i(1:3,iq00)
00379              enddo
00380          enddo
00381      endif
00382      !! - Add offset Gamma and Gamma point (these can be removed by greduce and q0irre)
00383      do iq00 = 1, nq0i+ nq0iadd
00384          ix = ix+1
00385          qq(1:3,ix) = q0i(1:3,iq00)
00386      enddo
00387      ix=ix+1
00388      qq(1:3,ix)=0d0
00389
00390
00391
00392      !! (this mtet block is not used now) Get qqm; q point for eigenvalues.
00393      !! Saved to Qmtet. Not so much used now...
00394      !! We need check again if we like to use this branch again (2016apr)
00395      if(sum(abs(mtet))/=3) then
00396          nqnumm= nqbzm * (nq0i+ nq0iadd +1)
00397          allocate( qqm(1:3,nqnumm) )
00398          ix=0
00399          do iq00 = 1, 1 + nq0i+ nq0iadd
00400              do iq = 1, nqbzm
00401                  ix = ix+1
00402                  if(iq00=1) then
00403                      qqm(1:3,ix) = qbz(1:3,iq)
00404                  else
00405                      qqm(1:3,ix) = q0i(1:3,iq00-1) + qbz(1:3,iq)
00406                  endif
00407              enddo
00408          enddo
00409          ifiqmtet=ifile_handle()
00410          open(ifiqmtet, file='Qmtet')
00411          write(ifiqmtet,"(i10)") nqnumm
00412          do iq=1,nqnumm
00413              write(ifiqmtet,"(3d24.16)") qqm(1:3,iq)

```

```

00414         enddo
00415         close(ifiqmtet)
00416         deallocate(qqm)
00417     endif
00418
00419 !! Remove equivalent q point by the translational symmetry
00420     if( qreduce0 ) then
00421         print *, 'goto qgsave nq0i nq0iadd nqnum', nq0i, nq0iadd, nqnum
00422         call cputid(0)
00423         nmax= nq0i+nq0iadd+nqnum
00424         allocate(qsave(3,nmax)) !,qsavel(nmax))
00425         imx=0
00426         if(iq0pin /=1) then
00427             do iq=1,nq0i+ nq0iadd
00428                 call qgsave(q0i(1:3,iq),nmax,ginv,qsave,imx)
00429             enddo
00430         endif
00431         do iq=1,nqnum
00432             call qgsave(qq(1:3,iq),nmax,ginv,qsave,imx)
00433         enddo
00434         nqnum = imx
00435         qq(:,1:imx)=qsave(:,1:imx)
00436         deallocate(qsave)
00437     endif
00438 !! -----
00439 2001 continue
00440 !! -----
00441
00442
00443 !! Here we get all requied q points. We do reduce them by space group symmetry.
00444     if(allocated(wt0)) deallocate(wt0)
00445     allocate(wt0(nqnum+nq0i+ nq0iadd ),qi(3,nqnum+nq0i+ nq0iadd ),wti(nqnum+nq0i+ nq0iadd ))
00446     wt0=1d0
00447 !! Set irreducible k-point flag. irr=1 for (irreducible point) flag, otherwise =0.
00448 !! irr(iq)=1 for irreducible qq(:,iq), iq=1,nqnum
00449     call q0irre(qibz,nqibz,qq,wt0,nqnum,symops,ngrp, qi,nqi,wti,plat,.true.,0,irr)
00450 !! nqnum is the finally obtained number of q points.
00451     allocate(ngpn(nqnum), ngcn(nqnum))
00452     if(debug) write(6,*) ' --- q vector in 1st BZ + Q0P shift. ngp ---'
00453     imx=0
00454     imxc=0
00455     do iq = 1, nqnum
00456         q = qq(1:3,iq)
00457         qxx=q
00458         if(iq0pin==1) then !use qxx on regular mesh points if q is on regular+Q0P(true).
00459             do iqbz=1,nqbz
00460                 do i=1,nq0itru+ nq0iadd ! nq0itru/=nq0i for anyq=F nov2015
00461                     if(sum(abs(qbz(1:3,iqbz)-dq_+ q0i(:,i)-qxx))<tolq) then
00462                         qxx=qbz(1:3,iqbz)
00463                     exit
00464                 endif
00465             enddo
00466         enddo
00467     endif
00468     ngpn(iq)=1
00469 !! get ngpn. # of G vector for |q+G| < QpGcut_psi
00470     call getgv2(alat,plat,qlat, qxx, qpgcut_psi,1,ngpn(iq),imxl1) !imxl1 !nov2015
00471     imx0=imxl1(1,1)
00472     if(imx0>imx) imx=imx0
00473     ngcn(iq)=1
00474 !! get ngcn. # ofG vector for |q+G| < QpGcut_cou
00475     call getgv2(alat,plat,qlat, qxx, qpgcut_cou,1,ngcn(iq),imxl1) !imxl1 to avoid warning.
00476     imx0c=imxl1(1,1)
00477     if(imx0c>imxc) imxc=imx0c
00478     if(verbose()>150)write(6,'(3f12.5,3x,2i4)') q ,ngpn(iq) !,ngcn(iq,iq00)
00479     if(verbose()>150)write(6,'(3f12.5,3x,2i4)') q ,ngcn(iq) !,ngcn(iq,iq00)
00480     enddo
00481
00482 !! Get G vectors and Write q+G vectors -----
00483     ngpmx = maxval(ngpn)
00484     ngcmx = maxval(ngcn)
00485     write(ifiqg ) nqnum,ngpmx,qpgcut_psi,nqbz,nqi,imx,nqibz
00486     write(ifiqgc) nqnum,ngcmx,qpgcut_cou,nqbz,nqi,imxc
00487 !! :nqi: The number of irreducible points (including irr. of offset points). irr=1.
00488 !! :: We calcualte eigenfunction and Vxc for these points.
00489 !! :nqnum: total number of q points.
00490 !! :imx: to allocate ngvecprev as follows.
00491     print *, ' number of irrecucible points nqi=',nqi
00492     print *, ' imx nqnum=',imx,nqnum
00493     write(6,*) ' --- Max number of G for psi =',ngpmx
00494     write(6,*) ' --- Max number of G for Cou =',ngcmx
00495     allocate( ngvecprev(-imx:imx,-imx:imx,-imx:imx) ) !inverse mapping table for ngvecp (psi)
00496     allocate( ngveccrev(-imxc:imxc,-imxc:imxc,-imxc:imxc) ) !inverse mapping table for ngvecc (cou)
00497     ngvecprev=9999
00498     ngveccrev=9999
00499     do iq = 1, nqnum
00500         q = qq(1:3,iq)

```

```

00501      qxx=q
00502      q0pf=''
00503      do iqbz=1,nqbz !use qxx on regular mesh points if q is on regular+Q0P(true).
00504      do i=1,nq0ittrue+ nq0iadd !nq0ittrue/=nq0i for anyq=F nov2015
00505          if(sum(abs(qbz(1:3,iqbz)-dq_+ q0i(:,i)-qxx))<tolq) then
00506              if(sum(abs(q0i(:,i)-qxx))<tolq) then
00507                  q0pf=' <--Q0P ' ! offset Gamma points
00508              else
00509                  q0pf=' <--Q0P+R' ! offset Gamma points-shifted nov2015
00510              endif
00511              if(iq0pin==1) then
00512                  qxx=qbz(1:3,iqbz)
00513              endif
00514              exit
00515          endif
00516      enddo
00517      enddo
00518      ngp = ngpn(iq)
00519      ngc = ngcn(iq)
00520      write(6,(' iq=',i8,' q=',3f9.5,' ngp ngc= ',2i6,' irr.=' ,i2,a)") !irr=1 is irreducible k points.
00521      write(6,(' iq=',i8,' q=',3f17.13,' ngp ngc= ',2i6,' irr.=' ,i2,a)") !irr=1 is irreducible k
points.
00522      &      iq, q, ngp, ngc, irr(iq),trim(q0pf)
00523      allocate( ngvecp(3,max(ngp,1)), ngvecc(3,max(ngc,1)) )
00524      call getgv2(alat,plat,qlat, qxx, qpgcut_psi, 2, ngp, ngvecp) ! for eigenfunctions (psi)
00525      call getgv2(alat,plat,qlat, qxx, qpgcut_cou, 2, ngc, ngvecc) ! for Coulomb (cou)
00526      write (ifiqq) q, ngp, irr(iq)
00527      do ig = 1,ngp
00528          nnn3 = ngvecp(1:3, ig)
00529          ngvecprev( nnn3(1), nnn3(2),nnn3(3)) = ig
00530      enddo
00531      write (ifiqq) ngvecp,ngvecprev !ngvecprev is added on mar2012takao
00532      do ig = 1,ngc
00533          nnn3 = ngvecc(1:3, ig)
00534          ngveccrev( nnn3(1), nnn3(2),nnn3(3)) = ig
00535      enddo
00536      write (ifiqgc) q, ngc
00537      write (ifiqgc) ngvecc,ngveccrev
00538      deallocate(ngvecp,ngvecc)
00539      enddo
00540      deallocate(ngpn,ngcn,ngvecprev,ngveccrev)
00541      if(iq0pin==1) deallocate(ngvect)
00542      if(debug) print *,'--- end of mkqg2 ---'
00543      end

```

## 4.19 gwsrc/readqg.F File Reference

### Data Types

- module [m\\_readq0p](#)
- module [m\\_readqg](#)

*Return QGcou and QGpsi ==.*

### Functions/Subroutines

- subroutine [readppovl0](#) (q, ngc, ppovl)

#### 4.19.1 Function/Subroutine Documentation

4.19.1.1 subroutine [readppovl0](#) ( [real](#)(8), dimension(3), intent(in) *q*, integer, intent(in) *ngc*, [complex](#)(8), dimension(ngc,ngc), intent(out) *ppovl* )

Definition at line 34 of file [readqg.F](#).

Here is the caller graph for this function:

## 4.20 readqg.F

```
00001      module m_readq0p
00002      real(8),allocatable,protected:: wgt(:), wgt0(:,:),q0i(:,:) !,nx(:,:),nblocha(:)
00003      integer,protected:: nq0i,nq0iadd
00004      integer,protected,allocatable:: ixyz(:)
00005
00006      contains
00007      subroutine readq0p()
00008      implicit none
00009      integer:: neps,ifig0p,ifile_handle,i,nq0ix,ig0pin
00010      logical:: debug=.false.
00011 c      write(6,*) 'reading QOP'
00012      ifig0p=ifile_handle()
00013      open (ifig0p,file='Q0P')
00014      read (ifig0p,*) nq0i,ig0pin,nq0iadd
00015      allocate( wgt(1:nq0i),q0i(1:3,1:nq0i+nq0iadd),ixyz(nq0i+nq0iadd) )
00016      do i=1,nq0i+nq0iadd
00017          read (ifig0p, * ) wgt(i),q0i(1:3,i),ixyz(i)
00018 c      write (*, * ) wgt(i),q0i(1:3,i),ixyz(i)
00019      enddo
00020      nq0ix = nq0i
00021      do i=1,nq0i
00022          if(wgt(i)==0d0 ) then
00023              nq0ix = i-1
00024              exit
00025          endif
00026      enddo
00027      neps=nq0i-nq0ix ! number of zero weight q0p which are used for ixc=2 or 3 mode.
00028      write(6,*) ' num of zero weight q0p=',neps
00029      write(6, '(i3,f14.6,2x, 3f14.6)' ) (i, wgt(i),q0i(1:3,i),i=1,nq0i+nq0iadd)
00030      close(ifig0p)
00031      end subroutine
00032      end module
00033
00034      subroutine readppovl0(q,ngc,ppovl)
00035      implicit none
00036      integer, intent(in) :: ngc
00037      complex(8), intent(out) :: ppovl(ngc,ngc)
00038      real(8), intent(in) :: q(3)
00039      integer:: ngc_r,ippovl0,ifile_handle
00040      real(8):: qx(3),tolq=1d-8
00041      ippovl0=ifile_handle()
00042      open(ippovl0,file='PPOVL0',form='unformatted')
00043      do
00044          read(ippovl0) qx,ngc_r
00045          if(sum(abs(qx-q))<tolq) then
00046              if(ngc_r/=ngc) call rx( 'readin ppovl: ngc_r/=ngc' )
00047              read(ippovl0) ppovl
00048              exit
00049          endif
00050      enddo
00051      close(ippovl0)
00052      end
00053
00054 !> Return QGcou and QGpsi ===
00055      module m_readqg
00056      implicit none
00057      real(8),allocatable,private,target:: qc(:,:),qp(:,:)
00058      logical,private:: init(2)=.true.
00059      real(8),private:: QpGcut_cou, QpGcut_psi
00060      integer(4),private,target:: nqnumc,nqnump,ngcmx,ngpmx
00061      integer(4),allocatable,private:: ngvecp(:,:),ngp(:),ngvecc(:,:),ngc(:)
00062      integer,pointer,private::nqtt
00063      real(8),pointer,private::qtt(:,:)
00064      real(8),private:: epsd=1d-7
00065      integer,private,pointer:: nkey(:),kk1(:),kk2(:),kk3(:),iqkkk(:,:),:)
00066      integer,target,private :: nkeyp(3),nkeyc(3)
00067      integer,target,allocatable,private:: keyp(:,:),kk1p(:),kk2p(:),kk3p(:),iqkkkp(:,:),:)
00068      integer,target,allocatable,private:: keyc(:,:),kk1c(:),kk2c(:),kk3c(:),iqkkkc(:,:),:)
00069      real(8),private:: ginv_(3,3)
00070      contains
00071 c-----
00072      subroutine readngmx(key,ngmx)
00073 c- get ngcmx or mgpmx
00074      implicit none
00075      integer(4):: ngmx,ifigq=4052
00076      character(*) key
00077      if (key=='QGpsi') then
00078          open(ifigq, file='QGpsi',form='unformatted')
00079          read(ifigq) nqnump, ngpmx, qpgcut_psi
00080          ngmx=ngpmx
00081      elseif(key=='QGcou') then
00082          open(ifigq, file='QGcou',form='unformatted')
00083          read(ifigq) nqnumc, ngcmx, qpgcut_cou
00084          ngmx=ngcmx
```

```

00085     else
00086         call rx( "readngmx: key is not QGpsi QGcou")
00087     endif
00088     close(ifiqg)
00089     end subroutine
00090
00091 !> Get ngv and ngvec(3,ngv) for given qin(3)
00092 !! key=='QGcou' or 'QGpsi'
00093 subroutine readqg(key,qin,ginv, qu,ngv,ngvec)
00094     implicit none
00095     character*(*) , intent(in) :: key
00096     real(8), intent(in) :: qin(3),ginv(3,3)
00097     real(8), intent(out) :: qu(3)
00098     integer(4), intent(out) :: ngv, ngvec(3,*)
00099
00100     integer(4):: ifi, iq,verbose
00101     if (key=='QGpsi') then
00102         ifi=1
00103         if(verbose())>=80 write (6, "(' readqg psi: qin=',3f8.3,i5)") qin
00104     elseif(key=='QGcou') then
00105         ifi=2
00106         if(verbose())>=80 write (6, "(' readqg cou: qin=',3f8.3,i5)") qin
00107     else
00108         call rx( "readqg: wrongkey")
00109     endif
00110     if(init(ifi)) then
00111         call init_readqg(ifi,ginv)
00112         init(ifi)=.false.
00113     endif
00114     if(verbose())>=40 write(6,*)'end of init_readqg'
00115     call iqindx2qg(qin,ifi, iq,qu)
00116     if(ifi==1) then
00117         ngv = ngp(iq)
00118         ngvec(1:3,1:ngv) = ngvecp(1:3,1:ngv,iq)
00119         return
00120     elseif(ifi==2) then
00121         ngv = ngc(iq)
00122         ngvec(1:3,1:ngv) = ngvecc(1:3,1:ngv,iq)
00123         return
00124     endif
00125     call rx( "readqg: can not find QGpsi or QPcou for given q")
00126     end subroutine readqg
00127
00128 !> Get ngv
00129 !! key=='QGcou' or 'QGpsi'
00130 subroutine readqg0(key,qin,ginv, qu,ngv)
00131     implicit none
00132     character*(*) , intent(in) :: key
00133     integer(4), intent(out) :: ngv
00134     real(8), intent(in):: qin(3),ginv(3,3)
00135     real(8), intent(out):: qu(3)
00136
00137     integer(4):: ifi, iq,verbose
00138     if (key=='QGpsi') then
00139         ifi=1
00140         if(verbose())>=80 write (6, "('readqg0 psi: qin=',3f8.3,i5)") qin
00141     elseif(key=='QGcou') then
00142         ifi=2
00143         if(verbose())>=80 write (6, "('readqg0 cou: qin=',3f8.3,i5)") qin
00144     else
00145         call rx( "readqg: wrongkey")
00146     endif
00147     if(init(ifi)) then
00148         call init_readqg(ifi,ginv)
00149         init(ifi)=.false.
00150     endif
00151     call iqindx2qg(qin,ifi, iq,qu)
00152     if(ifi==1) then
00153         ngv = ngp(iq)
00154         if(verbose())>=80 write(6,*)'ngp=',ngv
00155     elseif(ifi==2) then
00156         ngv = ngc(iq)
00157         if(verbose())>=80 write(6,*)'ngc=',ngv
00158     endif
00159     return
00160     call rx( "readqg0: can not find QGpsi or QPcou for given q")
00161     end subroutine
00162
00163 !> initialization. readin QGpsi or QGcou.
00164 subroutine init_readqg(ifi,ginv)
00165     implicit none
00166     integer(4), intent(in) :: ifi
00167     real(8), intent(in) :: ginv(3,3)
00168
00169     integer(4):: ifiqg,iq,verbose
00170     real(8)::qq(3)
00171     real(8),allocatable:: qxx(:, :)

```

```

00172 integer:: isig,i,ix,kkk,kkk3(3),ikl,ik2,ik3,ik
00173 integer,allocatable:: ieord(:),key(:,:)
00174 ginv_=ginv
00175 write(6,*)' init_readqg ifi=',ifi
00176 ifiqg=4052
00177 if(ifi==1) then
00178   open(ifiqg, file='QGpsi',form='unformatted')
00179   read(ifiqg) nqnump, ngpmx, qpgcut_psi
00180   if(verbose()>49)
00181     & write (6,('init_readqg ngnumc ngcmx QpGcut_psi=',2i5,f8.3))
00182     & nqnump, ngpmx, qpgcut_psi
00183   allocate(ngvecp(3,ngpmx,nqnump),qp(3,nqnump),ngp(nqnump))
00184   do iq=1, nqnump
00185     read (ifiqg) qp(1:3,iq), ngp(iq)
00186     read (ifiqg) ngvecp(1:3,1:ngp(iq),iq)
00187     if(verbose()>40)
00188       & write (6,('init_readqg psi qp ngp =',3f8.3,i5)) qp(1:3,iq),ngp(iq)
00189   enddo
00190 elseif(ifi==2) then
00191   open(ifiqg, file='QGcou',form='unformatted')
00192   read(ifiqg) nqnumc, ngcmx, qpgcut_cou
00193 c   write (6,('init_readqg ngnumc ngcmx QpGcut_cou=',2i5,f8.3))
00194 c   & nqnumc, ngcmx, QpGcut_cou
00195   allocate(ngvecc(3,ngcmx,nqnumc),qc(3,nqnumc),ngc(nqnumc))
00196   do iq=1, nqnumc
00197     read(ifiqg) qc(1:3,iq), ngc(iq)
00198 c   if(verbose()>40) write (6,('init_readqg cou qc ngc =',3f8.3,i5)) qc(1:3,iq), ngc(iq)
00199     write (6,('init_readqg cou qc ngc =',3f8.3,i5)) qc(1:3,iq), ngc(iq)
00200     read (ifiqg) ngvecc(1:3,1:ngc(iq),iq)
00201   enddo
00202 endif
00203 close(ifiqg)
00204
00205 !! === mapping of qtt ===
00206 !! nkey, kkl,kk2,kk3, iqkkk are to get iqindx.
00207 !! q --> call rangedq(matmul(ginv,q), qx) ---> n= (qx+0.5*epsd)/epsd
00208 !! ---> ikl,ik2,ik3= tabkk(kkk,iqk,nkey) ---> iqkkk(ikl,ik2,ik3)
00209 if(ifi==1) then
00210   nqtt => nqnump
00211   qtt => qp
00212   nkey => nkeyp
00213 elseif(ifi==2) then
00214   nqtt => nqnumc
00215   qtt => qc
00216   nkey => nkeyc
00217 endif
00218 !! followings are the same as codes in read eigen.F
00219 allocate(ieord(nqtt))
00220 allocate(key(3,0:nqtt),qxx(3,nqtt))
00221 key(:,0)=0 !dummy
00222 key=-99999
00223 c print *,'ginv_',ginv_
00224 do iq=1,nqtt
00225   call rangedq(matmul(ginv_,qtt(:,iq)), qxx(:,iq))
00226 ccccccccccccc
00227 c   ix=1
00228 c   print *,' xxxx ix qxx=',ix,iq,qtt(ix,iq),matmul(ginv_,qtt(:,iq))
00229 ccccccccccccc
00230 enddo
00231 !! get key and nkey for each ix.
00232 do ix =1,3
00233 ccccccccccccc
00234 c   do i=1,nqtt
00235 c   print *,' ix qxx=',ix,i,qtt(ix,i),qxx(ix,i)
00236 c   enddo
00237 ccccccccccccc
00238   call sortea(qxx(ix,:),ieord,nqtt,isig)
00239 ccccccccccccc
00240 c   do i=1,nqtt
00241 c   print *,' ix qxx=',ix,i,qxx(ix,ieord(i))
00242 c   enddo
00243 ccccccccccccc
00244   ik=0
00245   do i=1,nqtt
00246     kkk=(qxx(ix,ieord(i))+0.5d0*epsd)/epsd !kkk is digitized by 1/epsd
00247     if(i==1.or.key(ix,ik)<kkk) then
00248       ik=ik+1
00249       key(ix,ik) = kkk
00250     elseif (key(ix,ik)>kkk) then
00251       write(6,*)ix, ik,i, key(ix,ik), qxx(ix,ieord(i))
00252       call rx('iqindx: bug not sorted well')
00253     endif
00254   enddo
00255   nkey(ix)=ik
00256 enddo
00257 deallocate(ieord)
00258 !! key is reallocated. inverse mattping, iqkkk

```



```

00259     if(ifi==1) then
00260         allocate( kk1p(nkey(1)),kk2p(nkey(2)),kk3p(nkey(3)) )
00261         allocate( iqkkkp(nkey(1),nkey(2),nkey(3)) )
00262         iqkkk => iqkkkp
00263         kk1 =>kk1p
00264         kk2 =>kk2p
00265         kk3 =>kk3p
00266     elseif(ifi==2) then
00267         allocate( kk1c(nkey(1)),kk2c(nkey(2)),kk3c(nkey(3)) )
00268         allocate( iqkkkc(nkey(1),nkey(2),nkey(3)) )
00269         iqkkk => iqkkkc
00270         kk1 =>kk1c
00271         kk2 =>kk2c
00272         kk3 =>kk3c
00273     endif
00274
00275     kk1(:) = key(1,1:nkey(1))
00276     kk2(:) = key(2,1:nkey(2))
00277     kk3(:) = key(3,1:nkey(3))
00278     deallocate(key)
00279 c     write(6,*)' ifi init_qqq nqtt=',ifi,nqtt
00280 c     write(6,*)'kkk3=',kkk3
00281 c     write(6,*)'nkey=',nkey
00282 c     write(6,*)'kk1=',kk1
00283     do i=1,nqtt
00284         kkk3= (qxx(:,i)+0.5*epsd)/epsd !kkk is digitized by 1/epsd
00285         call tabkk(kkk3(1), kk1,nkey(1), ik1)
00286         call tabkk(kkk3(2), kk2,nkey(2), ik2)
00287         call tabkk(kkk3(3), kk3,nkey(3), ik3)
00288         iqkkk(ik1,ik2,ik3)=i
00289 cccccccccccccccc
00290 c     write(6,*)' ik1,ik2,ik3 i=',ik1,ik2,ik3,i
00291 cccccccccccccccc
00292     enddo
00293     deallocate(qxx)
00294     end subroutine init_readqg
00295 !! ---
00296     subroutine tabkk(kkin, kktable,n, nout)
00297     integer:: nout,n, kkin, kktable(n),i,mm,i1,i2
00298     i1=1
00299     i2=n
00300     if(kkin==kktable(1)) then
00301         nout=1
00302         return
00303     elseif(kkin==kktable(n)) then
00304         nout=n
00305         return
00306     endif
00307     do i=1,n
00308         mm=(i1+i2)/2
00309         if(kkin==kktable(mm)) then
00310             nout=mm
00311             return
00312         elseif(kkin>kktable(mm)) then
00313             i1=mm
00314         else
00315             i2=mm
00316         endif
00317     enddo
00318     write(6,*) i1,i2,kkin
00319     write(6,*) kktable(i1),kktable(i2)
00320     call rx( 'takk: error')
00321     end subroutine
00322
00323 c$$$c--- release to save memory area.
00324 c$$$     subroutine releaseqg_notusednow(key)
00325 c$$$     implicit none
00326 c$$$     character*(*) key
00327 c$$$     integer(4):: ifi
00328 c$$$     if (key=='QGpsi') then
00329 c$$$         ifi=1
00330 c$$$         deallocate(qp,ngvecp)
00331 c$$$     elseif(key=='QGcou') then
00332 c$$$         ifi=2
00333 c$$$         deallocate(qc,ngvecc)
00334 c$$$     else
00335 c$$$         stop "releaseqg: in readQGcou"
00336 c$$$     endif
00337 c$$$     init(ifi)=.false.
00338 c$$$     end subroutine
00339 !!-----
00340
00341 !> Find index as q=qg(:,iq) with modulo of primitive vector.
00342 !! ginv is the inverse of plat (primitive translation vector).
00343 !! Use kk1,kk2,kk3,nkey(1:3),iqkkk to get iqindx.
00344     subroutine iqindx2qg(q,ifi, iqindx,qu)
00345     implicit none

```

```

00346     integer, intent(in):: ifi
00347     integer, intent(out):: iqindx
00348     real(8), intent(in) :: q(3)
00349     real(8), intent(out) :: qu(3)
00350
00351     integer:: i_out, iq,iqx ,kkk3(3),ik1,ik2,ik3
00352     real(8):: qx(3),qzz(3)
00353     logical::debug=.false.
00354     if(ifi==1) then
00355 c         nqtt => nqnump
00356         qtt => qp
00357         nkey => nkeyp
00358         iqkkk => iqkkkp
00359         kk1 =>kk1p
00360         kk2 =>kk2p
00361         kk3 =>kk3p
00362     elseif(ifi==2) then
00363 c         nqtt => nqnumc
00364         qtt => qc
00365         nkey => nkeyc
00366         iqkkk => iqkkkc
00367         kk1 =>kk1c
00368         kk2 =>kk2c
00369         kk3 =>kk3c
00370     endif
00371 c     if(debug) write(*, "(' iqindx2_: q=' ,3f20.15)") q
00372 c     write(*, "(' iqindx2_: q=' ,3f20.15)") q
00373     call rangedq(matmul(ginv_,q), qzz)
00374     kkk3 = (qzz+0.5*epsd)/epsd
00375 c     write(6,*) 'kkk3=' ,kkk3
00376 c     write(6,*) 'kk1,nkey1' ,kk1,nkey(1)
00377 c     write(6,*) 'kk2,nkey2' ,kk2,nkey(2)
00378 c     write(6,*) 'kk3,nkey3' ,kk3,nkey(3)
00379     call tabkk(kkk3(1), kk1,nkey(1), ik1)
00380     call tabkk(kkk3(2), kk2,nkey(2), ik2)
00381     call tabkk(kkk3(3), kk3,nkey(3), ik3)
00382 c     write(6,*) ' iklik2ik3=' ,ik1,ik2,ik3
00383     iqindx = iqkkk(ik1,ik2,ik3)
00384 c     write(6,*) 'iqindx=' ,iqindx
00385     qu = qtt(:,iqindx)
00386 c     write(6,*) 'iqindx=' ,iqindx
00387 c     write(6,*) 'qu=' ,qu
00388     end subroutine
00389
00390 !> mini-sort routine.
00391 subroutine sortea(ea,ieaord,n,isig)
00392     real(8), intent(in) :: ea(n)
00393     integer(4), intent(inout) :: ieaord(n)
00394     integer, intent(in) :: n
00395     integer, intent(out) :: isig
00396     integer :: ix,i
00397     isig = 1
00398     do i = 1,n
00399         ieaord(i) = i
00400     enddo
00401     do ix= 2,n
00402         do i=ix,2,-1
00403             if( ea(ieaord(i-1)) >ea(ieaord(i)) ) then
00404                 call iswap(ieaord(i-1),ieaord(i))
00405                 isig= -isig
00406                 cycle
00407             endif
00408             exit
00409         enddo
00410     enddo
00411 end subroutine
00412 subroutine iswap(i,j)
00413     implicit none
00414     integer,intent(inout) :: i, j
00415     integer:: iwork
00416     iwork = j
00417     j = i
00418     i = iwork
00419 end subroutine
00420 end module m_readqg

```

## 4.21 gwsrsc/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, dwdummy, ecore, nlmtot, nqibz, nqbz, nctot,

### 4.21.1 Function/Subroutine Documentation

4.21.1.1 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,nqibz), intent(in) *qbz*, real(8), dimension(nqibz), intent(in) *wk*, integer(4), dimension(nqibz), 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) *dwdummy*, real(8), dimension(nctot) *ecore*, integer *nlmtot*, integer *nqibz*, integer *nqbz*, integer *nctot* )

$z1p(j,t') = S[i=1, nbloch] \langle \psi(q,t') | \psi(q-rk,t) B(rk,i) \rangle v(k)(i,j)$  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:

### 4.22 `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 dwdummy,ecore,
00008      d nlmtot,nqibz,nqbz,nctot,
00009 c      d nl,nnc,nclass,natom,mdimx,
00010      d nbloch,ngroup, nw_i,nw ,niw,niwx,nq, !nlmnm,
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 wkml,lxklm,
00017      i dwplot,
00018      o zsec,coh,exx)
00019      use m_readqg
00020      use m_readeigen,only: readeval
00021      use m_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 sigma_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  $SEc(q,t,t') = \langle \psi(q,t) | SEc | \psi(q,t') \rangle$ 
00044      c  $SEc(r,r';w) = (i/2\pi) \langle [w'=-inf,inf] G(r,r';w+w') Wc(r,r';w') \rangle$ 
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           =  $\langle [r1,r2] v(|r-r1|) X(r1,r2;w) v(|r2-r'|) \rangle$ 
00053 c  $W(r,r';w) = \langle [r','] ei(r,r';w) v(|r''-r'|) \rangle$ 
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            $\langle \psi(q,t) | \psi(q-k,n) B(k,i) \rangle \langle B(k,j) \psi(q-k,n) | \psi(q,t') \rangle$ 
00065 c            $(i/2\pi) \langle [w'=-inf,inf] Wc(k,w')(i,j) / (w'+w-e(q-k,n)-i\delta) \rangle$ 
00066 c
00067 c           +  $S[k=FBZ] S[n=unocc] S[i,j=1,nbloch]$ 
00068 c            $\langle \psi(q,t) | \psi(q-k,n) B(k,i) \rangle \langle B(k,j) \psi(q-k,n) | \psi(q,t') \rangle$ 
00069 c            $(i/2\pi) \langle [w'=-inf,inf] Wc(k,w')(i,j) / (w'+w-e(q-k,n)+i\delta) \rangle$ 
00070
00071 c the analytic structure of  $GWc$  for  $w \leq ef$ 
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  $w > ef$ 
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  $(i/2\pi) \langle [w'=-inf,inf] Wc(k,w')(i,j) / (w'+w-e(q-k,n)) \rangle$ 
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  $wint = (i/2\pi) \langle [w'=-inf,inf] Wc(w') / (w+w'-e) \rangle$ 
00116 c
00117 c  $w' \Rightarrow iw'$ ,  $w'$  is now real
00118 c  $wint = - (1/\pi) \langle [w'=0,inf] Wc(iw') (w-e) / \{(w-e)^2 + w'^2\} \rangle$ 
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  $wint = - (1/\pi) \langle [x=0,1] Wc(iw') (w-e)x^2 / \{(w-e)^2 + w'^2\} \rangle$ 
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  $wint = - (1/\pi) \langle [x=0,1] \{ Wc(iw') - Wc(0)\exp(-a^2 w'^2) \}$ 
00128 c            $\star (w-e) / \{(w-e)^2 + w'^2\} x^2 \rangle$ 
00129 c           -  $(1/2) Wc(0) \operatorname{sgn}(w-e) \exp(a^2 (w-e)^2) \operatorname{erfc}(a|w-e|)$ 
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 c
00140 c-----
00141 c contribution from the poles of  $G$ :  $SEc(pole)$ 
00142 c-----
00143 c
00144 c for  $w \leq ef$ 
00145 c  $SEc(pole) = - S[k=FBZ] S[n=occ] S[i,j=1,nbloch]$ 
00146 c  $\langle \psi(q,t) | \psi(q-k,n) B(k,i) \rangle \langle B(k,j) \psi(q-k,n) | \psi(q,t') \rangle$ 
00147 c  $Wc(k,e(q-k,n)-w)(i,j) \theta(e(q-k,n)-w)$ 
00148 c
00149 c for  $w > ef$ 
00150 c  $SEc(pole) = + S[k=FBZ] S[n=unocc] S[i,j=1,nbloch]$ 
00151 c  $\langle \psi(q,t) | \psi(q-k,n) B(k,i) \rangle \langle B(k,j) \psi(q-k,n) | \psi(q,t') \rangle$ 
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 c
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 c
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  $rot \cdot R = R' + T$ 
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 =  $\langle \phi(RLn) | \phi(RLn') \rangle B(R,i)$ 
00178 c pdb =  $\langle \phi(RLn) | \phi(RLn') \rangle B(R,i)$ 
00179 c dpb =  $\langle \phi(RLn) | \phi(RLn') \rangle B(R,i)$ 
00180 c ddb =  $\langle \phi(RLn) | \phi(RLn') \rangle B(R,i)$ 
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 deltax = energy mesh in  $SEc(q,t,w)$  ---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,cbhq = real and imaginary part of  $hb(q)$ 
00199 c rbkq,cbkq = real and imaginary part of  $b(q-k)$ 
00200 c rhbkq,cbbkq = real and imaginary part of  $hb(q-k)$ 
00201 c b is the eigenvector of the LMT0-Hamiltonian
00202 c ekq = eigenvalues at  $q-k$ 
00203 c rmel,cmel = real and imaginary part of
00204 c  $\langle \psi(q,t') | \psi(q-k,t) B(k,R,i) \rangle$ 
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 LMT0 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  $secq(t) = \langle \psi(q,t) | SEc | \psi(q,t) \rangle$ 

```

```

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 dwdummy,ecore,
00232      d nlmt0,nqibz,nqbz,nctot,
00233 c      d nl,nnc,nclass,natom,mdimx,
00234      d nbloch,ngroup, 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 wkml,lxklm
00241 c      i pomatr, qrr,nnr,nor,nmx,nomx,nkpo,
00242 c      i invg,!il,in,im,nn_, lx,nx_,nxx_,dwplot !ppbrd, !cgr,,nlnm
00243
00244      integer :: ntq, nqbz,nqibz,ngroup,nq,nw,niw, !natom,
00245      & nband, nlmt0, nq0i,nctot,mbytes,iwksize,nlmtobnd,nstate,nstatex,
00246      & irot, iqisp,ikpisp,isp,nsp, !nlnmx, !iq, idxk,
00247 c      & iwr1,iwr2,iwr3,iwr4,iwcl,iwc2,iwc3,iwc4
00248      & ip, it,itp,          !ifcphi, ! ifrb,ifcb,ifrbh,ifchb,
00249 c      i iiclass,          !mdim(*),
00250      i ifrcw,ifrcwi,          !iindxk,
00251      & ifvcfpout,ndummy1,ndummy2,kx,kr,ngc,ngb,nbloch, !nl,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,ngroup),
00256      & nstar(nqibz),irkip(nqibz,ngroup,nq),kount(nqibz,nq)
00257 c
00258      real(8) :: q(3),qbas(3*3),ginv(3*3), !tr(3,natom), !,tiat(3,natom,ngroup)
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      & dwdummy,wt,wexx,www,exx,exxq,wfacx2,weavx2,wex
00265 c      complex(8) :: zsec(-1:1,ntq,nq)
00266 c      real(8) :: shtw
00267 c      ! This sht 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,ngroup),qk(3), !qfbz(3),
00274      & qdiff(3),add(3),symgg(3,3,ngroup),symope(3,3), !qbasinv(3,3), det,
00275      & qxx(3),q0i(1:3,1:nq0i),shtv(3),alat,ecore(nctot), !shtvg(3,ngroup),
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      & wlp(:,:,:),w2p(:,:,:),w3p(:,:)
00289      complex(8),allocatable :: zlp(:,:,:),vcoul(:,:),vcoult(:,:)
00290      integer:: invrot,invr
00291 c      integer:: invg(ngroup),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 ccccccccccccccc
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 ccccccczzzzzzzz
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(nlmt0,nband), cphikq(nlmt0,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) :: ebm,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,n_index_qbz)
00337
00338 c integer(4)::icore(*),ncore(*),imdim(*) !,iclass(*),nlmv(*),nlmc(*),
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),qibz_kr(3)
00345 logical :: onlyimagaxis
00346
00347 logical ::zwz3mode
00348
00349
00350 real(8):: ua_,expa_(niw),ua2,freqw,freqwl,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 cccccccccccccccccccccccccccccc
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):: ibl,ib2,ifix
00362 cccccccccccccccccccccccccccccccccccccccccc
00363 logical ::iww2=.true., oncew
00364
00365
00366 C...
00367 c logical::smbasis
00368 integer(4):: iclose,ixx,ixq !nn,no,ifpommat,
00369 c complex(8),allocatable:: pomat(:,:)
00370 real(8):: q_r(3)
00371 c integer(4):: nnm, nomx,nkpo, nnr(nkpo),nor(nkpo)
00372 c complex(8):: pomatr(nnm,nomx,nkpo)
00373 c real(8):: qrr(3,nkpo)
00374
00375 real(8):: elxx,ehxx,ekxx,efxx
00376 integer(4):: ixsm, iwm,iir,nwxi, itini,itend, nrm
00377 real(8) :: ffr(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):: wk1m((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,tolq=1d-8
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$$$!! --- gauss_img : interpolation gaussian for W(i \omega).
00412 c$$$ 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 Timeeversal 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 = nlmtot*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 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 separated into WVR.ID and WVI.ID files.
00490 !! NOTE: vcoud and zcousq are in module m_zmelt.
00491 c         if(kx<=ngibz) qxx=qibz_k
00492 c         if(kx>ngibz ) qxx=q0i(:,kx-ngibz)
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))<tolq) goto 1133
00503         enddo
00504         if(sum(abs(qvv-qxx))>tolq) 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= 0 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 <= ngibz) 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-ngibz,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 <= ngibz) 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-ngibz,irot)==0d0 ) cycle
00555 c         endif
00556             write(*, "('ip,kx irot=',3i5, ' out of',2i4)") ip,kx,irot, igend,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 ebm1: to set how many bands of <i|sigma|j> do you calculate.
00571 !! nbmx2 ebm2: 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,nbmxe,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 zmel= 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<= ngibz) then
00595             wtt = wk(kr) ! wtx = ld0
00596         else
00597             wtt = wk(1)*wgt0(kx-ngibz,irot) ! wtx = wgt0(kx-ngibz,irot)
00598             if(abs(wk(1)-ld0/dble(ngbz))>ld-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> zlp(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= wk(1)* fpi*sqrt(fpi) /wk(kx)
00633 c                             write(6,*)'wk(1) vc=',wk(1),vc
00634                         else
00635                             vc= vcoul(ivc)
00636                         endif
00637                         w3p(it,itp) = w3p(it,itp)+ vc * abs(zmeltt(it,itp,ivc))*2
00638                     continue
00639                 continue
00640             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 weights wfac for valence by esmr
00661      do it = nctot+1, nctot+nbmax
00662          wfac = wfacx(-ld99, 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] zlp(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(-ld99, 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$$$c$ifdef USE_GEMM_FOR_SUM
00693      c$$$c      if(verbose())>39)write(*,*)'info: USE GEMM FOR SUM (zmel=zmel*ppovlz), in sxcf_fal2.F'
00694      c$$$c      allocate( zmelt (ngb, nstate) )
00695      c$$$c      do itp=1,ntp0
00696      c$$$c      zmelt = dcmplx(rmelt(:,itp),-cmelt(:,itp))
00697      c$$$c      call zgemm('C','N',ngb,nstate,ngb,(ld0,0d0),
00698      c$$$c      .      ppovlz,ngb,zmelt,ngb,(0d0,0d0),zmel(1,1,itp),ngb)
00699      c$$$c      enddo
00700      c$$$c      deallocate(zmelt)
00701      c$$$c$else
00702      c$$$c      do itp=1,ntp0
00703      c$$$c      do it=1,nstate
00704      c$$$c      zmel(:,it,itp) = matmul(zmel(:,it,itp),dconjg(ppovlz(:,it)))
00705      c$$$c      enddo
00706      c$$$c      enddo
00707      c$$$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      c      nrec=ix
00733      c      if(debug) write(6,*)' wvr nrec kx nw nw_i ix=',nrec,kx,nw,nw_i,ix
00734      c      read(ifrcw,rec=nrec) zw ! direct access read Wc(0) = W(0) - v
00735      c      zwz0=0d0
00736      c      !! this loop looks complicated but just in order to get zwz0=zmel*zwz0*zmel
00737      c      !! Is this really efficient???
00738      c      CCC!$OMP parallel do private(itp,it,igb2,zz2)
00739      c      do itp=1,ntp0
00740      c      do it=1,nstate
00741      c      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 = drealm(zwz0)
00751 c COH term test ----- The sum of the all states for zwz00 gives the delta function.
00752                if(cohrest) then
00753                    do itp = 1,ntq
00754                        coh(itp,ip) = coh(itp,ip)
00755    &                    + .5d0*wtt*sum(drealm(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 ",$)')
00768                if(verbose()>50) call cputid(0)
00769
00770                zwz=0d0
00771                do ix = 1,nx                !*npm                ! imaginary frequency w'-loop
00772                    nrec= ix
00773                    if(debug) write(6,*)' wvi nrec=',nrec
00774                    read(ifrcwi,rec=nrec) zw ! Readin W-v on imag axis
00775                    if(npm==1) then !then zwz is real so, we can use mode c2.
00776                        do itp= 1,ntp0
00777                            do it = 1,nstate
00778                                ppp=0d0
00779                                do igb2 = 2,ngb
00780                                    zz2 = sum( dconjg(zmel(1:igb2-1,it,itp))*zw(1:igb2-1,igb2) )
00781 ! only take real part
00782                                    ppp = ppp + drealm(zz2*zmel(igb2,it,itp)) * 2d0
00783    &                                    + dconjg(zmel(igb2,it,itp))*zw(igb2,igb2)*zmel(igb2,it,itp)
00784                                    enddo                !igb2
00785                                    zwz(ix,it,itp) = ppp +
00786    &                                    dconjg(zmel(1,it,itp))*zw(1,1)*zmel(1,it,itp)
00787                                    enddo                !it
00788                                    enddo                !itp
00789                                else
00790                                    !we need to use mode2 because zwz is not real now.
00791                                    call matzwz( zw(1:ngb,1:ngb), zmel, ntp0,nstate,ngb,
00792    o                                    zwz(ix,1:nstate,1:ntp0))
00793                                endif
00794                                if(debug) write(6,*)' sumzw=',sum(abs(zw))
00795                                enddo                !ix
00796                            if(verbose()>50) write(*, '( "xxx:6.1 before matzwz in ix cycle ",$)')
00797                            if(verbose()>50) call cputid(0)
00798                            if(debug) write(6,*)' sumzmel=',ngb, nstate, ntp0,sum(abs(real(zmel))),sum(abs(imag(zmel)))
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 ccccccccccccccccccccccccccccccccccc
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 ccccccccccccccccccccccccccccccccccc
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 ix 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         ixsmn = 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                     esmr = esmr
00929                     if(it<=nctot) esmr = 0d0
00930                     wfac = wfacx2(omg,ef, ekc(it),esmr)
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                         ix = iwp ! ix = iwp+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(ix>ixsmx .and. omg<=ef ) ixsmx = ix
00946 c                 if(ix>ixsmn .and. omg> ef ) ixsmn = ix
00947                     if(ix>ixsmx .and. omg>=ef ) ixsmx = ix
00948                     if(ix>ixsmn .and. omg< ef ) ixsmn = ix
00949                     wexx = we
00950                     if(ixs+1 > nw) then
00951                         write (*,*) ' nw_i ixsmn',nw_i, ixsmn
00952 c                 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                     nw = max(ixsmx+1,ixsmn+1)
00963                 else
00964                     nwxi = -ixsmn-1
00965                     nw = ixsmx+1
00966                 endif
00967                 if (nw > nw ) then
00968                     call rx( ' sxcf nw 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 nw nw=' ,nwxi,nw,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:nw))
00999             do ix = nwxi,nw ! 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                 &
01009                 ngb, ebb)
01010             do ii=1,ngb
01011                 if(abs(ebb(ii))>1d-8.and.ebb(ii)>0) then
01012                     write(6,"'iiilxxx: iw ii eb=',2i4,d13.5") ix,ii,ebb(ii)
01013                 else
01014                     write(6,"'iiil: iw ii eb=',2i4,d13.5") ix,ii,ebb(ii)
01015                 endif
01016             enddo
01017             deallocate(ebb)
01018         endif
01019         deallocate(zw)
01020     else
01021         nstatex= max(ntp0,nt_max)
01022         if(allocated(zwz)) deallocate(zwz)
01023         allocate( zwz(nwx1:nwx,1:nstatex,ntp0) )
01024         do ix = nwx1,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
01030                 zwz(ix,1:nstatex,1:ntp0))
01031             ! zmel (ngb, nstate, ntp0)
01032             enddo
01033             deallocate(zmel)
01034             deallocate(zw)
01035         endif
01036     c-----
01037     if(screen) then
01038         if(zwz3mode) call rx( ' this mode is not implimented')
01039         do ix = nw_i,nwx
01040             zwz(ix, :, :)=zwz(ix, :, :)- zwz00
01041         enddo
01042         deallocate(zwz00)
01043     endif
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                 esmr = esmr
01069                 if(it<=nctot) esmr = 0d0
01070                 wfac = wfacx2(omg,ef, ekc(it),esmr)
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                 ix=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 nwxi=',ixs+1,nwxi
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))*zwz(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 xxxxxxxxxxxxxx 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(wlp)) deallocate(wlp)
01167         if (allocated(w2p)) deallocate(w2p)
01168         if (allocated(w3p)) deallocate(w3p)
01169         if (allocated(zlp)) deallocate(wlp)
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

```

## 4.23 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)

#### 4.23.1 Function/Subroutine Documentation

**4.23.1.1** 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	<i>nctot</i>	Determine indexes of a range for calculation. It is better to clean this up...
in	<i>nw_i</i>	Determine indexes of a range for calculation. It is better to clean this up...
in	<i>nw</i>	Determine indexes of a range for calculation. It is better to clean this up...
in	<i>nstate</i>	Determine indexes of a range for calculation. It is better to clean this up...
in	<i>nt0p</i>	Determine indexes of a range for calculation. It is better to clean this up...
in	<i>nt0m</i>	Determine indexes of a range for calculation. It is better to clean this up...
in	<i>ntq</i>	Determine indexes of a range for calculation. It is better to clean this up...
in	<i>nband</i>	Determine indexes of a range for calculation. It is better to clean this up...
in	<i>ntqxx</i>	Determine indexes of a range for calculation. It is better to clean this up...

Definition at line 1314 of file [sxcf\\_fal2.sc.F](#).

Here is the caller graph for this function:

## 4.24 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,dwdummy,
00009 i ecore,
00010 i nlmto,nqibz,nqgz,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 wkml,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 m_keyvalue,only   : getkeyvalue
00024      use m_zmel,only      : get_zmelt,
00025      i ppovlz,
00026      o zmel,zmeltt
00027      implicit none
00028 !!> \brief
00029 !! Calculate 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 because it includes e_i.
00037 !!     i and j are band indexes
00038 !! \param coh dummy
00039 !! \param screen dummy
00040 !!
00041 !! \remark
00042 !!
00043 !! \verbatim
00044 !! Jan2013: eftrue is added.
00045 !!   ef=eftrue(true fermi energy) for valence exchange and correlation mode.
00046 !!   but ef is not the true fermi energy for core-exchange mode.
00047 !!
00048 !! Jan2006
00049 !!   "zsec from im-axis integral part" had been symmetrized as
00050 !!   &      wtt*.5d0*( sum(zwzi(:,itp,itpp))+ !S_{ij}(e_i)
00051 !!   &      dconjg( sum(zwzi(:,itpp,itp)) ) ) !S_{ji}*(e_j)= S_{ij}(e_j)
00052 !!   However, I now do it just the 1st term.
00053 !!   &      wtt* sum(zwzi(:,itp,itpp)) !S_{ij}(e_i)
00054 !!   This is OK because the symmetrization is in hqpe.sc.F
00055 !!   Now zsec given in this routine is simply written as <i|Re[S](e_i)|j>.
00056 !!   ( In the version until Jan2006 (fpgw032f8), only the im-axis part was symmetrized.
00057 !!   But it was not necessary from the begining because it was done in hqpe.sc.F
00058 !!
00059 !!   (Be careful as for the difference between
00060 !!   <i|Re[S](e_i)|j> and transpose(dconjg(<i|Re[S](e_i)|j>)).
00061 !!   ---because e_i is included.
00062 !!   The symmetrization (hermitian) procedure is included in hqpe.sc.F
00063 !!
00064 !!   NOTE: matrix element is given by "call get_zmelt". It returns  zmelt or zmeltt.
00065 !!
00066 !! jobsw switch
00067 !!   1-5 scGW mode.
00068 !!   diag+@EF      jobsw==1 SE_nn'(ef)+delta_nn'(SE_nn(e_n)-SE_nn(ef))
00069 !!   xxx modeB (Not Available now) jobsw==2 SE_nn'((e_n+e_n')/2) !we need to recover comment out for
00070 !!   jobsw==2, and test.
00071 !!   mode A      jobsw==3 (SE_nn'(e_n)+SE_nn'(e_n'))/2 (Usually used in QSGW).
00072 !!   @Ef      jobsw==4 SE_nn'(ef)
00073 !!   diagonly   jobsw==5 delta_nn' SE_nn(e_n) (not efficient memoryuse; but we don't use this mode so
00074 !!   often).
00075 !!
00076 !! Output file in hsf0 should contain hermitean part of SE
00077 !!   ( hermitean of SE_nn'(e_n) means SE_n'n(e_n')^* )
00078 !!   we use that zwz(itp,itpp)=dconjg( zwz(itpp,itp) )
00079 !! Caution! npm=2 is not examined enough...
00080 !!
00081 !! Calculate the exchange part and the correlated part of self-energy.
00082 !! T.Kotani started development after the analysis of F.Aryasetiawan's LMTO-ASA-GW.
00083 !! We still use some of his ideas in this code.
00084 !!
00085 !! See paper
00086 !! [1]T. Kotani and M. van Schilfgaarde, ??Quasiparticle self-consistent GW method:
00087 !!   A basis for the independent-particle approximation, Phys. Rev. B, vol. 76, no. 16, p.
00088 !!   165106[24pages], Oct. 2007.
00089 !! [2]T. Kotani, Quasiparticle Self-Consistent GW Method Based on the Augmented Plane-Wave
00090 !!   and Muffin-Tin Orbital Method, J. Phys. Soc. Jpn., vol. 83, no. 9, p. 094711 [11 Pages], Sep. 2014.
00091 !! -----
00092 !! Omega integral for SEC
00093 !!   The integral path is deformed along the imaginary-axis, but together with contribution of poles.
00094 !!   See Fig.1 and around in Ref.[1].
00095 !!
00096 !! ---Integration along imaginary axis.---
00097 !!   ( Current version for it, wintzsg_npm, do not assume time-reversal when npm=2.)

```

```

00096 !! Integration along the imaginary axis: -----
00097 !! (Here is a memo by F.Aryasetiawan.)
00098 !! (i/2pi) < [w'=-inf,inf] Wc(k,w')(i,j)/(w'+w-e(q-k,n) >
00099 !! Gaussian integral along the imaginary axis.
00100 !! transform: x = 1/(1+w')
00101 !! this leads to a denser mesh in w' around 0 for equal mesh x
00102 !! which is desirable since Wc and the lorentzian are peaked around w'=0
00103 !! wint = - (1/pi) < [x=0,1] Wc(iw') (w-e)x^2/((w-e)^2 + w'^2) >
00104 !!
00105 !! the integrand is peaked around w'=0 or x=1 when w=e
00106 !! to handel the problem, add and subtract the singular part as follows:
00107 !! wint = - (1/pi) < [x=0,1] { Wc(iw') - Wc(0)exp(-a^2 w'^2) }
00108 !! * (w-e)/((w-e)^2 + w'^2)x^2 >
00109 !! - (1/2) Wc(0) sgn(w-e) exp(a^2 (w-e)^2) erfc(a|w-e|)
00110 !!
00111 !! the second term of the integral can be done analytically, which
00112 !! results in the last term a is some constant
00113 !!
00114 !! when w = e, (1/pi) (w-e)/((w-e)^2 + w'^2) ==> delta(w') and
00115 !! the integral becomes -Wc(0)/2
00116 !! this together with the contribution from the pole of G (s.u.)
00117 !! gives the so called static screened exchange -Wc(0)
00118 !!
00119 !! ---Integration along real axis (contribution from the poles of G: SEc(pole))
00120 !! See Eq.(34),(55), and (58) and around in Ref.[1]. We now use Gaussian Smearing.
00121 !! -----
00122 !! \endverbatim
00123 !! \verbatim
00124 !!
00125 !! -----
00126 !! q =qip(:,iq) = q-vector in SEC(q,t).
00127 !! itq = states t at q
00128 !! ntq = no. states t
00129 !! eq = eigenvalues at q
00130 !! ef = fermi level in Rydberg
00131 !! WVI, WVR: direct access files for W. along im axis (WVI) or along real axis (WVR)
00132 !! freq_r(nw_i:nw) = frequencies along real axis. freq_r(0)=0d0
00133 !!
00134 !! qbas = base reciprocal lattice vectors
00135 !! ginv = inverse of qbas s. indxrk.f
00136 !!
00137 !! wk = weight for each k-point in the FBZ
00138 !! qbz = k-points in the 1st BZ
00139 !!
00140 !! wx = weights at gaussian points x between (0,1)
00141 !! ua_ = constant in exp(-ua^2 w'^2) s. wint.f
00142 !! expa = exp(-ua^2 w'^2) s. wint.f
00143 !!
00144 !! irkip(k,R,nq) = gives index in the FBZ with k{IBZ, R=rotation
00145 !!
00146 !! nqibz = number of k-points in the irreducible BZ
00147 !! nqbz = full BZ
00148 !! natom = number of atoms
00149 !! nctot = total no. of allowed core states
00150 !! nbloch = total number of Bloch basis functions
00151 !! nlmtot = total number of MTO+lo basis functions
00152 !! ngrp = no. group elements (rotation matrices)
00153 !! niw = no. frequencies along the imaginary axis
00154 !! nw_i:nw = no. frequencies along the real axis. nw_i=0 or -nw.
00155 !! zsec(itp,itpp,iq)> = <psi(itp,q(:,iq)) |SEc| psi(iq,q(:,iq)>
00156 !!
00157 !! -----
00158 !! \endverbatim
00159 !! integer:: dummy4doxygen
00160 !!
00161 ! input variables
00162 logical, intent(in) :: exchange,screen,cohtest
00163 integer, intent(in) :: ntq,nqbz,nqibz,ngrp,nq,niw !,natom
00164 integer, intent(in) :: nband,nlmtot,nq0i,nctot,isp,nsp !,mdim(*) !,nlmnm
00165 integer, intent(in) :: ifvcfpout,nbloch,nblochpmx !nl,nnc, nclass
00166 integer, intent(in) :: itq(ntq) !,nstar(nqibz) !miat(natom,ngrp),mdimx,
00167 integer, intent(in) :: irkip(nqibz,ngrp,nq),nrkip(nqibz,ngrp,nq)
00168 integer, intent(in) :: kount(nqibz,nq),ngpmx,ngcmx,ifexsp,jobsw
00169 integer, intent(in) :: nbmx(2) !,nlmnmv(*),nlmnc(*),!iclass(*),icore(*)
00170 integer, intent(in) :: nstbz(nqbz) !,nomx !,nkpo,nnmx,imdim(*)ncore(*),
00171 integer, intent(in) :: lxklm !,invg(ngrp) !nnr(nkpo),nor(nkpo),
00172 c integer, intent(in) :: il(*),in(*),im(*),nn_,lx(*),nx_(*),nxx_ !,nlmnm(*)
00173 real(8), intent(in) :: wgt0(nq0i,ngrp),symgg(3,3,ngrp)
00174 real(8), intent(in) :: q0i(1:3,1:nq0i),alat,ecore(nctot) !shtvg(3,ngrp),
00175 real(8), intent(in) :: qbas(3,3),ginv(3,3)
00176 real(8), intent(in) :: wk(nqbz),qibz(3,nqibz) !tiat(3,natom,ngrp),
00177 real(8), intent(in) :: qbz(3,nqbz),freqx(niw),wx(niw),ef,esmr,dwdummy
00178 real(8), intent(in) :: ebm(2),wklm((lxklm+1)**2) !,qrr(3,nkpo)
00179 real(8), intent(in) :: qip(3,nq),eftrue
00180
00181 c integer,intent(in):: iwini,iwend
00182 c real(8),optional::exx

```

```

00183
00184 ! output variables
00185 c      real(8),intent(in),optional:: freqsig(iwini:iwend)
00186      integer, intent(in) :: nbandmx(nq)
00187      complex(8), intent(out),optional :: zsec(ntq,ntq,nq) , coh(ntq,nq)
00188 c      complex(8), intent(out),optional :: zsecd(iwini:iwend,ntq,nq)
00189
00190 ! local variables
00191 c      complex(8) :: zsecx(ntq,ntq,nq)
00192 c      complex(8), intent(in) :: pomatr(nnmx,nomx,nkpo)
00193 c$$$      logical :: ua_auto !fixed to be .false.
00194 c      real(8)::ppbrd ( 0:nl-1, nn_, 0:nl-1,nn_, 0:2*(nl-1),1:nxx_, 1:nsp*nclass)
00195
00196      integer :: ifrcw,ifrcwi
00197      logical :: initp=.true.
00198      real(8),allocatable:: vcoud(:)
00199
00200      integer :: ip, it, itp, i, ix, kx, irot, kr
00201      integer :: nt0p, nt0m,nstate , nbmax, ntqxx !iatomp(natom),
00202      integer :: nt,nw,ixs,iw,ivc,ifvcoud,ngb0
00203      integer :: nprecx,mrecl,ifwd,nrot,nwp,nw_i,ierr
00204      integer :: nstatetot,igini,igend, ngb,ngc !nbcut,
00205      integer :: invr,nbmxe,ia,nn,ntp0,no,itpp,nrec,npm,itini,itend
00206      integer :: iwp,nwxi,nwx,iir, igb1,igb2,ix0,iii
00207
00208      real(8) :: tpi, ekc(nctot+nband),ekq(nband), det, q(3),ua_
00209      real(8) :: expa_(niw), qxx(3), symope(3,3),shv(3) !tr(3,natom),
00210      real(8) :: efp,efm,wtt,wfac,we,esmrq,gbasinv(3,3)
00211      real(8) :: qvv(3),pi,fpi,eq(nband),omega(ntq),quu(3),freqw,ratio
00212      real(8) :: qibz_k(3),qibz_kr(3),ddw,vc,omega0,omg
00213
00214      complex(8) :: cphiq(nlmt0,nband), cphikq(nlmt0,nband)
00215      complex(8) :: zwzs0,zz2,zwz3(3)
00216
00217 ! local arrays
00218      real(8),intent(in) :: freq_r(nw_i:nw)
00219      real(8),allocatable :: drealzzzmel(:, :, :), dimagzzzmel(:, :, :),uaa(:, :)
00220      complex(8),allocatable :: vcoul(:, :, :),w3p(:, :, :)
00221      complex(8),allocatable :: zzzmel(:, :, :),zw ( :, : )
00222      complex(8),allocatable :: zwz(:, :, :), zwz0(:, :, :),zwzi(:, :, : )
00223      complex(8),allocatable :: zwix(:, :, :),zwzix(:, :, :),zmell1(:) !,expikt(:)
00224      complex(8), allocatable :: zmell1(:, :, :), zw3(:, :, :),zw3x(:, : )
00225      complex(8), allocatable :: zwz4(:, :, :),zwz44(:, :, :),pomatr(:, :), zwzs(:)
00226      complex(8),allocatable :: ppovl(:, :),zcousq(:, : )
00227      complex(8),allocatable :: zlr(:, :),z2r(:, :),w3pi(:, : )
00228
00229      real(8), parameter :: wfaccut=1d-8
00230      complex(8), parameter :: img=(0d0,1d0)
00231
00232 ! external function
00233 c      logical :: smbasis
00234 c      logical :: test_symmetric_W
00235 c      logical :: GaussSmear !fixed to be T
00236 c      logical :: newaniso !fixed to be T
00237 c      integer :: bzcass !fixed to be 1
00238      character(5) :: charnum5
00239      integer :: iopen,iclose
00240      integer :: invrot
00241      complex(8) :: wintzsg_npm !wintzav,
00242      integer :: nocc
00243      real(8) :: wfaccx
00244      real(8) :: wfacc2
00245      real(8) :: weavx2
00246      complex(8) :: alagr3z
00247      complex(8) :: alagr3z2
00248
00249      integer:: ndummy1,ndummy2,nlmtobnd,nt0
00250      real(8):: wexx
00251 c      complex(8),allocatable :: zlp(:, :, :),vcoult(:, : )
00252      logical :: debug, debugp,debug2=.false.
00253 c      logical :: gass !external
00254 c      real(8):: wgtq0p
00255      integer::verbose,ififr,ifile_handle
00256      real(8):: ua2_(niw),freqw1
00257      integer :: istate, nt_max !nbcutc,nbcutin,
00258      real(8):: q_r(3),qk(3),omegat
00259      logical:: oncew, onceww, eibz4sig, timemix
00260
00261      integer,allocatable:: ixss(:, :),iirx(:)
00262      real(8),allocatable:: we_(:, :),wfacc_(:, : )
00263      complex(8),allocatable:: zw3av(:, :),zmellw(:, :, : )
00264      integer:: noccx
00265      real(8)::polinta
00266      logical,allocatable:: ititpskip(:, : )
00267
00268      logical:: tote=.false.
00269      logical:: hermitianw

```

```

00270
00271     real(8),allocatable:: wcorehole(:,:)
00272     logical:: corehole
00273     integer:: ifcorehole
00274     real(8):: tolq=1d-8
00275 c     real(8),allocatable:: ppb(:)
00276 c     allocate( ppb(nlnmx*nlnmx*mdimx*nclass))
00277
00278 c     real(8)::exxq
00279
00280 c-----
00281 c!TIME0_0000
00282 c     write(6,*)'sxcf_fal3_scz'
00283     timemix=.false.
00284     pi = 4d0*datan(1d0)
00285     fpi = 4d0*pi
00286     debug=.false.
00287     if(verbose())>=90) debug=.true.
00288
00289 c     corehole=.true.
00290     corehole=.false.
00291
00292 !! core-hole
00293     if(corehole) then
00294         ifcorehole=ifile_handle()
00295         open(ifcorehole,file='CoreHole')
00296         if(allocated(wcorehole)) deallocate(wcorehole)
00297         allocate(wcorehole(nctot,nspp))
00298         do it=1,nctot
00299             read(ifcorehole,*) wcorehole(it,1:nspp)
00300         enddo
00301         close(ifcorehole)
00302         write(*,*) 'end of reading CoreHole'
00303     endif
00304
00305     if(.not.exchange) then
00306         ifwd = iopen('WV.d',1,-1,0)
00307         read (ifwd,*) nprecx,mrecl
00308         ifwd = iclose('WV.d')
00309 !! gauss_img : interpolation gaussian for W(i \omega).
00310         call getkeyvalue("GWinput","gauss_img",ua_,default=1d0)
00311         if(debug) write(6,*) ' sxcf_fal3_scz: Gausssmear=T'
00312         do ix = 1,niw
00313             !! Energy mesh; along im axis.
00314             freqw = (1d0 - freqx(ix))/ freqx(ix)
00315             expa_(ix) = exp(-(ua_*freqw)**2)
00316         enddo
00317         npm = 1
00318         if(nw_i/=0) npm = 2
00319         ! npm=1 Timeveversal case
00320         ! npm=2 No TimeReversal case. Need negative energy part of W(omega)
00321     endif
00322
00323 c     call getkeyvalue("GWinput","nbcutlow_sig",nbcut, default=0 )
00324 c     nbcut=nctot+nbcut
00325     tpi = 8d0*datan(1d0)
00326     if(nctot/=0) ekc(1:nctot)= ecore(1:nctot) ! core
00327     nlmtobnd = nlmt*nband
00328     nstatetot = nctot + nband
00329
00330 != ip loop to spedify external q ==
00331     do 1001 ip = 1,nq
00332         if(sum(irkip(:,ip))/=0) cycle ! next ip
00333         write (6,*) ip, ' out of ',nq, ' k-points(extrnal q) '
00334         q(1:3)= qip(1:3,ip)
00335         call readeval(q,isp,eq)
00336         do i = 1,ntq
00337             omega(i) = eq(itq(i))
00338         enddo
00339
00340 !! we only consider bzcane()==1
00341     if(abs(sum(qibz(:,1)**2))/=0d0) call rx( ' sxcf assumes 1st qibz/=0 ')
00342     if(abs(sum( qbz(:,1)**2))/=0d0) call rx( ' sxcf assumes 1st qbz /=0 ')
00343
00344 !! NOTE total number of
00345     kx loop(do 1100) and irot loop (do 1000) makes all the k mesh points.
00346     When iqini=1 (Gamma point), we use effective W(q=0) defined in the paper.
00347     iqini=1
00348     iqend=nqibz
00349     !no sum for offset-Gamma points.
00350     do 1100 kx = iqini,iqend
00351         if(sum(irkip(kx,ip))/=0) cycle ! next kx
00352     !TIME0_01000
00353     write(6,*) ' ### do 1100 start kx=',kx, ' from ',iqini, ' through', iqend
00354 c     if( kx <= nqibz ) then
00355 c         qibz_k= qibz(:,kx)
00356 c     else
00357 c         qibz_k= 0d0
00358 c     endif
00359 c     if(timemix) call timeshow("1111 k-cycle")

```

```

00357      call readqg0('QGcou',qibz_k,ginv, quu,ngc)
00358      ngb = nbloch + ngc
00359      if(debug) write(6,*) ' sxcf: ngb=',ngb,nbloch
00360
00361 !! ===Readin diagonalized Coulomb interaction===
00362 !! Vcoud file is sequential file Vcoulomb matrix for qibz_k.
00363 !! A possible choice for paralellization is "Vcoud.ID" files where ID=kx
00364 !! Voud file is written in hvccfp0.m.F.
00365 !! For correlation, W-v is read instead of Vcoud file (ifrcw,ifrcwi for WVR and WVI)
00366 !! These can be also separated into WVR.ID and WVI.ID files.
00367 !! NOTE: vcoud and zcousq are in module m_zmelt.
00368      qxx=qibz_k
00369 c      if(kx<=nqibz) qxx=qibz_k
00370 c      if(kx>nqibz) qxx=q0i(:,kx-nqibz)
00371      ifvcoud = iopen('Vcoud.'//charnum5(kx),0,0,0)
00372      do
00373          read(ifvcoud) ngb0
00374          read(ifvcoud) qvv
00375          if(allocated(vcoud)) deallocate(vcoud)
00376          allocate( zcousq(ngb0,ngb0),vcoud(ngb0) )
00377          read(ifvcoud) vcoud
00378          read(ifvcoud) zcousq
00379          if(sum(abs(qvv-qxx))<tolq) goto 1133
00380      enddo
00381      if(sum(abs(qvv-qxx))>tolq) then
00382          write(6,*)'qvv =' ,qvv
00383          write(6,*)'qxx=' ,qxx,kx
00384          call rx( 'sxcf_fal2: qvv/=qibz(:,kx) hvcc is not consistent')
00385      endif
00386 1133      continue
00387      if( ngb0/=ngb ) then !sanity check
00388          write(6,*)' qxx ngb0 ngb=' ,qxx,ngb0,ngb
00389          call rx( 'hsfp0.m.f:ngb0/=ngb')
00390      endif
00391 !! ppovlz is used in get_zmel
00392 !! <I|v|J>= \sum_mu ppovl*zcousq(:,mu) v^mu (Zcousq^*(:,mu) ppovl)
00393 !! zmel contains O^-1<I|J>^-1 factor. zmel(phi phi J)= <phi phi|I> O^-1_IJ
00394 !! ppovlz= O Zcousq
00395 !! (V_IJ - vcoud_mu O_IJ) Zcousq(J, mu)=0, where Z is normalized with O_IJ.
00396      allocate(ppovl(ngc,ngc),ppovlz(ngb,ngb))
00397      call readppovl0(qibz_k,ngc,ppovl)
00398      ppovlz(1:nbloch,:) = zcousq(1:nbloch,:)
00399      ppovlz(nbloch+1:nbloch+ngc,:)=matmul(ppovl,zcousq(nbloch+1:nbloch+ngc,:))
00400 c      write(6,*)'sumcheck ppovlz 00000 =' ,sum(abs(ppovlz(:,:)))
00401      deallocate(zcousq,ppovl)
00402 !! === open WVR,WVI ===
00403      if(.not.exchange) then
00404          ifrcw = iopen('WVR.'//charnum5(kx),0,-1,mrecl)
00405          ifrcwi = iopen('WVI.'//charnum5(kx),0,-1,mrecl)
00406      endif
00407      nrot=0
00408      do irot = 1,ngrp
00409 c          if( kx <= nqibz) then
00410              kr = irkip(kx,irot,ip) ! index for rotated kr in the FBZ
00411              if(kr=0) cycle ! next irot
00412              qbz_kr= qbz(:,kr)
00413 c          else
00414              kr=-99999 !for sanity check
00415 c          qbz_kr= 0d0
00416 c          if( wgt0(kx-nqibz,irot)==0d0 ) cycle ! next irot
00417 c          endif
00418          nrot=nrot+1
00419      enddo
00420 !TIME1_01000 " :BeforDo1000"
00421
00422
00423 !! === loop 1000 over rotations irot ===
00424      do 1000 irot = 1,ngrp
00425 c          if( kx <= nqibz) then
00426              kr = irkip(kx,irot,ip) ! index for rotated kr in the FBZ
00427              if(kr=0) cycle
00428              qbz_kr= qbz(:,kr)
00429 c          else
00430              kr=-99999 !for sanity check
00431 c          qbz_kr= 0d0
00432 c          if( wgt0(kx-nqibz,irot)==0d0 ) cycle
00433 c          endif
00434
00435 !TIME0_1010
00436 !! no. occupied (core+valence) and unoccupied states at q-rk
00437      qk = q - qbz_kr
00438      call readeval(qk, isp, ekq)
00439      ekc(nctot+1:nctot+nband) = ekq(1:nband)
00440      nt0 = nocc(ekc,ef,.true.,nstatetot)
00441      ddw= .5d0
00442 c      if(GaussSmear()) ddw= 10d0
00443      ddw= 10d0

```

```

00444      efp= ef+ddw*esmr
00445      efm= ef-ddw*esmr
00446 c      nt0p = nocc (ekc,efp,.true.,nstatetot)
00447 c      nt0m = nocc (ekc,efm,.true.,nstatetot)
00448      nt0p = nocc(ekq,efp,.true.,nstatetot)+ nctot
00449      nt0m = nocc(ekq,efm,.true.,nstatetot)+ nctot
00450 !! nbmx1 ebmx1: to set how many bands of <i|sigma|j> do you calculate.
00451 !! nbmx2 ebmx2: to restrict num of bands of G to calculate G \times W
00452
00453      if(exchange) then
00454          nbmax = nt0p-nctot
00455      else
00456          nbmax = nband
00457          nbmx = nocc(ekc,ebmx(2),.true.,nstatetot)-nctot
00458          nbmax = min(nband,nbm(2),nbmx)
00459          if(ini) then
00460              write(6,*)' nbmax=',nbmax
00461              ini=.false.
00462          endif
00463      endif
00464 c$$$!! ntqxx is number of bands for <i|sigma|j>.
00465 c$$$      ntqxx = nocc (omega-eftrue,ebmx(1),.true.,ntq)
00466 c$$$!bug -ef is added jan2013
00467 c$$$!previous version do not give wrong results, but inefficient.
00468 c$$$      ntqxx = min(ntqxx, nbm(1))
00469 c$$$      if(ntqxx<nband) then
00470 c$$$          do i=ntqxx,1,-1 !reduce ntqxx when band tops are degenerated. !sep2012
00471 c$$$              if(omega(i+1)-omega(i)<ld-2) then
00472 c$$$                  ntqxx=i-1
00473 c$$$              else
00474 c$$$                  exit
00475 c$$$              endif
00476 c$$$          enddo
00477 c$$$      endif
00478 c$$$      nbndmx(ip)=ntqxx !number of bands to be calculated Sep2012.
00479
00480      ntqxx = nbndmx(ip) !mar2015
00481 if(debug) write(6,*)' sxcf: nbmax nctot nt0p =',nbmax,nctot,nt0p
00482 nstate = nctot + nbmax ! = nstate for the case of correlation
00483
00484 !! Get matrix element zmel= rmelt + img*cmelt, defined in m_zmel.F--
00485 c      if(debug) write(6,*)'zzBBB ppovlz =',sum(abs(ppovlz(:,:))),kx,irot
00486      if(allocated(zmel)) deallocate(zmel)
00487      if(allocated(zmeltt)) deallocate(zmeltt)
00488 !TIME1_1010 "Beforeget_zmel"
00489 ! this return zmeltt (for exchange), or zmel (for correlation)
00490 !TIME0_1088
00491      call get_zmelt(exchange,q,kx,qibz_k,irot,qbz_kr,kr,isp,
00492          &      ngc,ngb,nbmax,ntqxx,nctot,ncc=0)
00493      if(debug) write(6,*)' end of get_zmelt'
00494 !TIME1_1088 "get_zmelt"
00495
00496 c$$$!! ccccccccc START: old version, instead of get_zmelt ccccccccc
00497 c$$$      call readcphi(q, nlmt,isp, quu, cphikq)
00498 c$$$      if(debug) write(6,*)' sxcf: 2'
00499 c$$$      do      it = 1,ntq
00500 c$$$          itp      = itq(it)
00501 c$$$          cphiq(1:nlmt,it) = cphikq(1:nlmt,itp)
00502 c$$$          write(*,*)'svvvv ',it, itp, sum(cphiq(:,it))
00503 c$$$      enddo
00504 c$$$          write(*,*)'srrrrr lc',sum(cphiq(:,1:ntq)),ntq
00505 c$$$
00506 c$$$      call dinv33(qbas,0,qbasinv,det)
00507 c$$$      if(debug) write(6,*)' sxcf: 1'
00508 c$$$      if(allocated(expikt)) deallocate(expikt)
00509 c$$$      allocate(expikt(natom))
00510 c$$$$cccccccccccccccccccccccccccccccccccc
00511 c$$$!!      rotate atomic positions invrot*R = R' + T
00512 c$$$          invr = invrot (irot,invg,ngrp)
00513 c$$$          tr = tiat(:,invr)
00514 c$$$          iatomp= miat(:,invr)
00515 c$$$          symope= symgg(:,irot)
00516 c$$$          shtv = matmul(symope,shtvg(:,invr))
00517 c$$$!TIME1 "before ppbafp_v2"
00518 c$$$!TIME0
00519 c$$$
00520 c$$$!! -- ppb= <Phi(SLn,r) Phi(SLn',r) B(S,i,Rr)>
00521 c$$$c      call ppbafp_v2 (irot,ngrp,isp,nsp,
00522 c$$$c          i,il,in,im,nl,nl,w(i_mml),
00523 c$$$c          d,nl,nn_,ncl,nl,nl,nl,
00524 c$$$c          i,mdimx,lx,nx_,nx_, !Bloch wave
00525 c$$$c          i,cgr, nl-1, !rotated CG
00526 c$$$c          i,ppbrd, !radial integrals
00527 c$$$c          o,ppb)
00528 c$$$      ppb = ppbir(:,irot,isp)
00529 c$$$!! qk = q-rk. rk is inside 1st BZ, not restricted to the irreducible BZ
00530 c$$$      qk = q - qbz_kr !qbz(:,kr)

```



```

00531 c$$$      call readcphi(qk, nlmto,isp, quu, cphikq)
00532 c$$$!TIME1 "before expikt"
00533 c$$$!TIME0
00534 c$$$
00535 c$$$!! =====
00536 c$$$!!      matrix elements <psi(q,t') | psi(q-rk,t) B(rk,R,i)>
00537 c$$$!!      including the phase factor exp(ik.T)
00538 c$$$!!      B(rot*k,r) = B(k,invrot*r)
00539 c$$$!! =====
00540 c$$$!! phase factors expikt(ia) is for exp(ik.T(R))
00541 c$$$      do ia = 1,natom
00542 c$$$      expikt(ia) = exp(img*tpi* sum(qibz_k*tr(:,ia)) )
00543 c$$$      end do
00544 c$$$!! matrix elements
00545 c$$$!!      core
00546 c$$$      nt = nctot + nbmax ! = nstate for the case of correlation
00547 c$$$      allocate( zzzmel(nbloch,nt,ntqxx))
00548 c$$$      call psich_v2 (icore,ncore,ntqxx,iclass,
00549 c$$$      i      dreal(expikt(1:natom)),dimag(expikt(1:natom)),
00550 c$$$      i      cphiq,
00551 c$$$      i      ppb,
00552 c$$$      i      nlnmv,nlnmc,mdim,
00553 c$$$      i      imdim,iatomp,
00554 c$$$      d      mdimx,nlmto,nbloch,nlnmx,nt,ntqxx,natom,nclass,
00555 c$$$      d      nl,nnc,
00556 c$$$      o      zzzmel)
00557 c$$$      if(debug) write(6,*) ' sxcf_fal2sc: goto psi2bc1'
00558 c$$$cccccccccccccccccccccccccccccccccccccccccccccccc
00559 c$$$      write(*,*)'srrrrr 1',sum(cphikq(1:nlmto,1:ntq))
00560 c$$$      write(*,*)'srrrrr 1',sum(cphiq(1:nlmto,1:ntq))
00561 c$$$      write(*,*)'srrrrr 1',sum(ppb)
00562 c$$$      write(*,*)'srrrrr 1',sum(expikt)
00563 c$$$      write(*,*)'srrrrr 1',sum(zzzmel)
00564 c$$$
00565 c$$$!!      valence
00566 c$$$      call psi2b_v2 (nbmax, ntqxx,iclass,
00567 c$$$      i      dreal(expikt(1:natom)),dimag(expikt(1:natom)),
00568 c$$$      i      cphikq, !occ q-rk nband
00569 c$$$      i      cphiq, !unocc q ntq
00570 c$$$      i      ppb,
00571 c$$$      i      nlnmv,nlnmc,mdim,nctot,
00572 c$$$      i      imdim,iatomp,
00573 c$$$      d      mdimx,nlmto,nbloch,nlnmx, nband, nt,ntqxx,
00574 c$$$      d      natom,nclass,
00575 c$$$      o      zzzmel)
00576 c$$$      if(verbose()>50) call timeshow("4 after psi2bc1")
00577 c$$$c      if(debug2) then
00578 c$$$      write(6,('sum of zmel abszmel=',4d23.16)) sum(zzzmel),sum(abs(zzzmel))
00579 c$$$c      end if
00580 c$$$!TIME1 "before psi2b_v2"
00581 c$$$!TIME0
00582 c$$$!! -- IPW part.
00583 c$$$      if(debug) write(6,*) ' sxcf_fall: goto drvmelp2 xxxl11'
00584 c$$$      allocate(drealzzzmel(nbloch,nt,ntqxx),dimagzzzmel(nbloch,nt,ntqxx))
00585 c$$$      drealzzzmel=dreal(zzzmel)
00586 c$$$      dimagzzzmel=dimag(zzzmel)
00587 c$$$      deallocate(zzzmel)
00588 c$$$      allocate( rmelt(ngb, nctot+nbmax, ntqxx), ! nstate= nctot+nband
00589 c$$$      &      cmelt(ngb, nctot+nbmax, ntqxx))
00590 c$$$      call drvmelp2( q, ntqxx, ! q in FBZ
00591 c$$$      i      q-qbz_kr, nbmax, ! q-rk
00592 c$$$      i      qibz_k, ! k in IBZ for mixed product basis. rk = symope(qibz_k)
00593 c$$$      i      isp,ginv,
00594 c$$$      i      ngc,ngcmx, ngpmx,nband,itq,
00595 c$$$      i      symope, shtv, qbas, qbasinv,qibz,qbz,nqbz,nqibz,
00596 c$$$      i      drealzzzmel, dimagzzzmel, nbloch, nt,nctot,
00597 c$$$      o      rmelt,cmelt)
00598 c$$$      if(debug) write(6,*) ' sxcf_fall: end of drvmelp2'
00599 c$$$      deallocate(drealzzzmel,dimagzzzmel)
00600 c$$$      if(verbose()>50) call timeshow("5 after drvmelp")
00601 c$$$      if(nbcut/=0.and.(.not.exchange)) then
00602 c$$$      do it= nctot+1,nctot+min(nbcut,nbmax)
00603 c$$$      rmelt(:, it,:) =0d0
00604 c$$$      cmelt(:, it,:) =0d0
00605 c$$$      enddo
00606 c$$$      endif
00607 c$$$      write(6,('sum of rmelt cmelt=',4d23.16))sum(rmelt),sum(cmelt)
00608 c$$$
00609 c$$$!TIME1 "after drvmelp2"
00610 c$$$!! NOTE:=====
00611 c$$$!! zmel = rmelt(igb(qbz_kr), iocc(q), iunocc(q-qbz_kr)) + i* cmelt
00612 c$$$!! iunocc: band index at target q.
00613 c$$$!! iocc: band index at intermediate vector qk = q - qbz_kr
00614 c$$$!! igb: index of mixed product basis at qbz_kr (or written as rk)
00615 c$$$!! igb=1,ngb
00616 c$$$!! ngb=nbloch+ngc ngb: # of mixed product basis
00617 c$$$!! nbloch: # of product basis (within MTs)

```



```

00618 c$$$!!          ngc: # of IPW for the Screened Coulomb interaction.
00619 c$$$!!          igc is for given
00620 c$$$!! See readgeig in drvmelp2.
00621 c$$$!! =====
00622 c$$$!! smbasis ---need to fix this
00623 c$$$$$$$          if(smbasis()) then !
00624 c$$$$$$$          ntp0= ntqxx
00625 c$$$$$$$          nn= nnr(kx)
00626 c$$$$$$$          no= nor(kx)
00627 c$$$$$$$          allocate( pomat(nn,no) )
00628 c$$$$$$$          pomat= pomatr(1:nn,1:no,kx)
00629 c$$$$$$$          if( sum(abs(qibz_k-qrr(:,kx)))>1d-10 .and.kx <= ngibz ) then
00630 c$$$$$$$          call rx( 'qibz/= qrr' )
00631 c$$$$$$$          endif
00632 c$$$$$$$          if(no /= ngb.and.kx <= ngibz) then
00633 c$$$$$$$!!      A bit sloppy check only for kx<ngibz because qibze is not supplied...
00634 c$$$$$$$          write(6,('( ' q  ngb  ',3d13.5,3i5)') qibz_k,ngb
00635 c$$$$$$$          write(6,('( ' q_r  nn no',3d13.5,3i5)') q_r,nn,no
00636 c$$$$$$$          call rx( 'x0kf_v2h: P0mat err no=ngb' )
00637 c$$$$$$$          endif
00638 c$$$$$$$          if(timemix) call timeshow("xxx2222 k-cycle")
00639 c$$$$$$$          ngb = nn      ! Renew ngb !!!
00640 c$$$$$$$          allocate( zmel (nn, nctot+nbmax, ntp0) )
00641 c$$$$$$$          call matm( pomat, dcmplx(rmelt,cmelt), zmel,
00642 c$$$$$$$          &          nn, no, (nctot+nbmax)*ntp0 )
00643 c$$$$$$$          deallocate(rmelt, cmelt)
00644 c$$$$$$$          allocate( rmelt(ngb, nctot+nbmax, ntp0), !ngb is reduced.
00645 c$$$$$$$          &          cmelt(ngb, nctot+nbmax, ntp0) )
00646 c$$$$$$$          rmelt = drealm(zmel)
00647 c$$$$$$$          cmelt = dimag(zmel)
00648 c$$$$$$$          deallocate(zmel,pomat)
00649 c$$$$$$$          else
00650 c$$$$$$$          nn=ngb
00651 c$$$$$$$          no=ngb
00652 c$$$$$$$          endif
00653 c$$$          nn=ngb
00654 c$$$          no=ngb
00655 c$$$          if( oncew() ) then
00656 c$$$          write(6,('(ngb nn no=',3i6)') ngb,nn,no
00657 c$$$          endif
00658 c$$$          if(timemix) call timeshow("22222 k-cycle")
00659 c$$$!! === End of zmel ; we now have matrix element zmel= rmelt + img* cmelt ===
00660 c$$$          if(allocated(zzzmel))deallocate(zzzmel) !rmel,cmel)
00661 c$$$          if(debug) write(6,*) ' sxcf: goto wtt'
00662 c$$$          if(debug) write(6,('(sum of rmelt cmelt=',4d23.16)')sum(rmelt),sum(cmelt))
00663 c$$$
00664 c$$$!! === End of zmel ; we now have matrix element zmel= rmelt + img* cmelt ===
00665 c$$$!! ccccccccc END: old version, instead of get_zmel ccccccccc
00666
00667
00668 !! --- wtt setcion ---
00669 c$$$          if(bzcase()==2)then
00670 c$$$          if(kx<=ngibz) then
00671 c$$$          wtt = wk(kr)
00672 c$$$          if(nstbz(kr)/=0) wtt = wk(kr)*(1d0-wgtq0p()/nstbz(kr))
00673 c$$$          elseif(kx>ngibz) then ! wtx= wgt0(kx-ngibz,irot)/dble(nqibz)
00674 c$$$          wtt= wgt0(kx-ngibz,irot)
00675 c$$$          endif
00676 c$$$          else
00677 c          if(kx<= ngibz) then ! wtx = 1d0
00678 c          wtt = wk(kr)
00679 c          else ! wtx = wgt0(kx-ngibz,irot)
00680 c          wtt = wk(1)*wgt0(kx-ngibz,irot)
00681 c          if(abs(wk(1)-1d0/dble(nqibz))>1d-10) call rx( 'sxcf:wk(1) inconsistent' )
00682 c          endif
00683 !!
00684 c          if(eibz4sig()) then
00685 c          wtt=wtt*nrkip(kx,irot,ip)
00686 c          endif
00687
00688 !!-----
00689 !! --- exchange section ---
00690 !!-----
00691 c          if(exchange) then !At the bottom of this block, cycle do 1000 irot.
00692 !! We use the matrix elements zmeltt. Now given by "call get_zmelt"
00693 !!
00694 c need to check following comments ----
00695 c S[i,j=1,nbloch] <psi(q,t) |psi(q-rk,n) B(rk,i)>
00696 c v(k)(i,j) <B(rk,j) psi(q-rk,n) |psi(q,t')>
00697 c
00698 c > zlp(j,n,t) = S[i=1,nbloch] <psi(q,t) | psi(q-rk,n) B(rk,i)> v(k)(i,j)
00699 c
00700 c --- screened exchange case
00701 c if(screen) then
00702 c allocate( zw (nblochpmx,nblochpmx) )
00703 c ix = 1
00704 c ! write(*,*)(kx-2)*(nw_w+1)+ix

```

```

00705 c      read(ifrcw,rec=((kx-2)*nw+ix)) zw ! Readin W(0) - v      !sf 22May02
00706 c      !nw is number of frequency points in general mesh: freq_r(nw), freq_r(1)=0
00707 c      vcoul = vcoul + zw(1:ngb,1:ngb) !c screen test
00708 c      deallocate(zw)
00709 c      endif
00710 !TIME0_0130
00711      vc = vcoud(1)      ! save vcoud(1)
00712      if (kx == iqini) vcoud(1) = wk(1)* fpi*sqrt(fpi) /wk(kx)
00713      allocate(zlr(ntqxx,ngb),z2r(ntqxx,ngb),w3pi(ntqxx,ntqxx))
00714      allocate(w3p(nctot+nbmax,ntqxx,ntqxx))
00715      do it = 1, nctot+nbmax
00716          do ivc = 1, ngb
00717              do itp = 1, ntqxx
00718                  zlr(itp,ivc) = zmeltt(it,itp,ivc) * vcoud(ivc)
00719                  z2r(itp,ivc) = zmeltt(it,itp,ivc)
00720              enddo
00721          enddo
00722          call zgemm('N','C',ntqxx,ntqxx,ngb,(ld0,0d0),zlr,ntqxx,
00723                  z2r,ntqxx,(0d0,0d0),w3pi,ntqxx)
00724 C      call zprm('w3pi',w3p,ntqxx,ntqxx,ntqxx)
00725 C      Faster, but harder to parallelize
00726 !          call zgsmpy(11,'N','C',ntqxx,ngb,zlr,ntqxx,z2r,ntqxx,
00727 !                      (0d0,0d0),w3pi,ntqxx)
00728 C      call zprm('w3pi',w3p,ntqxx,ntqxx,ntqxx)
00729          do itp = 1, ntqxx
00730              do itpp = 1, ntqxx
00731                  w3p(it,itp,itpp) = w3pi(itp,itpp)
00732              enddo
00733          enddo
00734      enddo
00735      vcoud(1) = vc      !restore vcoud(1)
00736      deallocate(zlr,z2r,w3pi)
00737      if(verbose())>=30) call cputid2(' complete w3p',0)
00738      deallocate(zmeltt)
00739      if(debug) then
00740          do it = 1,nctot+nbmax; do itp = 1,ntqxx
00741              write(6, "(' w3p = ',2i4,2d14.6)") it,itp,w3p(it,itp,itp)
00742          enddo; enddo
00743      endif
00744 !TIME1_0130 "end_of_w3p"
00745
00746 c$$$else
00747 c$$$!kino 2014-08-13 !$OMP parallel private(vc)
00748 c$$$!kino 2014-08-13 !$OMP do
00749 c$$$          do itp= 1,ntqxx
00750 c$$$          do it = 1,nctot+nbmax
00751 c$$$          do ivc=1,ngb
00752 c$$$              zmeltt(it,itp,ivc) = sum( zmel(:,it,itp)* ppovlz(:,ivc) )
00753 c$$$          enddo
00754 c$$$          enddo
00755 c$$$          enddo
00756 c$$$!kino 2014-08-13 !$OMP end do
00757 c$$$!kino 2014-08-13 !$OMP do
00758 c$$$          do 992 itpp= 1,ntqxx
00759 c$$$          do 993 itp = 1,ntqxx
00760 c$$$              if(diagonally.and.(itpp/=itp)) cycle
00761 c$$$!! sep2013t a test:c          if(itpp>ntqxxd .and.itp/=itpp) cycle
00762 c$$$          do 994 it = 1,nctot+nbmax
00763 c$$$              w3p(it,itp,itpp) = 0d0
00764 c$$$          do ivc=1,ngb
00765 c$$$              if(ivc==1.and.kx==iqini) then
00766 c$$$                  vc= wk(1)* fpi*sqrt(fpi) /wk(kx)
00767 c$$$c          write(6,*)'wk(1) vc=',wk(1),vc
00768 c$$$          else
00769 c$$$              vc= vcoud(ivc)
00770 c$$$          endif
00771 c$$$c          zmel1 = sum( zmel(:,it,itp) *ppovlz(:,ivc) )
00772 c$$$c          zmel2 = sum( zmel(:,it,itpp) *ppovlz(:,ivc) )
00773 c$$$c          w3p(it,itp,itpp) = w3p(it,itp,itpp)
00774 c$$$c          &          + vc * zmeltt(it,itp,ivc)*dconjg(zmeltt(it,itpp,ivc))
00775 c$$$          enddo
00776 c$$$ 994          continue
00777 c$$$ 993          continue
00778 c$$$ 992          continue
00779 c$$$!kino 2014-08-13 !$OMP end do
00780 c$$$!kino 2014-08-13 !$OMP end parallel
00781 c$$$endif
00782 !KINO          write(*,*)'kino: w3p checksum=',sum(w3p)
00783 c          deallocate(zmeltt)
00784 c$$$          else
00785 c$$$!kino 2014-08-13 !$OMP parallel do
00786 c$$$          do itpp= 1,ntqxx
00787 c$$$          do itp = 1,ntqxx
00788 c$$$              if(diagonally.and.(itpp/=itp)) cycle
00789 c$$$c sep2013t a test:c          if(itpp>ntqxxd .and.itp/=itpp) cycle
00790 c$$$          do it = 1,nctot+nbmax
00791 c$$$          w3p(it,itp,itpp) =dcmplx(

```

```

00792 c$$$      &                                sum ( dreal(zlp(:,it,itp))*rmelt(:,it,itpp)
00793 c$$$      &                                +   dimag(zlp(:,it,itp))*cmelt(:,it,itpp) ) ,
00794 c$$$      &                                sum ( dimag(zlp(:,it,itp))*rmelt(:,it,itpp)
00795 c$$$      &                                -   dreal(zlp(:,it,itp))*cmelt(:,it,itpp) ) )
00796 c$$$      &                                enddo
00797 c$$$      &                                enddo
00798 c$$$      &                                enddo
00799 c$$$!kino 2014-08-13  !$OMP end parallel do
00800 c$$$      &                                deallocate(zlp)
00801 c$$$      &                                endif
00802 c      &                                deallocate(zmel)
00803 c$$$!-- Write the Spectrum function for exchange May. 2001
00804 c$$$      &                                if(ifexsp/=0) then
00805 c$$$      &                                do it = 1, nctot+nbmax
00806 c$$$      &                                do itp = 1,ntqxx
00807 c$$$      &                                write(ifexsp,"(3i4, 3f12.4, ' ',d23.15,' ',d23.15)")
00808 c$$$      &                                ip,itp,it, qbz_kr, ekc(it), -wtt*dreal(w3p(it,itp,itp))
00809 c$$$      &                                enddo
00810 c$$$      &                                enddo
00811 c$$$      &                                endif
00812 c$$$!TIME1 "end of write ifexsp"
00813
00814 !TIME0_0180
00815 !! --- Correct weights wfac for valence by esmr
00816      do it = nctot+1, nctot+nbmax
00817          wfac = wfacx(-ld99, ef, ekc(it), esmr) !gaussian
00818          w3p(it,1:ntqxx,1:ntqxx) = wfac * w3p(it,1:ntqxx,1:ntqxx)
00819      enddo
00820
00821 !! apr2015 correct weights for core-hole case
00822      if(corehole) then
00823          do it = 1, nctot
00824              w3p(it,1:ntqxx,1:ntqxx) = wcorehole(it,isp) * w3p(it,1:ntqxx,1:ntqxx)
00825          enddo
00826      endif
00827
00828      do itpp=1,ntqxx
00829          do itp = 1,ntqxx !S[j=1,nbloch] zlp(j,t,n) <B(rk,j) psi(q-rk,n) |psi(q,t')>
00830              if(jobsww==5.and.(itpp/=itp)) cycle
00831 c sep2013t a test:c if(itpp>ntqxxd .and.itp/=itpp) cycle
00832              zsec(itp,itpp,ip) = zsec(itp,itpp,ip)
00833              & - wtt * sum( w3p(:,itp,itpp) )
00834          enddo
00835      enddo
00836      deallocate( w3p)
00837 c$$$      if(.not.newaniso()) deallocate(vcou1)
00838 !TIME1_0180 "enddo_zsec_wtt_sum"
00839      cycle ! next irot do 1000 loop
00840      endif ! end of if(exchange)
00841 !! ===== End of exchange section =====
00842      if(timemix) call timeshow("33333 k-cycle")
00843 cc!TIME1 "end of exchange section"
00844
00845
00846 !!-----
00847 !!--- correlation section -----
00848 !!-----
00849 !! We use the matrix elements zmel, which is given by "call get_zmelt"
00850 !!
00851 !!=====
00852 !! need to check the following notes.
00853 !! The correlated part of the self-energy:
00854 !! S[n=all] S[i,j=1,nbloch]
00855 !! <psi(q,t) |psi(q-rk,n) B(rk,i)>
00856 !! < [w'=0,inf] (1/pi) (w-e)/{(w-e)^2 + w'^2} Wc(k,iw')(i,j) >
00857 !! <B(rk,j) psi(q-rk,n) |psi(q,t)>
00858 !! e = e(q-rk,n), w' is real, Wc = W-v
00859 !!=====
00860 !! Get zwz0(omega=0, m, i, j), and zwz(i omega, m, i, j)
00861 !! m intermediate state. zwz= \sum_I,J <i|m I> W_IJ(i omega) <J m|j>
00862 !!
00863 !! sum over both occupied and unoccupied states and multiply by weight
00864 ! new from Jan2006! I think this should be OK. -----
00865 ! The output of sxcf_fal2 is <i|Re[S](e_i)|j> -----
00866 ! Im-axis integral gives Hermitian part of S.
00867 ! (Be careful as for the difference between
00868 ! <i|Re[S](e_i)|j> and transpose(dconjg(<i|Re[S](e_i)|j>)).
00869 ! ---because e_i is included.
00870 ! The symmetrization (hermitian) procedure is included in hqpe.sc.F
00871 ! old befor Jan2006
00872 ! & wtt*.5d0*( sum(zwzi(:,itp,itpp))+ !S_{ij}(e_i)
00873 ! & dconjg( sum(zwzi(:,itpp,itp)) ) ) !S_{ji}^*(e_j)= S_{ij}(e_j)
00874 !-----
00875 !! omega integration along im axis.
00876 !! zwzi(istate,itqxx1,itqxx2)=\int_ImAxis d\omega' zwz(omega',istate,itqxx1,itqxx2) 1/(omt-omega')
00877 !! ,where omt=omegat is given in the following 1385-1386 loop.
00878 !!

```

```

00879
00880
00881 !! -----
00882 !! Contribution to SEc(qt,w) from integration along the imaginary axis
00883 !!      loop over w' = (1-x)/x, frequencies in Wc(k,w')
00884 !!      {x} are gaussian-integration points between (0,1)
00885 !! -----
00886 !! Readin W(omega=0) and W(i*omega)
00887 !! Then get zwz0 and zwz
00888 !! zwz0 = (zmel*)*(W(omega=0) -v)*zmel
00889 !! zwz = (zmel*)*(W(i*omega(ix))-v)*zmel
00890 !TIME0_0200
00891      allocate( zwz0(      nstate,ntqxx,ntqxx))
00892      allocate( zwz(niw*npn,nstate,ntqxx,ntqxx))
00893      allocate( zw(nblochpmx,nblochpmx))
00894      ix = 1 + (0 - nw_i) !at omega=0 ! nw_i=0 (Time reversal) or nw_i=-nw
00895      read(ifrcw,rec=ix) zw ! direct access read Wc(0) = W(0) - v
00896      call matzwz2(2, zw(1:ngb,1:ngb), zmel, ntqxx, nstate,ngb,
00897      o      zwz0)
00898      do 1380 istate=1,nstate
00899      zwz0(istate,1:ntqxx,1:ntqxx) = ! w(iw) + w(-iw) Hermitian part.
00900      &      (zwz0(istate,1:ntqxx,1:ntqxx)
00901      &      + dconjg(transpose(zwz0(istate,1:ntqxx,1:ntqxx))))/2d0
00902 1380      continue
00903      do 1390 ix=1,niw      !niw is usually ~10 points.
00904      read(ifrcwi,rec=ix) zw ! direct access read Wc(i*omega)=W(i*omega)-v
00905      call matzwz2(2, zw(1:ngb,1:ngb), zmel, ntqxx, nstate,ngb,
00906      o      zwz(ix,1:nstate,1:ntqxx,1:ntqxx)) ! zwz = zmel*(W(0)-v)*zmel
00907      do 1395 istate=1,nstate
00908      zw(1:ntqxx,1:ntqxx)= zwz(ix,istate,1:ntqxx,1:ntqxx)
00909      zwz(ix,istate,1:ntqxx,1:ntqxx) = ! w(iw) + w(-iw) Harmitian part
00910      &      ( zw(1:ntqxx,1:ntqxx)
00911      &      + dconjg(transpose(zw(1:ntqxx,1:ntqxx))) )/2d0
00912      &      if(npm==2) then ! w(iw) - w(-iw) Anti Hermitian part
00913      zwz(ix+niw,istate,1:ntqxx,1:ntqxx) =
00914      &      ( zw(1:ntqxx,1:ntqxx)
00915      &      - dconjg(transpose(zw(1:ntqxx,1:ntqxx))) )/2d0/img
00916      endif
00917 1395      continue
00918 1390      continue
00919      deallocate(zw)
00920 !TIME1_0200 "endofdo1390"
00921 !! Integration along imag axis for zwz(omega) for given it,itp,itpp
00922 !! itp : left-hand end of expternal band index.
00923 !! itpp : right-hand end of expternal band index.
00924 !! it : intermediate state of G.
00925 !TIME0_0210
00926      allocate(zwzi(nstate,ntqxx,ntqxx))
00927      do 1400 itpp= 1,ntqxx
00928      do 1410 itp = 1,ntqxx
00929      if((jobsw==5).and.(itpp/=itp)) cycle
00930      if (jobsw==1.or.jobsw==4) then
00931      omegat = ef
00932 c      elseif (jobsw==2)      omegat=.5d0*(omega(itp)+omega(itpp))
00933      else
00934      omegat = omega(itp)
00935      endif
00936      do 1420 it = 1,nstate
00937      we =.5d0*( omegat -ekc(it))
00938      if(it <= nctot) then
00939      esmr = 0d0
00940      else
00941      esmr = esmr
00942      endif
00943 !! ua_auto may be recovered in future...
00944 c      if(ua_auto) then
00945 c      ratio = .5d0 *( abs(zwz(niw,it,itp,itp )/zwz0(it,itp,itp ))
00946 c      &      +abs(zwz(niw,it,itpp,itpp)/zwz0(it,itpp,itpp)) )
00947 c      call gen_ua(ratio,niw,freqx, expa_,ua_)
00948 c      endif
00949 !! Gaussian smearing. Integration along im axis. zwz(1:niw) and zwz0 are used.
00950      zwzi(it,itp,itpp) =
00951      &      wintzsg_npm(npm, zwz(1,it,itp,itpp), zwz0(it,itp,itpp)
00952      &      ,freqx,wx,ua_,expa_,we,niw,esmr)
00953 c      zwzi(it,itp,itpp) = !rectangular smearing only for npm=1
00954 c      &      wintzav ( zwz(1,it,itp,itpp),zwz0(it,itp,itpp)
00955 c      &      ,freqx,wx,ua_,expa_,we,niw, esmr)
00956 1420      continue
00957 1410      continue
00958 1400      continue
00959      deallocate(zwz0,zwz) !zwzs
00960      if(debug) print *, 'zzzzzzzz sum zwzi ',sum(abs(zwzi(:, :, :)))
00961 !TIME1_0210 "endofdo1400"
00962 !! Contribution to Sigma_{ij}(e_i)
00963      do 1500 itpp= 1,ntqxx
00964      do 1510 itp = 1,ntqxx
00965      if( jobsw==5.and.(itpp/=itp)) cycle

```

```

00966          zsec(itp,itpp,ip) = zsec(itp,itpp,ip) + wtt*sum(zwzi(:,itp,itpp))
00967 1510      continue
00968 1500      continue
00969          deallocate(zwzi)
00970          if(jobsw==4) goto 2002
00971
00972 !! -----
00973 !! Contribution to SEc(qt,w) from the poles of G (integral along real axis)
00974 !! Currently, jobsw =1,3,5 are allowed...
00975 !! The variable we means \omega_epsilon in Eq.(55) in PRB76,165106 (2007)
00976 !! -----
00977 !TIME0_0310
00978          if(timemix) call timeshow("goto Sec pole part k-cycle")
00979          if(debug) write(6,*)'GOTO contribution to SEc(qt,w) from the poles of G'
00980          if (.not.(jobsw == 1 .or. jobsw == 3.or.jobsw==5)) then
00981              call rx( 'sxcf_fal3_scz: jobsw /= 1 3 5')
00982          endif
00983 !! Get index nwx nwt nt_max. finish quickly. We can simplify this...
00984          call get_nwx(omega,ntq,ntqxx,nt0p,nt0m,nstate,freq_r,
00985              i      nw_i,nw,esmr,ef,ekc,wfaccut,nctot,nband,debug,
00986              o      nwx,nwt,nt_max)
00987 !! assemble small arrays first.
00988          allocate(we_(nt_max,ntqxx),wfac_(nt_max,ntqxx),ixss(nt_max,ntqxx),ititpskip(nt_max,ntqxx),iirx(
ntqxx))
00989          call weightset4intreal(nctot,esmr,omega,ekc,freq_r,nw_i,nw,
00990              i      ntqxx,nt0m,nt0p,ef,nwx,nwt,nt_max,wfaccut,wtt,
00991              o      we_,wfac_,ixss,ititpskip,iirx)
00992
00993 !! We need zw3, the Hermitian part, because we need only hermitean part of Sigma_nn'
00994 !! This can be large array; nwx-nwt+1 \sim 400 or so...
00995          allocate( zw3(ngb,ngb,nwx:nwt))
00996          allocate( zw(nblochpmx,nblochpmx))
00997          do ix = nwt,nwx
00998              nrec = ix-nw_i+1 !freq_r(ix is in nw_i:nx)
00999              read(ifrcw,rec=nrec) zw ! direct access Wc(omega) = W(omega) - v
01000              if(hermitianw) then
01001                  zw3(:, :, ix)=(zw(1:ngb,1:ngb)+transpose(dconjg(zw(1:ngb,1:ngb))))/2d0
01002              else
01003                  zw3(:, :, ix)=zw(1:ngb,1:ngb)
01004              endif
01005          enddo
01006          deallocate(zw)
01007 !! rearrange index of zmel
01008          allocate(zmell(ngb))
01009          if(jobsw==3) then
01010              allocate(zmell_(ntqxx,ngb,nstate))
01011              do itpp= 1,ntqxx
01012                  do it = 1,nstate
01013                      zmell_(itpp,1:ngb,it) = zmel(1:ngb,it,itpp)
01014                  enddo
01015              enddo
01016          endif
01017 !! jobsw==3
01018          if( jobsw==3) then
01019              allocate(zwz44(3,ntqxx),zwz4(ntqxx,3))
01020              do itp=1,ntqxx
01021                  do it=1,nt_max
01022                      if(ititpskip(it,itp)) cycle
01023                      we = we_(it,itp)
01024                      ixss= ixss(it,itp)
01025                      zmell(:)=dconjg(zmell(:,it,itp))
01026                      zwz4=0d0
01027                      do ix0=1,3
01028                          ix=ixss+ix0-2
01029                          do igb2=1,ngb
01030 ! !
01031                              **** most time consuming part ****
01032                              zz2=sum(zmell(1:ngb)*zw3(1:ngb,igb2, iirx(itp)*ix) )
01033                              call zaxpy(ntqxx,zz2,zmell_(1,igb2,it),1,zwz4(1,ix0),1)
01034                          enddo
01035                      zwz44 = transpose(zwz4)
01036                      do itpp=1,ntqxx
01037                          if(npm==1) then
01038                              zsec(itp,itpp,ip) = zsec(itp,itpp,ip)
01039                              + wfac_(it,itp) * alagr3z2(we,freq_r(ixs-1),zwz44(1,itpp),itp==itpp ) !mar015
, itp,itpp)
01040                          else
01041                              zsec(itp,itpp,ip) = zsec(itp,itpp,ip)
01042                              + wfac_(it,itp) * alagr3z2(we,freq_r(ixs-1),zwz44(1,itpp))
01043                          endif
01044                      enddo
01045                  enddo
01046              enddo
01047              deallocate(zwz44,zwz4)
01048          endif
01049
01050 !! jobsw=1,5 Sigma are calculated.

```

```

01051         if( jobsw==1.or.jobsw==5) then
01052             do itp=1,ntqxx
01053                 do it=1,nt_max
01054                     if(ititpskip(it,itp)) cycle
01055                     we = we_(it,itp)
01056                     ix= ixss(it,itp)
01057                     zmel1(:)=dconjg(zmel(:,it,itp))
01058                     zwz3=0d0
01059                     do ix0=1,3
01060                         ix=ixs+ix0-2
01061                         **** most time consuming part for jobsw=1 *****
01062                         !! To reduce computational time, confusing treatment only uses lower half of zw3 (zw3 is
Hermitian)
01063                         !! Clean up needed.
01064
01065                         !! zwz3 contains <itp| it I> wz3_IJ(we) <J it| itp>
01066                         !! when zw3 is hermitian.
01067                         if(hermitianw) then
01068                             do igb2=2,ngb
01069                                 zz2 = sum(zmel1(1:igb2-1)*zwz3(1:igb2-1,igb2,iirx(itp)*ix) ) +
01070                                     .5d0* zmel1(igb2)*zwz3(igb2,igb2,iirx(itp)*ix)
01071                                 & zwz3(ix0) = zwz3(ix0)+zz2*zmel(igb2,it,itp)
01072                                 enddo !igb2
01073                                 zwz3(ix0) = 2d0*dreal(zwz3(ix0))+ !I think 2d0 is from upper half.
01074                                 & zmel1(1)*zwz3(1,1, iirx(itp)*ix)*zmel(1,it,itp)
01075                         !! when zw3 is not need to be hermitian case. This gives life time
01076                         else
01077                             zwz3(ix0) = sum( matmul(zmel1(1:ngb), zwz3(1:ngb,1:ngb,iirx(itp)*ix))*zmel(1:ngb,it,
itp) )
01078                             endif
01079                         enddo
01080                         if(npm==1) then
01081                             zsec(itp,itp,ip) = zsec(itp,itp,ip)
01082                             + wfac_(it,itp)*alagr3z2(we,freq_r(ixs-1),zwz3,.true.)
01083                         else
01084                             zsec(itp,itp,ip) = zsec(itp,itp,ip)
01085                             + wfac_(it,itp)*alagr3z(we,freq_r(ixs-1),zwz3)
01086                         endif
01087                     enddo
01088                 enddo
01089             endif
01090             !TIME1_0310 "EndReCorrelation"
01091             c goto 2012
01092
01093             cxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
01094             c$$$
01095             c$$$
01096             c$$$
01097             c$$$$cccccccccc old code ccccccccccccccccccccccccccc
01098             c$$$ if(timemix) call timeshow("55555 k-cycle")
01099             c$$$ if(debug) write(*,'(a,5i6)') 'kino: ntqxx,itini,itend,ngb=',ntqxx,itini,itend,ngb
01100             c$$$$c$$$ if(test_symmetric_W().and.npm==2) then
01101                 c$$$$c$$$ if(onceww(4)) write(6,*) ' test_symmetric_W()=' ,test_symmetric_W(),nwx,nwx
01102                 c$$$$c$$$ allocate(zw3x(ngb,ngb))
01103                 c$$$$c$$$ do ix= 1,min(abs(nwx),nwx)
01104                     c$$$$c$$$ zw3x = 0.5d0* (zw3(:,ix) + zw3(:,ix))
01105                 c$$$$c$$$ zw3(:,ix)=zw3x
01106                 c$$$$c$$$ zw3(:,ix)=zw3x
01107                 c$$$$c$$$ enddo
01108                 c$$$$c$$$ deallocate(zw3x)
01109             c$$$$c$$$ endif
01110             c$$$!TIME1 "before 2001"
01111             c$$$!TIME0
01112             c$$$ allocate(zwz44(3,ntqxx),zwz4(ntqxx,3))
01113             c$$$ do 2001 itp = 1,ntqxx ! loop over states (q-k,n)
01114                 c$$$ omg = omega(itp)
01115                 c$$$ if (omg >= ef) then
01116                     c$$$ itini= nt0m+1
01117                     c$$$ itend= nt_max
01118                     c$$$ iii= 1
01119                 c$$$ else
01120                     c$$$ itini= 1
01121                     c$$$ itend= nt0p
01122                     c$$$ iii= -1
01123                 c$$$ endif
01124                 c$$$ do 2011 it = itini,itend ! nt0p corresponds to efp
01125                     c$$$ esmr = esmr
01126                     c$$$ if(it<=nctot) esmr = 0d0
01127                     c$$$ wfac = wfacx2(omg,ef, ekc(it),esmr)
01128                     c$$$ if(wfac<wfaccut) cycle ! next it
01129                     c$$$ we = .5d0* abs( omg-weavx2(omg,ef, ekc(it),esmr) ) !Gaussian smearing
01130                     c$$$ if(it<=nctot .and.wfac>wfaccut) call rx( "sxcf: it<=nctot.and.wfac/=0")
01131                     c$$$$c$$$ Rectangular smearing
01132                     c$$$$c$$$ if( wfac==0d0) cycle ! next it
01133                     c$$$$c$$$ if( omg >= ef) we = 0.5d0* abs( max(omg-ekc(it), 0d0) )
01134                     c$$$$c$$$ if( omg < ef) we = 0.5d0* abs( min(omg-ekc(it), 0d0) )
01135                     c$$$$c$$$ if( it<=nctot) then !faleev

```

```

01136 c$$$c$$$          if(wfac/=0) call rx( "sxcf: it<=nctot.and.wfac/=0")
01137 c$$$c$$$          endif
01138 c$$$c$$$          endif
01139 c$$$          if(debug) write(6,"( ' xxx1',10d13.6)") omg,ef, ekc(it),wfac
01140 c$$$          wfac= iiii* wfac*wtw
01141 c$$$          do iwp = 1,nw
01142 c$$$             ix= iwp
01143 c$$$             if(freq_r(iwp)>we) exit
01144 c$$$          enddo
01145 c$$$          if(nw_i==0) then
01146 c$$$             if(ixs+1>nwx) call rx( ' sxcf: ixs+1>nwx xxx2')
01147 c$$$          else
01148 c$$$             if(omg >=ef .and. ixs+1> nwx ) then
01149 c$$$                write(6,*)'ixs+1 nwx=',ixs+1,nwx
01150 c$$$                call rx( ' sxcf: ixs+1>nwx yyy2a')
01151 c$$$            endif
01152 c$$$            if(omg < ef .and. abs(ixs+1)> abs(nwxi) ) then
01153 c$$$                write(6,*)'ixs+1 nwxi=',ixs+1,nwxi
01154 c$$$                call rx( ' sxcf: ixs-1<nwi yyy2b')
01155 c$$$            endif
01156 c$$$          endif
01157 c$$$          iir = 1
01158 c$$$          if(omg < ef .and. nw_i/=0) iir = -1
01159 c$$$          zmel1(:)=dconjg(zmel(:,it,itp))
01160 c$$$
01161 c$$$          if (jobsw == 1.or.jobsw==5) then
01162 c$$$             zwz3=(0d0,0d0)
01163 c$$$!kino 2014-08-13 !$OMP parallel do private(ix,zz2)
01164 c$$$             do 2014 ix0=1,3
01165 c$$$                ix=ixs+ix0-2
01166 c$$$                do igb2=2,ngb
01167 c$$$! !**** most time consuming part for jobsw=1 *****
01168 c$$$                zz2=sum(zmel1(1:igb2-1)*zwz3(1:igb2-1,igb2,iir*ix) ) +
01169 c$$$                & .5d0* zmel1(igb2)*zwz3(igb2,igb2,iir*ix)
01170 c$$$                zwz3(ix0)=zwz3(ix0)+zz2*zmel(igb2,it,itp)
01171 c$$$            enddo
01172 c$$$                !igb2
01173 c$$$                zwz3(ix0)=2d0*dreal(zwz3(ix0))+
01174 c$$$                & zmel1(1)*zwz3(1,1, iir*ix)*zmel(1,it,itp)
01175 c$$$            continue
01176 c$$$!kino 2014-08-13 !$OMP end parallel do
01177 c$$$            if(npm==1) then
01178 c$$$                zsec(itp,itp,ip) = zsec(itp,itp,ip)
01179 c$$$                + wfac*alagr3z2(we,freq_r(ixs-1),zwz3,itp,itp)
01180 c$$$            else
01181 c$$$                zsec(itp,itp,ip) = zsec(itp,itp,ip)
01182 c$$$                + wfac*alagr3z(we,freq_r(ixs-1),zwz3)
01183 c$$$            endif
01184 c$$$! this contribution to zsec_nn is real (hermitean)
01185 c$$$
01186 c$$$          elseif(jobsw == 3) then
01187 c$$$             zwz4=(0d0,0d0)
01188 c$$$!$OMP parallel private(ix,zz2)
01189 c$$$             do 2015 ix0=1,3
01190 c$$$                ix=ixs+ix0-2
01191 c$$$!$OMP do reduction(+:zwz4)
01192 c$$$! Next xzpy is most time consuming part for jobsw=3.****
01193 c$$$! I think we can speed up this section...
01194 c$$$             do igb2=1,ngb
01195 c$$$                zz2=sum(zmel1(1:ngb)*zwz3(1:ngb,igb2, iir*ix) )
01196 c$$$                call xzpy(ntqxx,zz2,zmel1_(1,igb2,it),1,zwz4(1,ix0),1)
01197 c$$$            enddo
01198 c$$$            continue
01199 c$$$!$OMP end parallel
01200 c$$$            zwz44 = transpose(zwz4)
01201 c$$$            do itpp=1,ntqxx
01202 c$$$                if(jobsw==5.and.(itpp/=itp)) cycle
01203 c$$$                if(npm==1) then
01204 c$$$                    zsec(itp,itpp,ip) = zsec(itp,itpp,ip)
01205 c$$$                    + wfac*alagr3z2(we,freq_r(ixs-1),zwz44(1,itpp),itp,itpp)
01206 c$$$                else
01207 c$$$                    zsec(itp,itpp,ip) = zsec(itp,itpp,ip)
01208 c$$$                    + wfac*alagr3z(we,freq_r(ixs-1),zwz44(1,itpp))
01209 c$$$                endif
01210 c$$$            enddo
01211 c$$$            !itpp
01212 c$$$            ! inner jobsw=1 or 3
01213 c$$$! this contribution to zsec_nn' is not hermitean because W(e_n)
01214 c$$$! and must be made hermitean when zsec will be written on disc
01215 c$$$            continue
01216 c$$$            !itp
01217 c$$$            continue
01218 c$$$            deallocate(we_,wfac_,ixss,ititpskip,iirx)
01219 c$$$            continue
01220 c$$$            deallocate(zwz3, zmel, zmel1)
01221 c$$$            if(allocated(zmel1_)) deallocate(zmel1_)
01222 c$$$            if(verbose())>50) call timeshow("11after alagr3z iw,itp,it cycles")

```



```

01223         if(debug) then
01224             write(6,*)' end of do 2001 '
01225             do itp = 1,ntq
01226                 write(6,(' zsec=",i3,2d15.7)') itp,zsec(itp,itp,ip)
01227             enddo
01228         endif
01229 1000     continue                ! end do irot
01230         ifvcoud = iclose('Vcoud.'//charnum5(kx))
01231         if(.not.exchange) then
01232             ifrcw = iclose('WVR.'//charnum5(kx))
01233             ifrcwi = iclose('WVI.'//charnum5(kx))
01234         endif
01235         deallocate(ppovlz)
01236 1100     continue                ! end of kx-loop
01237         ifvcoud = iclose('Vcoud')
01238         if(irot==1) write(6,(' sum(abs(zsec))=',d23.15)') sum(abs(zsec))
01239         if (allocated(vcou)) deallocate(vcou)
01240 1001     continue                ! end do ip
01241 c       if(allocated(freq_r)) deallocate(freq_r)
01242 c       if (allocated(expikt)) deallocate(expikt)
01243 c!TIME1_0000 "end of sxcf_fal3_scz"
01244         end subroutine sxcf_fal3_scz
01245
01246
01247     subroutine weightset4intreal(nctot,esmr,omega,ekc,freq_r,nw_i,nw,
01248 i ntqxx,nt0m,nt0p,ef,nwx,nwxi,nt_max,wfaccut,wt,
01249 o we_,wfac_,ixss,ititpskip,iirx)
01250 !! generate required data set for main part of real part integration.
01251     implicit none
01252     integer,intent(in):: ntqxx,nctot,nw_i,nw,nt0m,nwx,nwxi,nt_max
01253     real(8),intent(in):: ef,omega(ntqxx),ekc(ntqxx),freq_r(nw_i:nw),esmr,wfaccut,wt
01254     real(8),intent(out):: we_(nt_max,ntqxx),wfac_(nt_max,ntqxx)
01255     integer,intent(out) :: ixss(nt_max,ntqxx),iirx(ntqxx)
01256     logical,intent(out) :: ititpskip(nt_max,ntqxx)
01257     integer:: itini,iii,it,itend,wp,ixs,itp,iwp,nt0p
01258     real(8):: omg,esmr,x,wfacx2,we,wfac,weavx2
01259     ititpskip=.false.
01260     do itp = 1,ntqxx                !this loop should finish in a second
01261         omg = omega(itp)
01262 !     jobsw==2
01263 !         if (jobsw==2) omg=.5d0*(omega(itp)+omega(itpp))
01264         iirx(itp) = 1
01265         if( omg < ef .and. nw_i/=0) iirx(itp) = -1
01266         if (omg >= ef) then
01267             itini= nt0m+1
01268             itend= nt_max
01269             iii= 1
01270         else
01271             itini= 1
01272             itend= nt0p
01273             iii= -1
01274         endif
01275         ititpskip(:itini-1,itp)=.true.
01276         ititpskip(itend+1,itp)=.true.
01277         do it = itini,itend          ! nt0p corresponds to efp
01278             esmr = esmr
01279             if(it<=nctot) esmr = 0d0
01280             wfac_(it,itp) = wfacx2(omg,ef, ekc(it),esmr)
01281             wfac = wfac_(it,itp)
01282             if(wfac<wfaccut) then
01283                 ititpskip(it,itp)=.true.
01284             cycle
01285             endif
01286             wfac_(it,itp)= wfac_(it,itp)*wt*iii
01287 ! Gaussian smearing we_ = \bar{\omega\epsilon} in sentences next to Eq.58 in PRB76,165106 (2007)
01288 ! wfac_ = $w$ weight (smeared thus truncated by ef). See the sentences.
01289             we_(it,itp) = .5d0* abs( omg-weavx2(omg,ef, ekc(it),esmr) )
01290             we = we_(it,itp)
01291             if(it<=nctot.and.wfac>wfaccut) call rx( .and."sxcf: it<=nctotwfac/=0")
01292             do iwp = 1,nw
01293                 ix = iwp
01294                 if(freq_r(iwp)>we) exit
01295             enddo
01296             ixss(it,itp) = ix
01297             if(nw_i==0) then
01298                 if(ixs+1>nwx) call rx( ' sxcf: ixs+1>nwx xxx2')
01299             else
01300                 if(omg >=ef .and. ixs+1> nwx ) then
01301                     write(6,*)'ixs+1 nwxi=',ixs+1,nwxi
01302                     call rx( ' sxcf: ixs+1>nwx yyy2a')
01303                 endif
01304                 if(omg < ef .and. abs(ixs+1)> abs(nwxi) ) then
01305                     write(6,*)'ixs+1 nwxi=',ixs+1,nwxi
01306                     call rx( ' sxcf: ixs-1<nwi yyy2b')
01307                 endif
01308             endif
01309         enddo

```



```

01310         enddo
01311     end subroutine weightset4intreal
01312 end module m_sxcfsc
01313 !! -----
01314 subroutine get_nwx(omega,ntq,ntqxx,nt0p,nt0m,nstate,freq_r,
01315     i nw_i,nw,esmr,ef,ekc,wfaccut,nctot,nband,debug,
01316     o nwxi,nwx,nt_max)
01317 !> Determine indexes of a range for calculation.
01318 !! It is better to clean this up...
01319     implicit none
01320     integer,intent(in) :: nctot,nw_i,nw,nstate,nt0p,nt0m,ntq,
01321     & nband,ntqxx
01322     real(8),intent(in):: omega(ntq),esmr,ef,ekc(nctot+nband),wfaccut,
01323     & freq_r(nw_i:nw)
01324     integer,intent(out) :: nt_max,nwxi,nwx
01325
01326     integer:: itp,it,itini,itend,iwp,ixs,ixsmin,ixsmx,verbose
01327     real(8):: omg,wfac,wfacx2,we,weavx2,esmr,x,wexx
01328     logical:: debug
01329 !!     maximum ixs required.
01330     ixsmx = 0
01331     ixsmin = 0
01332     do 301 itp = 1,ntqxx
01333         omg = omega(itp)
01334         if (omg < ef) then
01335             itini = 1
01336             itend = nt0p
01337         else
01338             itini = nt0m+1
01339             itend = nstate
01340         endif
01341         do 311 it = itini, itend
01342             esmr = esmr
01343             if(it<=nctot) esmr = 0d0
01344             wfac = wfacx2(omg,ef, ekc(it),esmr)
01345             if(wfac<wfaccut) cycle !Gaussian case
01346             we = .5d0*(weavx2(omg,ef,ekc(it),esmr)-omg)
01347 cc Gaussian=F case keep here just as a memo
01348 c             if(wfac==0d0) cycle ! next it
01349 c             if(omg>=ef) we = max( .5d0*(omg-ekc(it)), 0d0) ! positive
01350 c             if(omg< ef) we = min( .5d0*(omg-ekc(it)), 0d0) ! negative
01351             if(it<=nctot) then
01352                 if(wfac>wfaccut) call rx( .and."sxcf: it<=nctotwfac/=0")
01353             endif
01354             do iwp = 1,nw
01355                 ixs=iwp
01356                 if(freq_r(iwp)>abs(we)) exit
01357             enddo
01358 c             This change is because G(omega-omg') W(omg') !may2006
01359 c             if(ixs>ixsmx .and. omg<=ef ) ixsmx = ixs
01360 c             if(ixs>ixsmin .and. omg> ef ) ixsmin = ixs
01361             if(ixs>ixsmx .and. omg>=ef ) ixsmx = ixs
01362             if(ixs>ixsmin .and. omg< ef ) ixsmin = ixs
01363             wexx = we
01364             if(ixs+1 > nw) then
01365                 write (*,*) ' nw_i ixsmin',nw_i, ixsmin
01366                 write (*,*) ' wexx ',wexx
01367                 write (*,*) ' omg ekc(it) ef ', omg,ekc(it),ef
01368                 call rx( ' sxcf 222: |w-e| out of range' )
01369             endif
01370 311 continue
01371 301 continue !end of SEc w and qt -loop
01372 !!
01373     if(nw_i==0) then !time reversal
01374         nwxi = 0
01375         nwx = max(ixsmx+1,ixsmin+1)
01376     else !no time reversal
01377         nwxi = -ixsmin-1
01378         nwx = ixsmx+1
01379     endif
01380     if (nwx > nw ) then
01381         call rx( ' sxcf_fal3_sc nwx check : |w-e| > max(w)')
01382     endif
01383     if (nwxi < nw_i) then
01384         call rx( ' sxcf_fal3_sc nwxi check: |w-e| > max(w)')
01385     endif
01386     if(debug) write(6,*) 'nw, nwx=',nw,nwx
01387     if(verbose()>50)call timeshow("10before alagr3z iw,itp,it ")
01388 !! Find nt_max
01389     nt_max=nt0p !initial nt_max
01390     do 401 itp = 1,ntqxx
01391         omg = omega(itp)
01392         if (omg > ef) then
01393             do it = nt0m+1,nstate ! nt0m corresponds to efm
01394                 wfac = wfacx2(ef,omg, ekc(it),esmr)
01395 c                 if( (GaussSmear().and.wfac>wfaccut)
01396 c                 & .or.(.not.GaussSmear().and.wfac/=0d0)) then

```

```

01397         if(wfac>wfacut) then
01398             if (it > nt_max) nt_max=it ! nt_max is unocc. state
01399             endif ! that ekc(it>nt_max)-omega > 0
01400             enddo ! so it > nt_max does not contribute to omega pole integral
01401         endif
01402 401 continue !end of w and qt -loop
01403     end subroutine get_nwx

```

## 4.25 gwsr/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,qbas, ginv, rk, wk,
- subroutine [x0kf\\_v4hz\\_symmetrize](#) (npm,
- 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)

### 4.25.1 Function/Subroutine Documentation

4.25.1.1 logical function [checkbelong](#) ( real(8), dimension(3) *qin*, real(8), dimension(3,nq) *qall*, integer *nq*, integer *ieibz* )

Definition at line [1567](#) of file [x0kf\\_v4h.F](#).

4.25.1.2 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 [1264](#) of file [x0kf\\_v4h.F](#).

Here is the call graph for this function:

Here is the caller graph for this function:

4.25.1.3 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 [1582](#) of file [x0kf\\_v4h.F](#).

Here is the caller graph for this function:

#### 4.25.1.4 real(8) function wcutef ( real(8) e, real(8) ecut, real(8) ecuts )

Definition at line 1683 of file x0kf\_v4h.F.

Here is the caller graph for this function:

4.25.1.5 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, 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:

4.25.1.6 subroutine x0kf\_v4hz\_symmetrize ( integer(4) npm )

Definition at line 866 of file x0kf\_v4h.F.

Here is the caller graph for this function:

## 4.26 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          chipmzr,eibzmode,
00016      i          nwgt,igx,igxt,ngrp,eibzsym, crpa)
00017      use m_readqg,only : readqg
00018      use m_readeigen,only: readeval
00019      use m_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> = 0 , 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-e_j))
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 accumulating 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. indxr.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 nlmt0 = total number of LMT0 basis functions
00082 c ngbz = 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,nlmt0 !,noccx,noccxv
00094 & ,nl,nclass,nnc,nlnmx,nbloch,ig,nqibz,iatom,nctot,nbm,lopen !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 c & ngvecp_kq(3,ngpmx),ngvecp_k(3,ngpmx)
00106 complex(8),allocatable :: zmelt(:,,:),zmeltl(:,,:)
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:: zlp(:,,:)

```

```

00116      integer(4) :: nbnb(nqbz,npm),
00117      & n1b(nbnbx,nqbz,npm), n2b(nbnbx,nqbz,npm)
00118      complex(8),allocatable:: zzm1(:, :, :)
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)!,ebandr(nband),ebandqr(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),tolq=1d-8,quu1(3),quu2(3) !tolqu=1d-4,
00147
00148      integer(4):: nbcut,nbcut2
00149      logical :: iww1=.true.,iww2=.true.
00150
00151      logical:: smbasis
00152      integer(4):: ifpomat, nbloch_r, ngbo,ixdummy
00153 c      integer(4),allocatable:: io(:,),in(:,),io_q(:,),in_q(:,)
00154      complex(8),allocatable:: pomat(:, :, ), zmelt1n(:, :, )
00155      real(8):: q_r(3)
00156      complex(8):: img=(0d0,1d0),zzz(ngbb)
00157
00158      integer(4):: nkmin, nkmax, nkqmin, nkqmax,nkmax1,nkqmax1
00159 c      real(8):: qq(3)
00160      integer(4):: ib1, ib2,      ngcx,ix,iy
00161
00162 c      integer(4),allocatable:: ngvecc(:,)
00163      logical :: onceww !testtr,negative_testtr
00164
00165 !! takao apr2012
00166      logical :: zloffd !, z1stcol
00167      complex(8),target :: zzz(ngbb,nmbas) !ppv1z(ngbb,ngbb),
00168      integer:: igb
00169 c      logical:: symmetrize
00170
00171 !! jun2012takao
00172 c      real(8):: qeibz(3,nqbz), ! aik(3,3,ngrpt)
00173      integer:: ngrp,nwgt(nqbz) !,ngrpt, aiktimereversal(ngrp),nwgtieibz,ieibz
00174      integer:: igx(ngrp*2,nqbz),igxt(ngrp*2,nqbz),ieqbz
00175 !! nwgt(neibz
00176      logical:: checkbelong,eibzmode, chipmzrr
00177      complex(8):: zcousq(ngbb,ngbb) !ppv1(ngc,ngc) ,
00178      complex(8),allocatable:: zcousqr(:, :, ),rcxq0(:, :, ),rcxq00(:, :, ),rcxq000(:, :, ),rcxqwww(:, :, )
00179 c      complex(8):: zcousqrsum(ngbb,ngbb,2),      zmeltx(ngbb),zmeltz(ngbb),zcousqrx(ngbb,ngbb)
00180      complex(8):: zmeltx(ngbb),zmeltz(ngbb),zcousqrx(ngbb,ngbb), zcousqc(ngbb,ngbb)
00181      & ,zrc(ngbb,ngbb),cmat(ngbb,ngbb) !zcousqinv(ngbb,ngbb),
00182      integer:: eibzsym(ngrp,-1:1),neibz,icc,ig,eibzmoden,ikp,i,j,itimer,icount,iele
00183      integer:: irotm,nrotx,ixx,iyy,itt,ntimer, nccc, nxx,iagain,irotml,irotm2
00184      integer,allocatable:: il(:, :, ),i2(:, :, ),nrotm(:,)
00185      complex(8),allocatable:: zrotm(:, :, ),zrr(:, :, ),zrrc(:, :, ),zrr_(:, :, ),zrrc_(:, :, ),zmmm(:, ),zrrx(:, )
00186 c      complex(8),pointer:: zmat(:,)
00187 c      complex(8),allocatable,target:: ppov1(:,)
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 c      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$$$!! check eibzmode assumes nmbas1=nmbas2
00210 c$$$      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 c      print *, 'qqqqqqqqqq',qbas
00224 c      call minv33(qbas,qbasinv)
00225 c      phase=(1d0,0d0)!coskt = 1d0; sinkt = 0d0
00226 c      allocate(cphi_k(nlmt0,nband),cphi_kq(nlmt0,nband), geig_kq(ngpmx,nband), geig_k(ngpmx,nband) )
00227 c      call getkeyvalue("GWinput","nbcutlow",nbcut, default=0 )
00228 c      call getkeyvalue("GWinput","nbcutlowto",nbcut2, default=0 )
00229 c      call getnemx(nbm,ebmx,7,.true.)
00230 c$$$!TIME0
00231 c$$$      if(smbasis()) then !need to check again, when we will make smbasis on.
00232 c$$$$cccccccccccccccccccccccccccccccccccc
00233 c$$$      if(ncc/=0) then
00234 c$$$          write(6,*)"Timereversal=F(ncc/=0) not yet implemented for smbasis."
00235 c$$$          write(6,*)" pomat should be generated correctly ."
00236 c$$$Cstop2rx 2013.08.09 kino      stop "Timereversal=F(ncc/=0) not yet implemented for smbasis."
00237 c$$$      call rx("Timereversal=F(ncc/=0) not yet implemented for smbasis.")
00238 c$$$      endif
00239 c$$$$cccccccccccccccccccccccccccccccccccc
00240 c$$$
00241 c$$$      ifpomat = iopen('POmat',0,-1,0) !oct2005
00242 c$$$C... smoothed mixed basis !oct2005
00243 c$$$C This replace original zmelt with new zmelt based on smoothed mixed basis.
00244 c$$$      do
00245 c$$$          read(ifpomat) q_r,nn,no,ixdummy !readin reduction matrix pomat
00246 c$$$          write(6,('ttt: q =',3f12.5)") q
00247 c$$$          write(6,('ttt: q_r=',3f12.5)") q_r
00248 c$$$          allocate( pomat(nn,no) )
00249 c$$$          read(ifpomat) pomat
00250 c$$$          if( sum(abs(q-q_r))<1d-10) then ! .and.kx <= nqibz ) then
00251 c$$$              write(6,*) 'ok find the section for give qibz_k'
00252 c$$$              exit
00253 c$$$          elseif (kx >nqibz ) then
00254 c$$$              exit
00255 c$$$          endif
00256 c$$$          deallocate(pomat)
00257 c$$$      enddo
00258 c$$$      if( sum(abs(q-q_r))>1d-10 ) then
00259 c$$$          write(6,('q =',3f12.5)") q
00260 c$$$          write(6,('q_r=',3f12.5)") q_r
00261 c$$$Cstop2rx 2013.08.09 kino      stop 'POmat reading err q/=q_r'
00262 c$$$      call rx('POmat reading err q/=q_r')
00263 c$$$      endif
00264 c$$$      isx = iclose('POmat')
00265 c$$$      endif
00266 c$$$!TIME1 "after if smbasis"
00267
00268 ckino
00269 !KINO      it=0
00270 !KINO      do k=1,nqbz
00271 !KINO          if(eibzmode.and.nwgt(k)==0 ) cycle
00272 !KINO          if(sum(nbnb(k,1:npm))==0) cycle
00273 !KINO          it=it+1
00274 !KINO      enddo
00275 !KINO      write(6,('a,i3,lx,a,i4')'iq=',iq,'active-k-points=',it
00276 ckinoend
00277
00278 !TIME1_1001 "beforedo1000"
00279 !! loop over k-points -----
00280 c      qq=0d0
00281 c      do 1000 k = 1,nqbz
00282 c          if(eibzmode.and.nwgt(k)==0 ) cycle
00283 c          if(debug) write(6,('do 1000 k=",i4,3f10.4)')k,rk(:,k)
00284 c          ipr=(k<5.or.k==nqbz.or.debug)
00285 c          if(sum(nbnb(k,1:npm))==0) cycle
00286 !TIME0_1101
00287 c          if(k<=5.or. (mod(k,max(10,nqbz/20))==1.or.k>nqbz-10) ) then
00288 c              write(6,(' x0kf_v4hz: k rk=',i7,3f10.4,$)k, rk(:,k)
00289 c              call cputid(0)

```

```

00290         endif
00291 cccccccccc
00292 c          write(6,('xxxxxxx  x0kf_v4hz: k rk=',i7,3f10.4,$))k, rk(:,k)
00293 cccccccccc
00294 !! --- tetra ----- override nt0 itps ntp0 -----
00295         nkmin = 999999
00296         nkmax= -999999
00297         nkqmin= 999999
00298         nkqmax=-999999
00299         do jpm=1,npm !npm
00300             do ibib = 1, nbnb(k,jpm)
00301                 nkmin = min(nlb(ibib,k,jpm),nkmin)
00302                 nkqmin = min(n2b(ibib,k,jpm),nkqmin)
00303                 if(nlb(ibib,k,jpm)<=nbnd) nkmax = max(nlb(ibib,k,jpm),nkmax)
00304                 if(n2b(ibib,k,jpm)<=nbnd) nkqmax = max(n2b(ibib,k,jpm),nkqmax)
00305             enddo
00306         enddo
00307 !! ebmx band cutoff is
00308 c$$$!!
00309 c$$$ nkmax1=nkmax
00310 c$$$ nkqmax1=nkqmax
00311 c$$$$cccccccccccccccccccccccccccccccccccc
00312 c$$$c ebndqr=eband
00313 c$$$c call readeval(rk(:,k),isp_kq,ebandr)
00314 c$$$$cccccccccccccccccccccccccccccccccccc
00315 c$$$c call readeval(q+rk(:,k),isp_kq,eband)
00316 c$$$ nkqmax = nocc (eband,ebmx,.true.,nbnd)
00317 c$$$ if(npm==2) then
00318 c$$$ call readeval(rk(:,k),isp_k,eband)
00319 c$$$ nkmax = nocc (eband,ebmx,.true.,nbnd)
00320 c$$$ endif
00321 c$$$ write(6,('nnnnnnk ',2i5,2i5))nkmax1, nkmax
00322 c$$$ write(6,('nnnnnnkq ',2i5,2i5))nkqmax1,nkqmax
00323 c$$$!!
00324         itps = nkqmin ! nkqmin = the num of min n2=unocc for jpm=1
00325         nt0 = nkmax
00326         ntp0 = nkqmax - nkqmin +1
00327         if( npm==2.and. nkqmin/=1) then
00328             write(6,*)' npm==2 nkqmin nkqmax nkmin nkmax=',nkqmin,nkqmax,nkmin,nkmax
00329             call rx( " When npm==2, nkqmin==1 should be.")
00330         endif
00331         if(nkmin/=1) then
00332             call rx( " nkmin==1 should be.")
00333         endif
00334
00335 !... feb2006
00336 ! zzmel(1:nbloch, ib_k,ib_kq)
00337 !     ib_k =[1:nctot] core
00338 !     ib_k =[nctot+1:nctot+nkmax] valence
00339 !     ib_kq =[1:nctot] core
00340 !     ib_kq =[ncc+1:ncc+nkqmax - nkqmin] valence range [nkqmin,nkqmax]
00341 !     If jpm=1, ncc=0.
00342 !     If jpm=2, ncc=ncore. itps=1 should be.
00343 ! There is a little confusion. nlb index contains cores are after valence.
00344 ! You can see codes to treat the confusion.
00345 ! NOTE:
00346 ! q+rk n2b vec_kq vec_kq_g geig_kq cphi_kq ngp_kq ngvecp_kq isp_kq
00347 ! rk nlb vec_k vec_k_g geig_k cphi_k ngp_k ngvecp_k isp_k
00348
00349 c         if(ipr) then
00350 c             write(6,(' nkRange nkqRange=',2i6,2x,2i6)) nkmin,nkmax,nkqmin,nkqmax
00351 c         endif
00352
00353
00354
00355 c$$$$cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00356 c$$$ goto 8828
00357 c$$$$cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00358 c$$$
00359 c$$$
00360 c$$$
00361 c$$$
00362 c$$$
00363 c$$$!TIME1 "before readphi"
00364 c$$$!TIME0
00365 c$$$!-- calculate the matrix elements <psi(k+q,t') | psi(k,t) B(R,i)>
00366 c$$$! Read cphi part of eigenfunctions for k and k+q
00367 c$$$ call readcphi(q+rk(:,k)-qq, nlmt0,isp_kq, quu2,cphi_kq)
00368 c$$$ call readcphi( rk(:,k)-qq, nlmt0,isp_k, quu1,cphi_k) !quu is used q for eigenfunctions.
00369 c$$$! ... core
00370 c$$$ if(debug) call cputid(0)
00371 c$$$! allocate( zzmel(nbloch,nccx, ntp0) )
00372 c$$$! q k q+k
00373 c$$$ if(debug) write(6,*)nbloch,nctot,nt0,ncc,ntp0
00374 c$$$ allocate( zzmel(nbloch, nctot+nt0, ncc+ntp0) )
00375 c$$$ if(debug) write(6,('zzzw nkmin nkqmin=',2i5)) nkmin,nkqmin
00376 c$$$ if(onceww(5) ) write(6,*)' nctot ncc=',nctot,ncc

```

```

00377 c$$$c allocate( ppb(nlnmx*nlnmx*mdimx*nclass))
00378 c$$$c ppb=ppbir(:,irot,ispq)
00379 c$$$c call psich_v3 ( nctot,ncc,ntp0,icclass,phase,
00380 c$$$c i cphi_k(1,nkmin),
00381 c$$$c i cphi_kq(1,nkqmin),
00382 c$$$c i ppbir(:,irot,isp_k),!ppb,
00383 c$$$c i nlnmv,nlnmc,mdim,
00384 c$$$c i imdim,iatomp,
00385 c$$$c i mdimx,nlmt0,nbloch,nlnmx,natom,nclass,
00386 c$$$c i icore,ncore,nl,nnc,
00387 c$$$c o zzmel)
00388 c$$$C ... valence
00389 c$$$c if(debug) write(6,('4 zzzqgbbb222 ",3d13.5)') sum(abs(zzmel)),sum(zzmel)
00390 c$$$c !call cputid(0)
00391 c$$$c call psi2b_v3 ( nctot,ncc,ntp0,icclass,phase,
00392 c$$$c i cphi_k(1,nkmin),
00393 c$$$c i cphi_kq(1,nkqmin),
00394 c$$$c i ppbir(:,irot,isp_k),! ppb,
00395 c$$$c i nlnmv, nlnmc,mdim,
00396 c$$$c i imdim,iatomp,
00397 c$$$c d mdimx,nlmt0,nbloch,nlnmx,natom,nclass,
00398 c$$$c o zzmel)
00399 c$$$c if(debug) call cputid(0)
00400 c$$$c if(debug) write(6,('4 zzzqgbbb222 ",3d13.5)') sum(abs(zzmel)),sum(zzmel)
00401 c$$$!TIME1 "after psich_v3"
00402 c$$$!TIME0
00403 c$$$!! --- IPW set
00404 c$$$c call readqg('QGpsi',q+rk(:,k)-qq,ginv, vec_kq, ngp_kq, ngvecp_kq)
00405 c$$$c call readqg('QGpsi', rk(:,k)-qq,ginv, vec_k, ngp_k, ngvecp_k)
00406 c$$$!! ngp_k = ngpn(kp) ! q+k ntp0 in FBZ
00407 c$$$!! ngp_k = ngpn(k) ! k np0 in FBZ
00408 c$$$!! ngc ! q in IBZ
00409 c$$$c ngb = nbloch + ngc ! This is not ngbb for smbasis()=T. oct2005
00410 c$$$c if(ngb/=ngbb) then
00411 c$$$c write(6,*)' x0kf_v4h: ngb ngbb=',ngb,ngbb
00412 c$$$c call rx( 'x0kf_v4h: ngb/=ngbb')
00413 c$$$c endif
00414 c$$$!! q k q+k
00415 c$$$c allocate( zmel(ngb,nctot+ntp0,ncc+ntp0) )
00416 c$$$c allocate( zlp(ngb,ngb) )
00417 c$$$!! ... read plane wave part of eigenfunction. (note isp is opposite).
00418 c$$$c call readgeig(q+rk(:,k)-qq, ngpmx,isp_kq, vec_kq_g, geig_kq)
00419 c$$$c call readgeig( rk(:,k)-qq, ngpmx,isp_k, vec_k_g, geig_k)
00420 c$$$c if(sum(abs(vec_kq_g-vec_kq))>tolqu) then
00421 c$$$c write(6,('vec_kq_g :',3d23.10)') vec_kq_g
00422 c$$$c write(6,('vec_kq :',3d23.10)') vec_kq
00423 c$$$c call rx( 'x0kf_v4hz:vec_kq_g/=vec_kq')
00424 c$$$c endif
00425 c$$$c if(sum(abs(vec_k_g-vec_k))>tolqu) then
00426 c$$$c write(6,('vec_k_g :',3d23.10)') vec_k_g
00427 c$$$c write(6,('vec_k: ',3d23.10)') vec_k
00428 c$$$c call rx( 'x0kf_v4hz:vec_k_g/=vec_k')
00429 c$$$c endif
00430 c$$$!! qdiff = q - qbkp(:) + rk(:,k)
00431 c$$$c qdiff = q - vec_kq + vec_k
00432 c$$$c ! q - (q+k) + k is not zero.
00433 c$$$c ! qc - q1 + q2
00434 c$$$c add = matmul(qbasinv, qdiff)
00435 c$$$c nadd = idint( add + dsign(.5d0,add)) ! write(6,*)' qdif=',qdiff,qbkp(:),rk(:,k)
00436 c$$$c if(sum(abs(add-nadd))>1d-10) call rx( "sexc: abs(add-nadd)>1d-10")
00437 c$$$c zmel = 0d0
00438 c$$$!TIME1 "before melpln2t"
00439 c$$$!TIME0
00440 c$$$!! <Bq Pq1> = < Bq Pqu2| Pqu1> *exp(i2pi nadd)
00441 c$$$c if(ngc/=0) then !Aug2005
00442 c$$$c call melpln2t(ngp_kq, ngvecp_kq ! q+k ; kp ngp_kq 1:ntp0 q-point
00443 c$$$c & , ngp_k, ngvecp_k ! k ; k ngp_k 1:ntp0 occupied
00444 c$$$c & , ngc, nadd,
00445 c$$$c & geig_kq(1:ngp_kq, itps:itps+ntp0-1), ntp0, ! q+k ; kq
00446 c$$$c & geig_k(1:ngp_k, 1:ntp0 ), ntp0, ! ; k
00447 c$$$c i shtv, q, q, symope,qbas,
00448 c$$$c i vec_kq, !qt oct2013
00449 c$$$c o zmel (nbloch+1:nbloch+ngc, nctot+1:nctot+ntp0,ncc+1:ncc+ntp0))
00450 c$$$c endif
00451 c$$$!! == zmel contain O^-1=I|J>^-1 factor. zzmel(J,it,itp)= \sum_I <phi phi|I> O^-1_IJ ==
00452 c$$$c zmel(1:nbloch, 1:nctot+ntp0, 1:ncc+ntp0) =
00453 c$$$c & zzmel(1:nbloch, 1:nctot+ntp0, 1:ncc+ntp0)
00454 c$$$!! k q+k
00455 c$$$c deallocate(zzmel)
00456 c$$$c if(debug) write(6,('4 zzzppp222bbb ",3d13.5)') sum(abs(zmel)),sum(zmel)
00457 c$$$c if(debug) call cputid(0)
00458 c$$$!TIME1 "after melpln2t"
00459 c$$$!TIME0
00460 c$$$!! == zmel conversion on different basis.
00461 c$$$c allocate(zmmm(nmbas)) ! this is also obsolete if USE_GEMM_FOR_SUM
00462 c$$$c if(chipmzr) then !spin moment basis.
00463 c$$$c zmat => zzzr

```



```

00464 c$$$ elseif(nolfco .and. nmbas==1) then !for <e^iq r|x0|e^iq r>
00465 c$$$ zmat => ppovlz
00466 c$$$ else !may2013 this removes O^-1 factor from zmelt
00467 c$$$ allocate(ppovl_(ngb,ngb))
00468 c$$$ ppovl_=0d0
00469 c$$$ do i=1,nbloch
00470 c$$$ ppovl_(i,i)=1d0
00471 c$$$ enddo
00472 c$$$ ppovl_(nbloch+1:nbloch+ngc,nbloch+1:nbloch+ngc)=ppovl
00473 c$$$ if(.not.eibz4x0()) then !sep2014 added for eibz4x0=F
00474 c$$$ ppovl_=matmul(ppovl_,zcousq)
00475 c$$$ endif
00476 c$$$ zmat => ppovl_
00477 c$$$ endif
00478 c$$$!! :: zmelt conversion muplitpled by z zr.
00479 c$$$ if(verbose())>39) write(6,*)'info: USE GEMM FOR SUM (zmelt = zmelt*zmat) in x0kf_v4h.F'
00480 c$$$ allocate( zmelt_tmp(ngb,nctot+nt0,ncc+ntp0) )
00481 c$$$ call zcopy(ngb*(nctot+nt0)*(ncc+ntp0),zmel,1,zmelt_tmp,1)
00482 c$$$ call zgemm('T','N',ngb,(nctot+nt0)*(ncc+ntp0),ngb,(1d0,0d0),
00483 c$$$ zmat,ngb,zmelt_tmp,ngb,(0d0,0d0),zmel,ngb)
00484 c$$$ deallocate(zmelt_tmp)
00485 c$$$ deallocate(zmmm)
00486 c$$$ if(allocated(ppovl_)) deallocate(ppovl_)
00487 c$$$!TIME1 "after matmul zmel"
00488 c$$$c$$$cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00489 c$$$ print *,'xxxxxxxxxxxxx 8829 xxxxxxxxxxxxxxxx'
00490 c$$$ goto 8829
00491 c$$$c if(debug) write(6,('( "4 zzzppp 11l ",3d15.6)') sum(abs(zmel)),sum(zmel)
00492 c$$$cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00493 c$$$
00494 c$$$
00495 c$$$
00496 c$$$
00497 c$$$
00498 c$$$c$$$!march2013--> this if branch of nolfco is now unified to do 25
00499 c$$$c$$$! No LocalFieldCorrection mode
00500 c$$$c$$$ if(nolfco) then !iepsmode==202) then ! just for <exp(iq r)|x0(q,\omega)|exp(iq r)>
00501 c$$$c$$$ do jpm = 1,npm !
00502 c$$$c$$$ do ibib = 1, nbnb(k,jpm) !--- ibib loop
00503 c$$$c$$$ if(jpm==1) then
00504 c$$$c$$$ if( n1b(ibib,k,jpm) <= nbcut.and. n2b(ibib,k,jpm)>nbcut2) then !oct2005
00505 c$$$c$$$ if(iww2) then
00506 c$$$c$$$ write(6,('( ' nband_chi0 nbcut nbcut2 n2b n1b=',4i6)')
nbcut,n2b(ibib,k,jpm),n1b(ibib,k,jpm)
00507 c$$$c$$$ iww2=.false.
00508 c$$$c$$$ endif
00509 c$$$c$$$ cycle
00510 c$$$c$$$ endif
00511 c$$$c$$$ else
00512 c$$$c$$$ if( n2b(ibib,k,jpm) <= nbcut.and. n1b(ibib,k,jpm)>nbcut2) then !oct2005
00513 c$$$c$$$ if(iww2) then
00514 c$$$c$$$ write(6,('( ' nband_chi0 nbcut nbcut2 n2b n1b=',4i6)')
nbcut,n2b(ibib,k,jpm),n1b(ibib,k,jpm)
00515 c$$$c$$$ iww2=.false.
00516 c$$$c$$$ endif
00517 c$$$c$$$ cycle
00518 c$$$c$$$ endif
00519 c$$$c$$$ endif
00520 c$$$c$$$
00521 c$$$c$$$ if( jpm==1.and.n2b(ibib,k,jpm) > nbmx) cycle
00522 c$$$c$$$ if( jpm==2.and.n1b(ibib,k,jpm) > nbmx) cycle
00523 c$$$c$$$
00524 c$$$c$$$ if( n1b(ibib,k,jpm) <= nband) then
00525 c$$$c$$$ it = nctot + n1b(ibib,k,jpm) !valence
00526 c$$$c$$$ else
00527 c$$$c$$$ it = n1b(ibib,k,jpm) - nband !core
00528 c$$$c$$$ endif
00529 c$$$c$$$
00530 c$$$c$$$ if( n2b(ibib,k,jpm) <= nband) then
00531 c$$$c$$$ itp = ncc + n2b(ibib,k,jpm) - itps + 1 !val
00532 c$$$c$$$ if(itp > ncc + nkqmax-itps+1 ) cycle
00533 c$$$c$$$ else
00534 c$$$c$$$ itp = n2b(ibib,k,jpm) - itps + 1 - nband !core
00535 c$$$c$$$ endif
00536 c$$$c$$$
00537 c$$$c$$$ do imb2=1,nmbas
00538 c$$$c$$$ zq02 = zmelt(imb2,it,itp)
00539 c$$$c$$$ do imbl=1,imb2
00540 c$$$c$$$ zq01 = zmelt(imbl,it,itp)
00541 c$$$c$$$ zq0zq0 = dconjg(zq01)*zq02
00542 c$$$c$$$ do iw = ihw(ibib,k,jpm),ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-1
00543 c$$$c$$$ if (iw .gt. nwt) stop "x0kf_v4hz: iw > nwt"
00544 c$$$c$$$ !iww=iw+ihw(ibib,k)-1
00545 c$$$c$$$ imagweight = whw(jhw(ibib,k,jpm)+iw-ihw(ibib,k,jpm))
00546 c$$$c$$$ if(eibzmode) imagweight = nwgt(k)*imagweight
00547 c$$$c$$$ rcxqmean(iw,jpm,imbl,imb2) = ! here we sum over ibib (or n, n') and k.
00548 c$$$c$$$c & rcxqmean(iw,jpm,imbl,imb2) + zq0zq0*imagweight

```

```

00549 c$$$$$ rcxq(imb1,imb2,iw,jpm) = ! here we sum over ibib (or n, n') and k.
00550 c$$$$$ & rcxq(imb1,imb2,iw,jpm) + zq0zq0*imagweight !sum over spin in hx0fp0
00551 c$$$$$ enddo ! iw
00552 c$$$$$ enddo ! imb1
00553 c$$$$$ enddo ! imb2
00554 c$$$$$ enddo ! ----- ibib loop
00555 c$$$$$ enddo ! ----- jpm loop
00556 c$$$$$ deallocate(zmelt,zlp)
00557 c$$$$$ cycle !cycye do 1000 here
00558 c$$$$$ endif
00559 c$$$$$!TIME1 "before jpm ibib loop"
00560 c$$$$$!TIME0
00561 c$$$
00562 c$$$
00563 c$$$
00564 c$$$
00565 c$$$
00566 c$$$ 8828 continue
00567
00568
00569 c$$$ this (ppovlz generation) is moved to hx0fp0 and/or hx0fp0_sc.
00570 c$$$!! === zmelt conversion on different basis.
00571 c$$$ if(chipmzsr) then !spin moment basis.
00572 c$$$ if(allocated(ppovlz)) deallocate(ppovlz)
00573 c$$$ allocate(ppovlz(ngb,nmbas))
00574 c$$$ ppovlz= zsr
00575 c$$$ elseif(nolfco .and. nmbas==1) then !for <e^iqr|x0|e^iqr>
00576 c$$$ continue
00577 c$$$ else !may2013 this removes O^-1 factor from zmelt
00578 c$$$ allocate(ppovl_(ngb,ngb))
00579 c$$$ ppovl_=0d0
00580 c$$$ do i=1,nbloch
00581 c$$$ ppovl_(i,i)=1d0
00582 c$$$ enddo
00583 c$$$ ppovl_(nbloch+1:nbloch+ngc,nbloch+1:nbloch+ngc)=ppovl
00584 c$$$ if(.not.eibz4x0()) then !sep2014 added for eibz4x0=F
00585 c$$$ ppovl_=matmul(ppovl_,zcousq)
00586 c$$$ endif
00587 c$$$ ppovlz = ppovl_
00588 c$$$ deallocate(ppovl_)
00589 c$$$ endif
00590 c$$$ if(allocated(zmel)) deallocate(zmel)
00591 c nbmax= nctot+nt0
00592 c ntqxx= ncc+ntp0
00593 cc call get_zmelt(exchange,q,kx,qibz_k,irot,qbz_kr,kr,isp,
00594 cc & ngc,ngb,nbmax,ntqxx,isp_k,isp_kq)
00595 c
00596 !! -----
00597 !!note: for usual correlation mode, I think nctot=0
00598 !!--- For dielectric funciton, we use irot=1 kvec=rkvec=q
00599 ! < MPB middle | end >
00600 !! q rkvec | q + rkvec
00601 ! nkmin:nt0 | nkqmin:ntp0
00602 ! occ | unocc
00603 ! (nkmin=1)
00604 ! (cphi_k | cphi_kq !in x0kf)
00605 !
00606 !! rkvec= rk(:,k) ! <phi(q+rk,nqmax)|phi(rk,nctot+nmmax) MPB(q,ngb )>
00607 !! qbz_kr= rk(:,k) !
00608 !! qibz_k= rk(:,k) ! k
00609 ngb = nbloch + ngc
00610 !!Get the matrix element zmel ZO^-1 <MPB psi|psi> , where ZO is ppovlz
00611 !! Output is zmel(ngb, nctot+nt0,ncc+ntp0) nkmin:nt0, nkqmin:ntp0
00612 ! nt0=nkmax-nkmin+1 , ntp0=nkqmax-nkqmin+1
00613 call get_zmelt2(exchange,
00614 & q,irot,q,ngc,ngb, !MPB
00615 & nkmin, nkmax, isp_k,nctot, !middle state 1:nt0 --> true index of eigen is
00616 & q+rk(:,k),nkqmin,nkqmax,isp_kq,ncc) !end state 1:ntp0 --> is
00617 nkqmin:nkqmin+ntp0-1
00617 zmel = dconjg(zmel)
00618 allocate( zlp(ngb,ngb) )
00619
00620 8829 continue
00621 if(debug) write(6,('4 zzzppp 222 ",3d15.6)') sum(abs(zmel)),sum(zmel)
00622
00623 !TIME1_1101 "before_get_zmelt2"
00624 !TIME0_1201
00625 c-----
00626 !! zlp = <M_ibg1 psi_it | psi_itp> < psi_itp | psi_it M_ibg2 >
00627 !! zxq(iw,ibg1,igb2) = sum_ibib wwq(iw,ibib)* zlp(ibib, igb1,igb2)
00628 !KINO write(6,('a,i4)')'kino: npm=',npm
00629 !kino 2014-08-13 !$OMP parallel private(it,itp,iww1,iww2,zmelt2,imagweight)
00630
00631 cccccccccccccccccccccccccc
00632 c write(6,('gggx ',3f9.4,x2x,3f9.4)") q+rk(:,k),rk(:,k)
00633 cccccccccccccccccccccccccc

```

```

00634
00635 do 25 jpm = 1, npm !
00636 do 25 ibib = 1, nbnb(k,jpm) !--- ibib loop
00637 !KINO write(6,'(a,5i8)') 'kino: ngb,hw(ibib,k,jpm),ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-1=',
00638 !KINO& ngb,ihw(ibib,k,jpm),ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-1
00639 if(nlb(ibib,k,jpm) <= nband) then
00640 it = nctot + nlb(ibib,k,jpm) !valence
00641 if(it > nctot + nkmax) cycle
00642 else
00643 it = nlb(ibib,k,jpm) - nband !core
00644 endif
00645 if( n2b(ibib,k,jpm) <= nband) then
00646 itp = ncc + n2b(ibib,k,jpm) - itps + 1 !val
00647 if(itp > ncc + nkqmax-itps+1) cycle
00648 else
00649 itp = n2b(ibib,k,jpm) - itps + 1 - nband !core
00650 endif
00651
00652 ccccccccccccccccccccccccccc
00653 c write(6,'(gggx ',2i3)") ib,jb,ebandqr(it),ebandr(itp)
00654 ccccccccccccccccccccccccccc
00655
00656 ccccccccccccccccccccccccccc
00657 c if(itp/=1) cycle
00658 c if(k/=1) cycle
00659 ccccccccccccccccccccccccccc
00660
00661
00662 if(jpm==1) then !nbmx is moved to tetwt5.
00663 c$$$ if(n2b(ibib,k,jpm)>nbmx) then !nbmx
00664 c$$$ if(iww1) then
00665 c$$$ write(6,*)' nband_chi0 nbmx=',nbmx
00666 c$$$ iww1=.false.
00667 c$$$ endif
00668 c$$$ cycle
00669 c$$$ endif
00670 if( nlb(ibib,k,jpm) <= nbcut .and. nbcut2<n2b(ibib,k,jpm) ) then
00671 if(iww2) then
00672 write(6,'( ' nband_chi0 nbcut nbcut2 n2b nlb=',4i6)") nbcut,n2b(ibib,k,jpm),nlb(ibib,k,jpm)
00673 iww2=.false.
00674 endif
00675 cycle
00676 endif
00677 else !jpm==2 -----
00678 c$$$ if( nlb(ibib,k,jpm) > nbmx) then !nbmx
00679 c$$$ if(iww1) then
00680 c$$$ write(6,*)' nband_chi0 nbmx=',nbmx
00681 c$$$ iww1=.false.
00682 c$$$ endif
00683 c$$$ cycle
00684 c$$$ endif
00685 if( n2b(ibib,k,jpm) <= nbcut .and. nbcut2<nlb(ibib,k,jpm) ) then
00686 if(iww2) then
00687 write(6,'( ' nband_chi0 nbcut nbcut2 n2b nlb=',4i6)") nbcut,n2b(ibib,k,jpm),nlb(ibib,k,jpm)
00688 iww2=.false.
00689 endif
00690 cycle
00691 endif
00692 endif
00693
00694 ccccccccccccccccccccccccccc takao variant begin
00695 cc if(takao) then
00696 cc
00697 cc do ic = 1,ngb
00698 cc zlp(1:ngb,ic) =
00699 cc & zmelt(ic,it,itp)*dconjg(zmelt(1:ngb,it,itp))
00700 cc end do
00701 cc ihww = ihw(ibib,k)
00702 cc
00703 clini-----
00704 cc do iw = 1, nhw(ibib,k)
00705 cc rviw = whw(jhw(ibib,k)+iw-1)
00706 cC ... this part dominates the cpu time -----!
00707 c! call zaddr_(zxq(1,1,ihww+iw-1),rviw,zlp,ngb**2)
00708 cc call daxpy(ngb**2*2,rviw,zlp,1,
00709 cc & zxq(1,1,ihww+iw-1),1)
00710 cc enddo
00711 clend-----
00712 c2ini -----
00713 cc call rcxq_zxq(rc1p,zlp,ngb,-1)
00714 cc do iw = 1, nhw(ibib,k)
00715 cc rviw = whw(jhw(ibib,k)+iw-1)
00716 C ... this part dominates the cpu time -----!
00717 ! call zaddr_(rcxq(1,1,ihww+iw-1),rviw,zlp,ngb**2)
00718 cc call daxpy(ngb**2,rviw,rc1p,1,
00719 cc & rcxq(1,1,ihww+iw-1),1)
00720 cc enddo

```

```

00721 c2end -----
00722 cc      else
00723 cccccccccccccccccccccccccc takao variant end
00724
00725 c$$$      if(newanisox.and.eibzmoden==1) then ! This is slow.
00726 c$$$      zmeltx = zmeltx(:,it,ity)
00727 c$$$      zlp=0d0
00728 c$$$      do iegbz =1, nwtg(k) !equivalent points for ieibz
00729 c$$$      !igx,igxt specifies space-group operation (including ID)
00730 c$$$      call rotMPB(zcousq,nbloch,ngbb,q,igx(iegbz,k),igxt(iegbz,k),ginv,zcousqrx)
!zcousq=Rotate_igx(zcousq)
00731 c$$$      zmelty = matmul(zmeltx,zcousqrx)
00732 c$$$      do igb2=1, nbg !.....
00733 c$$$      zmelty2 = zmelty(igb2)
00734 c$$$      do igb1=1,igb2
00735 c$$$      zlp(igb1,igb2) = zlp(igb1,igb2) + dconjg(zmelty(igb1)) * zmelty2
00736 c$$$      enddo
00737 c$$$      enddo
00738 c$$$      enddo
00739 c$$$      else
00740 c$$$      if (ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-1 >nwt) call rx("x0kf_v4hz: iw>nwt")
00741
00742 !kino 2014-08-13 !$OMP do private(zmelty2)
00743 c$$$      do igb2=1, nmbas !.....
00744 c$$$      zmelty2 = zmel(igb2,it,ity) !zmel(igb2,it,ity)
00745 c$$$      do igb1=1,igb2
00746 c$$$      zlp(igb1,igb2) = dconjg(zmel(igb1,it,ity)) * zmelty2
00747 c$$$      zlp(igb1,igb2) = dconjg(zmel(igb1,it,ity)) * zmelty2
00748 c$$$      enddo
00749 c$$$      enddo
00750
00751 !! -----
00752 c$$$      if(crpa) then
00753 c$$$      print *, 'readout readqkm init'
00754 c$$$      if(nlb(ibib,k,jpm) <= nband) then
00755 c$$$      call readpkm4crpa(nlb(ibib,k,jpm), rk(:,k), isp_k, wpw_k) !k nlb
00756 c$$$      else
00757 c$$$      wpw_k=0d0
00758 c$$$      endif
00759 c$$$      if(n2b(ibib,k,jpm) <= nband) then
00760 c$$$      call readpkm4crpa(n2b(ibib,k,jpm), q+rk(:,k), isp_kq, wpw_kq) !kq n2b
00761 c$$$      else
00762 c$$$      wpw_kq=0d0
00763 c$$$      endif
00764 c$$$      write(6,('rrrrrr: nlb wpw_k n2b wpw_kq fac irange=',i5,x,f9.4, i5,x,f9.4, x,f9.4, 2i5))
00765 c$$$      & nlb(ibib,k,jpm),wpw_k, n2b(ibib,k,jpm),wpw_kq, (1d0-wpw_k*wpw_kq),
00766 c$$$      & ihw(ibib,k,jpm), ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-1
00767 c$$$      endif
00768 c$$$cccccccccccccccccccccccccccccccccccc
00769 c$$$ For SrV03 test
00770 c$$$      if(crpa) then
00771 c$$$      wpw_k=0d0
00772 c$$$      if(15<nlb(ibib,k,jpm).and.nlb(ibib,k,jpm)<19) wpw_k=1d0
00773 c$$$      wpw_kq=0d0
00774 c$$$      if(15<n2b(ibib,k,jpm).and.n2b(ibib,k,jpm)<19) wpw_kq=1d0
00775 c$$$      endif
00776 c$$$cccccccccccccccccccccccccccccccccccc
00777
00778
00779 ccccccccccccccccccccccccccccccccccccc
00780 !kino 2014-08-13 !$OMP end do
00781 c$$$ endif
00782
00783 !$OMP parallel private(imagweight)
00784 !$OMP master
00785 !$      if (jpm.eq.1 .and. ibib.eq.nbnb(k,1)) then
00786 !$      write(6,('a,i5,a,i5')) 'OMP parallel iw, threads=', omp_get_num_threads(),
00787 !$      . ' nw=', ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-ihw(ibib,k,jpm)
00788 !$      endif
00789 !$OMP end master
00790 !$OMP do
00791 c$$$      do iw = ihw(ibib,k,jpm),ihw(ibib,k,jpm)+nhw(ibib,k,jpm)-1 !iiw=iw+ihw(ibib,k)-1
00792 c$$$      imagweight = whw(jhw(ibib,k,jpm)+iw-ihw(ibib,k,jpm))
00793 c$$$      if(crpa) imagweight = imagweight*(1d0-wpw_k*wpw_kq)
00794 c$$$      if(eibzmode) imagweight = nwtg(k)*imagweight
00795 c$$$      do igb2=1,nmbas !this part dominates cpu time most time consuming.....
00796 c$$$      do igb1=1,igb2
00797 c$$$      rcxq(igb1,igb2,iw,jpm) = !here we sum over ibib (or n, n') and k.
00798 c$$$      & rcxq(igb1,igb2,iw,jpm) + zlp(igb1,igb2)*imagweight !sum over spin in hx0fp0
00799 c$$$      enddo !igb1
00800 c$$$      enddo !igb2
00801 c$$$      enddo ! iw
00802 !$OMP end do
00803 !$OMP end parallel
00804 25 continue
00805 !kino 2014-08-13 !$OMP end parallel
00806 !TIME1_1201 "after_rcxq"

```

```

00807
00808 c$$$$cccccccccccccccccccccccccccccccccccccccccccc
00809 c$$$$c      if(ipr) then
00810 c$$$          do jpm=1,npm
00811 c$$$              write(6,"(' k jpm sum(rcxq) ngb ngbb=' ,2i5,2d23.15,2i8)")
00812 c$$$          &      k,jpm,sum(rcxq(:, :, jpm)),ngb,ngbb
00813 c$$$          enddo
00814 c$$$          do ib1 =1,ngbb
00815 c$$$              if(ib1<4) then
00816 c$$$                  elseif(ib1>ngbb-3) then
00817 c$$$                  else
00818 c$$$                      cycle
00819 c$$$                  endif
00820 c$$$          do ib2 =1,ngbb
00821 c$$$              if(ib2<4) then
00822 c$$$                  elseif(ib2>ngbb-3) then
00823 c$$$                  else
00824 c$$$                      cycle
00825 c$$$                  endif
00826 c$$$          do iw =1,nwt
00827 c$$$              write(6,"('uuu: k iw ib1 ib2 sum(rcxq)=' ,4i5,4d23.15)")
00828 c$$$          &      k,iw,ib1,ib2,(rcxq(ib1,ib2,iw,1)), (rcxq(ib1,ib2,iw,2))
00829 c$$$          enddo
00830 c$$$          enddo
00831 c$$$          enddo
00832 c$$$$c      endif
00833 c$$$$cccccccccccccccccccccccccccccccccccccccccccc
00834      deallocate(zlp,zmel) !zmelt,zlp)
00835      if(debug) call cputid(0)
00836      if(debug) write(6,*)' end of kloop k jpm=',k,jpm
00837 1000 continue
00838
00839 !! Not need to be symmetrized
00840      if(nolfco .and. nmbas==1) then
00841          write(6,*)' nmbas=1 nolfco=T ---> not need to symmetrize'
00842          goto 9999
00843      endif
00844 !TIME0_1301
00845
00846
00847 !! ==== Hermitianize. jun2012takao moved from dpsion5 ====
00848 c      if(eibzmode) then !comment out sep2014.
00849     do jpm=1,npm
00850     do iw= 1,nwt
00851     do igb2= 1,nmbas !eibzmode assumes nmbas1=nmbas2
00852     do igb1= 1,igb2-1
00853     rcxq(igb2,igb1,iw,jpm) = dconjg(rcxq(igb1,igb2,iw,jpm))
00854     enddo
00855     enddo
00856     enddo
00857     enddo
00858 c      endif
00859 !TIME1_1301 "before_eibzmode_symmetrization"
00860 9999 continue
00861      write(6,"(' --- x0kf_v4hz: end')")
00862      end subroutine x0kf_v4hz
00863
00864
00865 !! -----
00866      subroutine x0kf_v4hz_symmetrize (npm, !ncc,
00867 c      i      ihw,nhw,jhw,whw,nhwtot, ! tetwt5
00868 c      i      n1b,n2b,nbnbx,nbnb,      ! use whw by tetwt5 ,
00869 c      i      q,
00870 c      i      nsp,isp_k,isp_kq, !symmetrize,
00871 c      i      qbas,ginv,lrk,wk,
00872 c      i      mdim,
00873 c      d      nlmt0,nqbz,nctot,
00874 c      d      natom,
00875 c      d      nbloch,nwt,
00876 c      i      iq, ngbb, ngc, ngpmx,ngcmx,
00877 c      i      nqbze, nband,nqibz,
00878 c      o      rcxq,
00879 c      i      nolfco,zzr,nmbas, zcousq,
00880 c      i      chipmzzr,eibzmode,
00881 c      i      ngrp,eibzsym) !, crpa) !nwtg,igx,igxt,
00882 c      use m_readqg,only : readqg
00883 c      use m_readeigen,only: readeval
00884 c      use m_keyvalue,only : getkeyvalue
00885 c      use m_rotmpb,only : rotmpb2
00886 c      use m_readqgcou,only:
00887 c      o qtt_, nqnum
00888 c      use m_pkm4crpa,only : readpkm4crpa
00889 c      use m_zmel,only : get_zmelt2,
00890 c      o zmel !,ppbir ,ppvzlz
00891 !! === symmetrization for EPIBZ mode ===
00892      implicit none
00893      integer(4):: npm,ncc,ngbb,natom,nwt,nsp,isp_k,isp_kq !nlmt0 !,noccx,noccxv

```

```

00894      & ,nl,nclass,nnc,nlnmx,nbloch,iq,nqibz,iatom,nctot,nbm,x,iopen !mdimx,
00895      & ,jpm,ibib,itps,nt0,ntp0,ngp_kq,ngp_k,it,itp,iw,igb2,igb1,ngb
00896      & ,nn,no,isx,iclose,k,nbnbx,nqbz
00897      real(8):: q(3),qbas(3,3),ginv(3,3),ebmx !,rk(3,nqbz),wk(nqbz)
00898      complex (8):: rcxq (nmbas,nmbas,nwt,npm)
00899      complex(8) :: imag=(0d0,1d0),trc,aaa !phase(natom),
00900      integer(4):: ngpmx, ngcmx, nqbze, nband,
00901      &      ngc,nadd(3)
00902 c      integer(4) :: nbnb(nqbz,npm), n1b(nbnbx,nqbz,npm), n2b(nbnbx,nqbz,npm)
00903      logical,parameter:: debug=.false.
00904 c      integer(4):: nhwtot, ihw(nbnbx,nqbz,npm),nhw(nbnbx,nqbz,npm), jhw(nbnbx,nqbz,npm)
00905 c      real(8):: whw(nhwtot)
00906      complex(8) :: zmelt1,zmelt2,zmeltt(ngbb) !.....sf 21May02
00907      real(8) :: imagweight !.....sf 21May02
00908      integer(4)::nocc
00909      real(8):: eband(nband)!,ebandr(nband),ebandqr(nband)
00910      integer(4):: verbose
00911
00912      logical :: nolfco !iepsmode
00913      integer(4):: nmbas, imbl,imb2, imb !nmbas1x !nmbas2,nmbas1,
00914      real(8):: vec_k_g(3),vec_k_g(3),vec_k_q(3),vec_k(3),quu(3),tolq=1d-8,quu1(3),quu2(3) !tolqu=1d-4,
00915      integer(4):: nbcut,nbcut2
00916      logical :: iww1=.true.,iww2=.true.
00917      complex(8):: img=(0d0,1d0)
00918      integer(4):: nkmin, nkmax, nkqmin, nkqmax,nkmax1,nkqmax1
00919      integer(4):: ib1, ib2,      ngcx,ix,iy
00920      complex(8),target :: zxr(ngbb,nmbas) !ppovlz(ngbb,ngbb),
00921      integer:: igb
00922      integer:: ngrp !,nwgtnqbz) !,ngrpt, aiktimereversal(ngrpt),nwgctieibz,ieibz
00923 c      integer:: igx(ngrp*2,nqbz),igxt(ngrp*2,nqbz),ieqbz
00924      logical:: checkbelong,eibzmode, chipmzxr
00925      complex(8):: zcousq(ngbb,ngbb), zcousqc(ngbb,ngbb)
00926      integer:: eibzsym(ngrp,-1:1),neibz,icc,ig,eibzmoden,ikp,i,j,itimer,icount,iele
00927      integer:: irotm,nrotx,ixx,iyy,itt,ntimer, nccc, nxx,iagain,irotm1,irotm2
00928      integer,allocatable:: il(:,:),i2(:,:),nrotx(:)
00929      complex(8),allocatable:: zrotx(:,:),zrr(:,:),zrrc(:,:),zrrc_(:,:),zrrc_(:,:),zmmm(:),zrrx(:,:)
00930 c      complex(8),pointer:: zmat(:,:)
00931      complex(8),allocatable:: rcxq_core(:,:)
00932      complex(8),allocatable:: zcousqr(:,:),rcxq0(:,:),rcxq00(:,:),rcxq000(:,:),rcxqwww(:,:)
00933
00934 c      logical:: eibz4x0
00935 c      logical :: crpa
00936 c      real(8):: wpw_k,wpw_kq
00937 c      real(8):: vec_kcrpa(3),vec_kqcrpa(3)
00938
00939      logical :: exchange=.false.
00940      integer:: irot=1
00941      integer:: ntqxx,nbmax
00942
00943 !! -----
00944 !! == Symmetrizer of EIBZ PRB.81,125102(2010) Eq.(51) july2012takao ==
00945 !! This may be not so effective ---> only for limited cases?
00946 !! --- zrotx(J,J') = <Mbar^k_J| \hat{A}^k_i Mbar^k_J'>. ---
00947 !! We do \sum_i T_alpha_i [ zrotx_i^dagger (I,I') P_I'J' zrom_i(J'J) ]
00948 !! (exactly speaking, we insert conversion matrix between Enu basis and M_I basis).
00949 !!
00950 !! input qin = q
00951 !! \hat{A}^k_i is specified by symops(:,igx),and igxt (-1 for time-reversal).
00952 !! Note that k= \hat{A}^k_i(k) (S_A^k)
00953 !! See Eq.(51) around in PRB81 125102(2010)
00954 !!
00955 c 5000 continue
00956 !! === zmelt conversion ===
00957      if(nolfco .and. nmbas==1) then
00958          write(6,*)' nmbas=1 nolfco=T ---> not need to symmetrize'
00959          goto 9999
00960      endif
00961 !!
00962      if(eibzmode) then
00963          ngb = nbloch + ngc ! This is not ngbb for smbasis()=T. oct2005
00964          if(ngb/=ngbb) then
00965              write(6,*)' x0kf_v4h: ngb ngbb=',ngb,ngbb
00966              call rx('x0kf_v4h: ngb=ngbb')
00967          endif
00968 !TIME0_1401
00969          call iqindx2(q, ginv, qtt_, nqnum, ikp,quu) !to get ikp for timereversal mode
00970 !TIME1_1401 "after_iqindx2"
00971 !TIME0_1501
00972          if(sum(abs(q-quu))>tolq) call rx('x0kf_v2h: eibz 111 q/quu')
00973          neibz = sum(eibzsym(:,1))+sum(eibzsym(:,1))
00974          !itimer=-1 means time reversal. eibzsym(ig,itimer) where ig: space rotation.
00975          write(6,*)' --- goto symmetrization --- ikp neibz q=',2i3,3f12.8)"ikp,neibz,q
00976          call cputid2(' --- x0kf: start symmetrization ',0)
00977
00978 c      allocate(rcxq0(ngb,ngb),rcxq00(ngb,ngb),rcxq000(ngb,ngb),rcxqwww(ngb,ngb))
00979      ntimer=1
00980      if(sum(eibzsym(:,1))>0) ntimer=2 !timereversal case

```

```

00981         allocate(zrotm(ngb,ngb),nrotm(ngrp*2))
00982 !!
00983 c         zcousqinv=zcousq
00984 c         call matcinv(ngb,zcousqinv)
00985
00986 !! == Assemble rotantion matrix zrr,zrrc ==
00987 !! Rotation matrix zrrx can be a sparse matrix.
00988 !! Thus it is stored to "il(nrotmx,nccc),i2(nrotmx,nccc),zrr(nrotmx,icc),nrotm(icc)".
00989 !! See folloings: matmul(rcxqwww,zrrx) is given by
00990 !!         do irotml = 1,nrotm(icc)
00991 !!             rcxq0(:,i2(irotml,icc)) = rcxqwww(:,il(irotml,icc)) * zrr(irotml,i2(irotml,icc))
00992
00993         allocate(zrrx(nmbas,nmbas))
00994         nrotmx = 10000 !trial value
00995 !TIME1_1501 "before_1011"
00996         do 1011 !this loop is only in order to to set large enough nrotmx.
00997 !TIME0_1601
00998             if(allocated(il)) deallocate(il,i2,zrr,zrrc!),zrr_,zrrc_)
00999             nccc=ngrp*2
01000             allocate(il(nrotmx,nccc),i2(nrotmx,nccc),zrr(nrotmx,nccc),zrrc(nrotmx,nccc))
01001             !,zrr_(ngb,ngb,nccc),zrrc_(ngb,ngb,nccc))
01002             il=-99999
01003             i2=-99999
01004             zrr=-99999d0
01005             zrrc=-99999d0
01006             call cputid2(' --- x0kf:11111 :',0)
01007 !TIME1_1601 "allocatezrr"
01008 !!
01009             icc=0
01010             do itimer=1,-1,-2
01011                 if(ntimer==1.and.itimer==--1) exit
01012                 if(itimer==1) itt=1
01013                 if(itimer==--1) itt=2
01014                 do ig=1,ngrp
01015                     if(eibzsym(ig,itimer)==1) then
01016                         icc=icc+1
01017 !TIME0_1701
01018 !! Get rotation matrix zrrx, which can be a sparse matrix. Thus stored to zrr.
01019             call rotmpb2(nbloch,ngb,q,ig,itimer,ginv,zrotm)
01020             if(nolfco.and.chipmzrr) then
01021 !! We assume <svec_I | svec_J >= \delta_IJ, In addition, we use fact that we have no IPW parts in svec.
01022 !! If IPW part exist, we may have to take into account <IPW|IPW> matrix, e.g. as in ppovlz.
01023 !! svec --> zrr
01024                 if(itimer==1) then
01025                     zrrx= matmul(transpose(dconjg(zrr)), matmul(zrotm, zrr))
01026                 else
01027                     zrrx= matmul(transpose(zrr), matmul(dconjg(zrotm), zrr))
01028                 endif
01029             elseif(nolfco) then
01030                 call rx('x0kf_v4h: this case is not implemented xxxxxxxxxxxxxxxx')
01031             else
01032 !! zrotm(J,J') is the rotation matrix = <Mbar^k_J| \hat{A}^k_i Mbar^k_J'>
01033 !! See rotMPB2 defined in readeigen.F.
01034 !! zrrx(mu nu)= dconjg(Zcousq(I, mu)) *zrotm(I,J)* Zcousq(J, nu)
01035 !! zrrx is very sparse matrix. Size is \sim ngb or something.
01036
01037 c$$$             if(itimer==1) then
01038 c$$$                 call matmmsparse(zcousqinv,zrotm,zcousq,zrrx,ngb,ld-8,iele)
01039 c$$$                 ! this means zrrx= matmul(zcousqinv,matmul(zrotm, zcousq))
01040 c$$$             else
01041 c$$$                 call matmmsparse(dconjg(zcousqinv),dconjg(zrotm),zcousq,zrrx,ngb,ld-8,iele)
01042 c$$$                 ! this means zrrx= matmul(dconjg(zcousqinv),matmul(dconjg(zrotm), zcousq))
01043 c$$$             endif
01044
01045             if(itimer==1) then
01046                 zrrx=zrotm
01047 c                 call matmmsparse(zcousqinv,zrotm,zcousq,zrrx,ngb,ld-8,iele)
01048                 ! this means zrrx= matmul(zcousqinv,matmul(zrotm, zcousq))
01049             else
01050                 zrrx=dconjg(zrotm)
01051 c                 call matmmsparse(dconjg(zcousqinv),dconjg(zrotm),zcousq,zrrx,ngb,ld-8,iele)
01052                 ! this means zrrx= matmul(dconjg(zcousqinv),matmul(dconjg(zrotm), zcousq))
01053             endif
01054         endif
01055     endif
01056 !TIME1_1701 "end_matmmsparse"
01057 !TIME0_1801
01058         il(:,icc)=0
01059         i2(:,icc)=0
01060         irotm=0
01061         iagain=0
01062         do ix=1,ngb
01063             do iy=1,ngb
01064                 if(abs(zrrx(ix,iy))>1d-8) then
01065                     irotm=irotm+1
01066                     if(irotm>nrotmx) then

```



```

01067         iagain=1
01068     endif
01069     if(iagain/=1) then
01070         i1(irotm,icc)=ix
01071         i2(irotm,icc)=iy
01072         zrr(irotm,icc) = zrrx(ix,iy)
01073         zrrc(irotm,icc)= dconjg(zrr(irotm,icc))
01074     endif
01075 endif
01076 enddo
01077 enddo
01078 !TIME1_1801 "before_iagain1"
01079 !TIME0_1901
01080     if(iagain==1) then
01081         nrotx=irotx !enlarge allocation and do things again.
01082         write(6,*)' warn:(slow speed) xxxx goto 1011 xxxxxx nrotx+=nrotx+10000 again'
01083         goto 1011
01084         !enlarge nrotx ang try it again.
01085     endif
01086     nrotx(icc)=irotx
01087     if(debug) write(6,*)'ig itimer icc nrotx=',ig,itimer,icc,nrotx(icc) ,iele
01088 !TIME1_1901 "end_ig_itimer_icc_nrotx"
01089     endif
01090 enddo
01091 enddo
01092 exit
01093 1011 continue !only when nrotx overflow.
01094 !TIME0_2001
01095
01096 !! === main part to obtain symmetrized rcxq ===
01097 !! neibz is total number of symmetrization operation.
01098 !!     rcxq is rotated and accumulated; finally divided by neibz
01099     zcousq = dconjg(transpose(zcousq))
01100     if(debug) call cputid2(' --- x0kf:qqqqq222ini:',0)
01101 !$OMP parallel private(rcxq000,icc,itt,icount,rcxqwww,rcxq00,rcxq0,rcxq_core)
01102     allocate(rcxq0(ngb,ngb),rcxq00(ngb,ngb),rcxq000(ngb,ngb),rcxqwww(ngb,ngb),rcxq_core(ngb,ngb))
01103 !$OMP master
01104 !$     write(6,'(a,i5,a,i5)') 'OMP parallel nwt, threads=',omp_get_num_threads(),' nwt=',nwt
01105 !$OMP end master
01106 !$OMP do
01107     do iw=1,nwt
01108     do jpm=1,npm
01109         rcxq000 = 0d0
01110         icc=0
01111         do itimer=1,-1,-2
01112             if(itimer==1) itt=1
01113             if(itimer==-1) itt=2
01114             icount=0
01115             if(itimer==1) then
01116                 rcxqwww = rcxq(:, :, iw, jpm)
01117             else
01118                 rcxqwww = transpose(rcxq(:, :, iw, jpm))
01119             endif
01120             rcxq00 = 0d0
01121             do ig=1,ngrp
01122                 if(eibzsym(ig,itimer)==1) then
01123                     icount=icount+1
01124                     icc=icc+1
01125                     rcxq0 =0d0
01126
01127 c$$$             if(itimer==1) then
01128 c$$$                 do irotx1 = 1,nrotx(icc)
01129 c$$$                 do irotx2 = 1,nrotx(icc)
01130 c$$$                     rcxq0(i2(irotx2,icc),i2(irotx1,icc)) =rcxq0(i2(irotx2,icc),i2(irotx1,icc))
01131 c$$$ & + zrrc(irotx2,icc)* rcxq(i1(irotx2,icc),i1(irotx1,icc),iw,jpm)*zrr(irotx1,icc)
01132 c$$$                     enddo
01133 c$$$                     enddo
01134 c$$$                 else
01135 c$$$                 do irotx1 = 1,nrotx(icc)
01136 c$$$                 do irotx2 = 1,nrotx(icc)
01137 c$$$                     rcxq0(i2(irotx1,icc),i2(irotx2,icc)) =rcxq0(i2(irotx1,icc),i2(irotx2,icc)) !transpose
01138 c$$$ & + zrrc(irotx2,icc)* rcxq(i2(irotx2,icc),i1(irotx1,icc),iw,jpm)*zrr(irotx1,icc)
01139 c$$$                     enddo
01140 c$$$                     enddo
01141 c$$$                 endif
01142
01143 !! Followings are equivalent with
01144 !!     rcxq00= rcxq00 + matmul(zrrc(:, :, icc),matmul(rcxqwww,zrr(:, :, icc)))
01145             do irotx1 = 1,nrotx(icc)
01146                 if(abs(zrr(irotx1,icc))<1d-8) cycle
01147                 rcxq0(:,i2(irotx1,icc)) =rcxq0(:,i2(irotx1,icc)) + rcxqwww(:,i1(irotx1,icc)) * zrr(irotx1,
01148 c$$$ icc)
01149             enddo
01150             do irotx2 = 1,nrotx(icc)
01151                 if(abs(zrrc(irotx2,icc))<1d-8) cycle
01152                 rcxq00(i2(irotx2,icc),:)= rcxq00(i2(irotx2,icc),:) + zrrc(irotx2,icc) * rcxq0(i1(irotx2,
01153 c$$$ icc),:)

```



```

01152                enddo
01153
01154 c                if(itimer==1) then
01155 c                    rcxq000 = rcxq000 + rcxq00
01156 c                else
01157 c                    rcxq000 = rcxq000 + transpose(rcxq00)
01158 c                endif
01159 c
01160 c$$$                do irotm = 1,nrotm(icc)
01161 c$$$                    iyy = il(irotm,icc)
01162 c$$$                    iy = i2(irotm,icc)
01163 c$$$                    rcxq0(:,iy)= rcxq0(:,iy)+ rcxq(:,iyy,iw,jpm)* zrr(irotm,icc)
01164 c$$$                enddo
01165 c$$$                do irotm = 1,nrotm(icc)
01166 c$$$                    iyy = il(irotm,icc)
01167 c$$$                    iy = i2(irotm,icc)
01168 c$$$                    rcxq00(iy,:)= rcxq00(iy,:)+ dconjg(zrr(irotm,icc)) * rcxq0(iyy,:)
01169 c$$$                enddo
01170 c$$$
01171 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01172 c                if(iw==1.and.jpmm==1) then
01173 c                    write(6,"('bbbbbbb ig icc iw jpm rcxq', 4i3, 13d13.6)")
01174 c                &                    ig,icc,iw,jpm, sum(abs(rcxq00)), rcxq00(1,1),sum(abs(rcxqwww)),sum((rcxqwww))
01175 c                endif
01176 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01177
01178                endif
01179            enddo
01180
01181 c$$$                if(itimer==1) then
01182 c$$$                    rcxq000(:, :) = matmul(zcousq,matmul(rcxq00,zcousq))
01183 c$$$c$$$c$$$c      call zgemm("N","N",ngb,ngb,ngb, (1d0,0d0), rcxq00, ngb, zcousq,ngb, (0d0,0d0),
01184 c$$$c$$$c$$$c      call zgemm("N","N",ngb,ngb,ngb, (1d0,0d0), zcousq,ngb, rzc,ngb, (0d0,0d0),
01185 c$$$c$$$c$$$c      rcxq000,ngb)
01186 c$$$c$$$c$$$c      elseif(icount>0) then
01187 c$$$c$$$c$$$c          write(6,*)'qqqqq icount=',icount
01188 c$$$c$$$c$$$c          rcxq000(:, :) = rcxq000(:, :) +
01189 c$$$c$$$c$$$c          transpose(matmul(transpose(zcousq),matmul(rcxq00,dconjg(zcousq))))
01190 c$$$c$$$c$$$c          rcxq000(:, :) = rcxq000(:, :) + matmul(matmul(zcousq,transpose(rcxq00)),zcousq)
01191 c$$$c$$$c$$$c      endif
01192
01193                if(itimer==1) then
01194 c                    rcxq000=rcxq00
01195 c                else
01196 c                    rcxq000=rcxq000+rcxq00
01197 c                endif
01198 c            enddo
01199 c            rcxq_core = rcxq000/neibz
01200 #if 1
01201 !! matmul(rcxq(:, :,iw,jpm),zcousq) fails in ifort 14.0.3.
01202 !! It looks that ifort 14.0.3 has a bug
01203 !! But, zgemm works. So I changed like that.
01204 call zgemm('N','N',ngb,ngb,ngb,(1.0d0,0.0d0),rcxq_core,ngb,zcousq, ngb, (0.0d0,0.0d0),rcxq000,ngb
01205 )
01206 call zgemm('N','N',ngb,ngb,ngb,(1.0d0,0.0d0),zcousq, ngb,rcxq000,ngb, (0.0d0,0.0d0),rcxq_core,
01207 ngb)
01208 rcxq(:, :,iw,jpm) = rcxq_core
01209 #else
01210 rcxq(:, :,iw,jpm) = matmul(zcousq,matmul(rcxq_core,zcousq))
01211 #endif
01212 enddo
01213 !$OMP end do
01214 deallocate(rcxq00,rcxq000,rcxq0,rcxqwww)
01215 !$OMP end parallel
01216 !TIME1_2001 "after_sym_rcxq"
01217 deallocate(zrotm,il,i2)
01218
01219 c$$$                allocate(zcousqr(ngb,ngb,neibz),rcxq0(ngb,ngb),rcxq00(ngb,ngb),rcxqtr(ngb,ngb))
01220 c$$$                icc=0
01221 c$$$                do itimer=1,-1,-2
01222 c$$$                do ig=1,ngb
01223 c$$$                    if(eibzsym(ig,itimer)==1) then
01224 c$$$                        icc=icc+1
01225 c$$$                        if(itimer==1) then
01226 c$$$                            call rotMPB(zcousq,nbloch,ngb,q,ig,itimer,ginv,zcousqr(1,1,icc))
01227 c$$$                        else
01228 c$$$                            time reversal mapping ---
01229 c$$$                            call rotMPB(dconjg(zcousq),nbloch,ngb,q,ig,itimer,ginv,zcousqr(1,1,icc))
01230 c$$$                        endif
01231 c$$$                    endif
01232 c$$$                enddo
01233 c$$$                enddo
01234 c$$$                do iw=1,nwt
01235 c$$$                do jpm=1,npm

```

```

01234 c$$$      rcxq0=0d0
01235 c$$$      icc=0
01236 c$$$c      do itimer=1,1 !1,-1,-2
01237 c$$$      do itimer=1,-1,-2
01238 c$$$      do ig=1,ngrp
01239 c$$$      if(eibzsym(ig,itimer)==1) then
01240 c$$$      icc=icc+1
01241 c$$$      rcxq00(:, :) = matmul(dconjg(transpose(zcousqr(:, :,icc))),
01242 c$$$      & matmul(rcxq(:, :,iw,jpm),zcousqr(:, :,icc)))
01243 c$$$!! time reversal mapping ---
01244 c$$$      if(itimer==1) rcxq00(:, :) = transpose(rcxq00)
01245 c$$$      rcxq0(:, :) = rcxq0(:, :) + rcxq00(:, :)
01246 c$$$      endif
01247 c$$$      enddo
01248 c$$$      enddo
01249 c$$$      rcxq(:, :,iw,jpm)=rcxq0(:, :)/neibz
01250 c$$$      enddo
01251 c$$$      enddo
01252 c$$$      deallocate(zcousqr,rcxq0,rcxq00,rcxqtr)
01253 c$$$      if(debug) call cputid2(' --- qqqqq222end:',0)
01254 c$$$      endif
01255 9999 continue
01256 !kino 2014.08.19 use automatic deallocation, deallocate(cphi_k,cphi_kq,geig_kq,geig_k)
01257 write(6,(' --- x0kf_v4hz_symmetrize: end'))
01258 end subroutine x0kf_v4hz_symmetrize
01259
01260
01261
01262
01263 C=====
01264 subroutine dpsion5 (frhis,nwhis, freqr,nw_w, freqi,niwt,
01265 i realomega, imagomega, !freqr ->frhis ...sf
01266 i rcxq, npm, nw_i,nmbas1,nmbas2,
01267 o zxq,zxqi,
01268 c i nolfco,chipm,schi,isp, rcxqmean,nmbas, !iepsmode, rcxqmean, ! epsmode
01269 i chipm,schi,isp, !No nolfco mode. Apr2012.
01270 i ecut,ecuts)
01271 c o x0mean)
01272 C- Calculate W-v zxqi(on the imaginary axis) and zxq(real axis) from sperctum weight rcxq.
01273 Cr v4 works for timereversal=F (npm=2 case).
01274 Cr See rcxq_zcxq for rcxq, which contains the spectrum weight for given bins along the real-axis.
01275 Cr ! Note that zxq and zxqi are not accumulating
01276 Ci frhis(1:nwhis+1) :: specify histogram bins i-th bin is [frhis(i), frhis(i+1)].
01277 Ci We suppose "freqr(i)=moddle of i-th bin; freqr(0)=0."
01278 Ci (I think called routine hilbertmat itself is not limited by this condition).
01279 Ci freqr (0:nw_w) : Calcualte zxq for these real energies.
01280 Ci freqi (1:niwt) : Calcualte zxqi for these imaginary energies.
01281 Ci realomega : A switch to calculate zxq or not.
01282 Ci imagomega : A switch to calculate zxqi or not.
01283 Ciw rcxq may be altered ---used as work area.
01284 Cio zxq : W-v along the real axis on freqr(0:nw_w)
01285 Cio zxqi: W-v along the imag axis on freqi(niwt)
01286 C!
01287 C! Feb2006: v4 for timereversal=F
01288 C! July2005: v3Add spin chipm mode
01289 C! July2005: This version alter rcxq---it is used as work area.
01290 C! sergey faleev Apr 2002 ; Rebuiled by takao
01291 C-----
01292 implicit none
01293 integer(4):: nw_w,niwt,igb1,igb2, iw,iwp,nwhis,ix,npm,ifxx,nmbas1,nmbas2
01294 real(8) :: freqi(niwt),pi,px,omp,om,om2,oml, !omg2max from hx0fp0
01295 & frhis(nwhis+1), freqr(0:nw_w), aaa,d_omg
01296 logical :: realomega, imagomega
01297 complex(8):: rcxq(nmbas1,nmbas2, nwhis,npm) !sf 13June
01298 c logical :: iepsmode
01299 logical :: chipm
01300
01301 integer(4)::isp,ispix !, nmbas
01302 c complex(8):: rcxqmean(nwhis,npm,nmbas,nmbas) !takao sep2006 add nmbas
01303 C... ecut mode
01304 real(8):: ecut,ecuts,wcute,wcutef,dee,schi
01305 logical ::debug=.false.
01306 real(8),allocatable :: his_l(:),his_r(:),his_c(:)
01307 integer(4) it
01308 real(8):: domega_r,domega_c,domega_l,delta_l,delta_r
01309 real(8),allocatable :: rmat(:, :, :), rmati(:, :, :), rmatt(:, :, :), imatt(:, :, :),
01310 complex(8),allocatable :: rmatic(:, :, :), imattc(:, :, :),
01311 complex(8) :: beta,wfac
01312 complex(8):: zz
01313 complex(8),allocatable :: zxqn(:, :),zxqnl(:, :, :),rx0mean1(:, :, :),rx0mean(:)
01314 complex(8),allocatable:: rrr(:)
01315
01316 integer(4)::nw_i,jpm,ipm,verbose,isgi
01317 c complex(8):: x0mean(nw_i:nw_w,nmbas,nmbas)
01318 complex(8)::
01319 o zxq(nmbas1,nmbas2, nw_i: nw_w), !iw=0 means omg=0,
01320 !iw=1:nw_w corresponds to iw's bit of the frequensy histogram

```

```

01321      o  zxqi(nmbas1,nmbas2,niwt),img !npm), img !zxqi(...,npm) may2006
01322
01323      real(8),allocatable:: ebb(:)
01324      integer(4):: ii,i,ibas1,ibas2
01325      logical :: evaltest !,testttr
01326
01327 c      if(verbose())>89) debug=.true.
01328 c -----
01329      write(6,(' -- dpsion5: start... ',,$))
01330      write(6,(' nw_w nwhis=',2i5)) nw_w,nwhis
01331      if(debug) then
01332          write(6,*)' nmbas1 nmbas2 nwhis npm =', nmbas1,nmbas2,nwhis,npm
01333          write(6,*)' sumchk rcxq=', sum(abs(rcxq))
01334      endif
01335      pi = 4d0*datan(1d0)
01336      img = (0d0,1d0)
01337      call cputid(0)
01338      ispx = isp
01339      if(schi<0) then
01340          ispx = 3-isp !flip
01341      endif
01342
01343 !! Check freqr
01344      if(realomega) then
01345          if( nwhis <= nw_w ) then
01346              write(6,*)nwhis,nw_w
01347              call rx( ' dpsion5: nwhis<=nw_w' )
01348          endif
01349          if( freqr(0)/=0d0 ) call rx( ' dpsion5: freqr(0)/=0d0' )
01350 !! I think current version allows any freqr(iw), independent from frhis.
01351 c$$$      aaa = 0d0
01352 c$$$      if(nw_w>0) then
01353 c$$$          do iw = 1,nw_w
01354 c$$$              aaa = aaa + abs( freqr(iw) - (frhis(iw)+frhis(iw+1))/2d0 )
01355 c$$$              if(debug) write(6,(' iw freqr frhis_m=',i5,2f13.6)" )
01356 c$$$              & iw,freqr(iw), (frhis(iw)+frhis(iw+1))/2d0
01357 c$$$          enddo
01358 c$$$          if(aaa>1d-10)call rx( 'dpsion5:freqr/=frhis_m is not implimented yet')
01359 c$$$      endif
01360      endif !realomega
01361
01362 C-----
01363 !! Each histogram bins are [his_Left, his_Right], and his_Center is middle.
01364 !! his_C(0) is at zero. his_R(0) and his_L(0) are not defined.
01365      if(debug) write(6,*)' dpsion5: RRR 222222222 '
01366      allocate(his_l(-nwhis:nwhis),his_r(-nwhis:nwhis),his_c(-nwhis:nwhis))
01367      his_l(1:nwhis) = frhis( 1: nwhis)
01368      his_r(1:nwhis) = frhis(1+1:1+nwhis)
01369      his_c(1:nwhis) = (his_l(1:nwhis) + his_r(1:nwhis) )/2d0
01370      do iw= 1,nwhis
01371          his_l(-iw) = -his_r(iw)
01372          his_r(-iw) = -his_l(iw)
01373          his_c(-iw) = -his_c(iw)
01374      enddo
01375      his_c(0) = 0d0; his_r(0)=-999; his_l(0)=-999
01376 C
01377      if(debug) write(6,*)'sumchk 111 rcxq=', sum(abs(rcxq))
01378
01379      do iw= 1, nwhis
01380          if(ecut<ld9) then
01381              wfac= wcutef(his_c(iw), ecut,ecuts)
01382          else
01383              wfac= 1d0
01384          endif
01385 ! rcxq is used as work---> rcxq= Average value of Im chi.
01386 ! Note rcxq is "negative" (
01387          do jpm=1,npm
01388              call dscal(2*nmbas1+nmbas2, -wfac/(his_r(iw)-his_l(iw)),rcxq(1,1,iw,jpm),1)
01389          enddo
01390 c      if(debug) write(6,*) 'dpsion5: RRR 7777 iw wfac=',iw,wfac,ecut,ecuts
01391      enddo
01392      if(debug) write(6,*)'sumchk 122 rcxq=', sum(abs(rcxq))
01393
01394 C... Temporary. maybe, we will have better procedure...
01395 ctakao moved this to x0kv_v4h.F jun2012takao
01396 !! hermitianize.
01397 c      if(nmbas1==nmbas2) then !Is this required??? apr2012takao
01398 c          do jpm=1,npm
01399 c              do iw= 1, nwhis
01400 c                  do igb2= 1, nmbas2
01401 c                      do igb1= 1, igb2-1
01402 c                          rcxq(igb2,igb1,iw,jpm) = dconjg(rcxq(igb1,igb2,iw,jpm))
01403 c                      enddo
01404 c                  enddo
01405 c              enddo
01406 c          enddo
01407 c      endif

```

```

01408 ccccccccccccccccccc
01409     if(debug) write(6,*)'sumchk 222 rcxq=', sum(abs(rcxq))
01410
01411     if(evaltest().and.nmbas1==nmbas2) then
01412         write(6, "('hhh --- EigenValues for rcxq -----')")
01413         allocate(ebb(nmbas1))
01414         do jpm= 1,npm
01415             do iw = 1, nwhis
01416                 call diagcvh2(rcxq(:, :, iw, jpm), nmbas1, ebb)
01417                 do ii=1,nmbas1
01418                     write(6, "('hhh1: xxxxxxxxxxxxxxxxx', 2i4)") jpm, iw
01419                     if(abs(ebb(ii))>1d-8.and.ebb(ii)>0)
01420                         & write(6, "('hhh1: jpm iw eb=', 2i4, d13.5)") jpm, iw, ebb(ii)
01421                     enddo
01422                 enddo
01423             enddo
01424         deallocate(ebb)
01425     endif
01426
01427 C--- realomega case
01428     if(realomega)then
01429         write(6,*) " --- realomega --- "
01430         if(npm==1) then
01431             allocate( rmat(0:nw_w, -nwhis:nwhis, npm), rrr(-nwhis:nwhis))
01432             rmat = 0d0
01433             do it = 0, nw_w
01434                 zz = freqr(it) !his_C(it)
01435                 call hilbertmat(zz, nwhis, his_l, his_c, his_r, rrr)
01436                 rmat(it, :, 1) = drealm(rrr)
01437             enddo ; if(debug) write(6,*) 'dpsion5: RRR 5555555555'
01438             allocate( rmatt(0:nw_w, nwhis, npm) )
01439             if( chipm.and.ispx==1 ) then
01440                 rmatt(:, :, 1) = rmat(:, 1:nwhis, 1)
01441             elseif( chipm.and.ispx==2 ) then
01442                 do iw= 1, nwhis
01443                     rmatt(:, iw, 1) = -rmat(:, -iw, 1)
01444                 enddo
01445             else
01446                 do iw= 1, nwhis
01447                     rmatt(:, iw, 1) = rmat(:, iw, 1) - rmat(:, -iw, 1)
01448                 enddo
01449             endif
01450             deallocate(rmat, rrr)
01451         else ! npm==2 case -----
01452             allocate( rmatt(-nw_w:nw_w, nwhis, npm), rrr(-nwhis:nwhis))
01453             rmatt = 0d0
01454             do it = -nw_w, nw_w
01455                 if(it<0) then
01456                     zz = -freqr(-it) !his_C(it)
01457                 else
01458                     zz = freqr(it) !his_C(it)
01459                 endif
01460                 call hilbertmat(zz, nwhis, his_l, his_c, his_r, rrr)
01461                 rmatt(it, :, 1) = drealm(rrr(1:nwhis))
01462                 rmatt(it, :, 2) = -drealm(rrr(-1:-nwhis:-1))
01463             enddo ; if(debug) write(6,*) 'dpsion5: RRR2 5555555555'
01464             deallocate(rrr)
01465         endif
01466         rmatt = rmatt/pi ; if(debug) write(6,*) 'dpsion5: RRR 6666'
01467
01468 !! takao remove if(nolfc) block here.
01469 c write(6,*) " --- realomega dgemm--- "
01470
01471
01472 !! WARN! I think npm==2.and.chipm does not make sense. apr2012.
01473 !!
01474     if(npm==2.and.chipm)
01475         & stop 2013.08.09 kino & stop 'x0kf_v4h:npm==2.and.chipm is not meaningful probably'
01476         & call rx( .and.'x0kf_v4h:npm==2chipm is not meaningful probably')
01477
01478
01479 !! Note rcxq is negative now (converted at the top of this routine !!!
01480     if( chipm .and. ispx==2 ) then
01481         !nothing here
01482         !Since the range of zxq is nw_i=0, we have no area to store negative energy part of chipm.
01483     elseif( chipm ) then
01484         call zaxpy( nmbas1*nmbas2*nw_w, img, rcxq, 1, zxq(1,1,1), 1)
01485     else
01486         zxq = 0d0 ! not accumulating case.
01487         call zaxpy( nmbas1*nmbas2*nw_w, img, rcxq(1,1,1,1), 1, zxq(1,1,1), 1)
01488     endif
01489
01490     if(npm==2) then
01491         do iw=1,nw_w
01492             call zaxpy( nmbas1*nmbas2, img, rcxq(1,1,iw,2), 1, zxq(:, :, -iw), 1)
01493         enddo
01494     endif

```

```

01495
01496         if(npm==1) then
01497             call dgemm('n','t', 2*nmbas1*nmbas2, nw_w+1, nwhis, ld0,
01498             &         rcxq, 2*nmbas1*nmbas2, rmatt, nw_w+1,
01499             &         ld0, zxq, 2*nmbas1*nmbas2 )
01500         elseif(npm==2) then
01501             call dgemm('n','t', 2*nmbas1*nmbas2, npm*nw_w+1, nwhis, ld0,
01502             &         rcxq(1,1,1,1), 2*nmbas1*nmbas2, rmatt(:,1), npm*nw_w+1,
01503             &         ld0, zxq, 2*nmbas1*nmbas2 )
01504             call dgemm('n','t', 2*nmbas1*nmbas2, npm*nw_w+1, nwhis, ld0,
01505             &         rcxq(1,1,1,2), 2*nmbas1*nmbas2, rmatt(:,2), npm*nw_w+1,
01506             &         ld0, zxq, 2*nmbas1*nmbas2 )
01507         else
01508             Cstop2rx 2013.08.09 kino          stop 'dpsion5: npm=1 or 2'
01509             call rx( 'dpsion5: npm=1 or 2' )
01510         endif
01511         deallocate(rmatt)
01512     endif
01513
01514     !! === imagomega case          imatt(niwt -->niwt,npm may2005 ===
01515     if(imagomega) then
01516         allocate( rrr(-nwhis:nwhis))
01517         if(npm==1) then
01518             allocate( rmati(niwt,-nwhis:nwhis,npm))
01519             rmati= 0d0
01520         else
01521             allocate( rmatic(niwt,-nwhis:nwhis,npm))
01522             rmatic = 0d0
01523         endif ; if(debug) write(6,*) 'dpsion5: III 111111155555555555'
01524         do it = 1,niwt
01525             zz = img*freqi(it) !his_C(it)
01526             call hilbertmat(zz, nwhis,his_l,his_c,his_r, rrr) !Im(zz)>0
01527             if(npm==1) then
01528                 rmati(it,:1) = drealm(rrr)
01529             else
01530                 rmatic(it,:1) = rrr
01531             endif
01532         enddo ; if(debug) write(6,*) 'dpsion5: III 555555555555'
01533     !! ==== npm=1 case ====
01534     if(npm==1) then
01535         allocate( imatt(niwt, nwhis,npm) )
01536         do iw= 1,nwhis
01537             imatt(:,iw,1) = rmati(:,iw,1) - rmati(:,iw,1)
01538         enddo
01539         deallocate(rmati,rrr)
01540         imatt = imatt/pi; if(debug) write(6,*) 'dpsion5: III '
01541         call dgemm('n','t', 2*nmbas1*nmbas2, niwt, nwhis, ld0,
01542             &         rcxq, 2*nmbas1*nmbas2, imatt, niwt,
01543             &         0d0, zxqi, 2*nmbas1*nmbas2 )
01544         deallocate(imatt)
01545     !! ==== npm=2 case ====
01546     else
01547         allocate( imattc(niwt, nwhis,npm) )
01548         do iw= 1,nwhis
01549             imattc(:,iw,1) = rmatic(:, iw,1)
01550             imattc(:,iw,2) = - rmatic(:,iw,1)
01551         enddo
01552         deallocate(rmatic,rrr)
01553         imattc = imattc/pi; if(debug) write(6,*) 'dpsion5: IIIc '
01554         call zgemm('n','t', nmbas1*nmbas2, niwt, nwhis, ld0,
01555             &         rcxq(1,1,1,1), nmbas1*nmbas2, imattc(1,1,1), niwt,
01556             &         0d0, zxqi, nmbas1*nmbas2 )
01557         call zgemm('n','t', nmbas1*nmbas2, niwt, nwhis, ld0,
01558             &         rcxq(1,1,1,2), nmbas1*nmbas2, imattc(1,1,2), niwt,
01559             &         ld0, zxqi, nmbas1*nmbas2 )
01560         deallocate(imattc)
01561     endif
01562     deallocate(his_l,his_c,his_r)
01563     write(6,(' ' end dpsion5 ",,$)')
01564     call cputid(0)
01565     end
01566
01567     logical function checkbelong(qin, qall, nq,ieibz) !ieibz is also returned
01568     integer:: nq,ieibz
01569     real(8):: qin(3), qall(3,nq),tolq=1d-8
01570     checkbelong=.false.
01571     do i=1,nq
01572         if(sum(abs(qin-qall(:,i)))<tolq) then
01573             ieibz=i
01574             checkbelong=.true.
01575             return
01576         endif
01577     enddo
01578     end
01579
01580
01581     !!-----

```

```

01582      subroutine hilbertmat (zz,nwhis, his_L,his_C,his_R, rmat)
01583 C- Martix for hilbert transformation, rmat.
01584 Cr zz is real---> no img*delta function part
01585 Cr zz is complex (and Im(zz)>0) : includes all contribution when Im(zz)>eps
01586 Co rmat(-nwhis:nwhis) : rmat(0) is not meaningful.
01587 Ci i-th Histogram bin on real axis are given by [his_L, his_R]. center is his_C.
01588 Cr f(zz) = \int_{-x}^{x(nwhis)} f(x)/(zz-x)
01589 Cr      = \sum_{i=0}^{nwhis} rmat(i)*f(i)
01590 Cr      ,where f(i) is the average value at i-th bin.
01591 C!!! 23May2006 I think
01592 C!!! rmat is -----
01593 C!!! f(zz) = - \int_{-x}^{x(nwhis)} f(x)/(zz-x)
01594 C!!!      = - \sum_{i=0}^{nwhis} rmat(i)*f(i)
01595 C I forgot minus sign in the previous note.
01596 C-----
01597      implicit none
01598      integer(4):: iw,nwhis
01599      complex(8) ::zz,imgepsz
01600      real(8)     :: his_l(-nwhis:nwhis),his_c(-nwhis:nwhis),his_r(-nwhis:nwhis)
01601      complex(8) :: rr_fac(-nwhis:nwhis),rl_fac(-nwhis:nwhis),img=(0d0,1d0)
01602      real(8):: eps=1d-8, epsz=1d-13,delta_r,delta_l,ddr,ddl
01603      complex(8):: domega_c,domega_r,domega_l
01604      complex(8) :: rmat(-nwhis:nwhis)
01605      imgepsz =img*epsz
01606      do iw = -nwhis, nwhis
01607          if(iw==0) cycle
01608          domega_r = zz - his_r(iw) + imgepsz
01609          domega_c = zz - his_c(iw) + imgepsz
01610          domega_l = zz - his_l(iw) + imgepsz
01611          if( abs(domega_c)<eps .or. abs(domega_r)<eps ) then
01612              rr_fac(iw) = 0d0
01613          else
01614              ! rr_fac(his_C(is)) = \int^{his_R}_{his_C} d omega' /(his_C(is) -omega')
01615              rr_fac(iw) = log( abs((domega_r/domega_c)) )
01616              rr_fac(iw) = log( domega_r/domega_c )
01617          endif
01618          if( abs(domega_c)<eps .or. abs(domega_l)<eps ) then
01619              rl_fac(iw) = 0d0
01620          else
01621              ! rl_fac(his_C(is)) = \int^{his_C}_{his_L} d omega' /(his_C(is) -omega')
01622              rl_fac(iw) = log( abs((domega_c/domega_l)) )
01623              rl_fac(iw) = log( domega_c/domega_l )
01624          endif
01625      enddo
01626      rmat=0d0
01627      do iw = -nwhis, nwhis !symmetric version. iw=0 is meaningless
01628          if(iw==0) cycle
01629          c      if(debug) print *, ' it iw=',it, iw
01630          domega_c = zz - his_c(iw)
01631          if(iw== nwhis) then
01632              delta_r = his_r(iw) - his_c(iw)
01633          elseif(iw== -1) then
01634              delta_r = 0d0 - his_c(iw)
01635          else
01636              delta_r = his_c(iw+1) - his_c(iw)
01637          endif
01638          !      if(debug) print *, ' it iw RRR1'
01639          if(iw== -nwhis) then
01640              delta_l = his_c(iw) - his_l(iw)
01641          elseif(iw== 1) then
01642              delta_l = his_c(iw) - 0d0
01643          else
01644              delta_l = his_c(iw) - his_c(iw-1)
01645          endif
01646          !      if(debug) print *, ' it iw RRR2'
01647          !      ddr = (his_r(iw)-his_c(iw))/delta_r
01648          !      ddl = (his_c(iw)-his_l(iw))/delta_l
01649          rmat(iw) = rmat(iw) + rr_fac(iw)*( 1d0-domega_c/delta_r) !+ ddr
01650          if(iw/=nwhis.and.iw/=-1) then
01651              rmat(iw+1) = rmat(iw+1) + rr_fac(iw)*domega_c/delta_r !- ddr
01652          endif
01653          rmat(iw) = rmat(iw) + rl_fac(iw)*( 1d0+domega_c/delta_l) !- ddl
01654          if(iw/=-nwhis.and. iw/=1) then
01655              rmat(iw-1) = rmat(iw-1) - rl_fac(iw)*domega_c/delta_l !+ ddl
01656          endif
01657      cccccccccccccccccccccccccc
01658      c no-derivarive test
01659      c      rmat(iw) = rr_fac(iw) + rl_fac(iw)
01660      cccccccccccccccccccccccccc
01661      enddo
01662      end
01663
01664 c$$$      subroutine reducezmel(aold, ngbo,ngb,nx,
01665 c$$$      i      io, in, nmat, pmat,
01666 c$$$      i      io_q, in_q, nmat_q, pmat_q,
01667 c$$$      o      anew)
01668 c$$$c For given q+G basis, we augment the basis within MT.

```

```

01669 c$$$c For given atom and l prod and prodd at MT boundary (reserved in PPBRD
01670 c$$$ integer(4):: nmat,io(nmat),in(nmat),nmat_q,io_q(nmat),in_q(nmat)
01671 c$$$ complex(8):: aold(ngbo,nx), anew(ngb,nx),pmat(nmat) ,pmat_q(nmat)
01672 c$$$ anew=0d0
01673 c$$$ do ix=1,nmat
01674 c$$$ anew(in(ix), :)
01675 c$$$ & = anew(in(ix), :) + pmat(ix) * aold(io(ix), :)
01676 c$$$ enddo
01677 c$$$ do ix=1,nmat_q
01678 c$$$ anew(in_q(ix), :)
01679 c$$$ & = anew(in_q(ix), :) + dconjg(pmat_q(ix)) * aold(io_q(ix), :)
01680 c$$$ enddo
01681 c$$$ end
01682
01683 real(8) function wcutef(e,ecut,ecuts)
01684 real(8):: e,ecut,ecuts
01685 c wcutef = ld0/( exp((e-ecut)/ecuts)+ ld0)
01686 wcutef = exp( -(e/ecut)**2 ) ! ecuts is not used in this case
01687 end

```

## 4.27 main/hbasfp0.m.F File Reference

### Functions/Subroutines

- program [hbasfp0\\_v2](#)

#### 4.27.1 Function/Subroutine Documentation

##### 4.27.1.1 program hbasfp0\_v2 ( )

Definition at line 1 of file [hbasfp0.m.F](#).

Here is the call graph for this function:

## 4.28 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,nnn,nrx, cutbase,lcutmx,nindxc,
00015 & nindxv,occv,unoccv,occc,unoccc,iclass
00016 use m_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 1 ifphiv(2),ifphic(2), iphiv(2),iphivd(2),iphic(2),iphi(2),iphidot(2),
00022 1 ifev(2),ifevf(2),ibas,ibas1,ic,icx,ifaln,ifinin,iflmt0,ifphi,
00023 1 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):: ired, 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,ired,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     iflmt0 = iopen('LMT0',1,0,0)
00083     if (iflmt0 <= 0) call rx( 'unit file for LMT0 <= 0')
00084     call rgwinf_v3(iflmt0,ifinin,incwfin) ! readin inputs. See use use m_rgwinf_v3,only: ... at the
beginning.
00085     iflmt0= iclose('LMT0')
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,nindexv, occc,unoccc,nindexc,
00135 d nl,nn,nnv,nnn,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 orthogonal
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,nnn,nclass,nsp,natom,
00206 & phitotr(1:nrx,0:nl-1,1:nnn,1:nclass,1:nsp), !core
00207 & nindexc,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 anfsig()
00215      if(laf) then
00216  !!      Check iclass =ibas ; CLASS file contains true classes 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, lcutmx(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

```

## 4.29 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)

### 4.29.1 Function/Subroutine Documentation

#### 4.29.1.1 program hsfp0\_sc ( )

Definition at line 1 of file [hsfp0.sc.m.F](#).

Here is the call graph for this function:

4.29.1.2 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 1228 of file [hsfp0.sc.m.F](#).

Here is the caller graph for this function:

## 4.30 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      !! See papers;
00011      !! [1] T. Kotani and M. van Schilfgaarde, Quasiparticle self-consistent GW method:
00012      !!   A basis for the independent-particle approximation, Phys. Rev. B, vol. 76, no. 16,
00013      !!   p. 165106[24pages], Oct. 2007.
00014      !! [2] T. Kotani, Quasiparticle Self-Consistent GW Method Based on the Augmented Plane-Wave
00015      !!   and Muffin-Tin Orbital Method, J. Phys. Soc. Jpn., vol. 83, no. 9, p. 094711 [11 Pages], Sep. 2014.
00016      !! EIBZ symmetrization;
00017      !! See [3] C. Friedrich, S. Blugel, and A. Schindlmayr,
00018      !! Efficient implementation of the GW approximation within the all-electron FLAPW method,
00019      !! Physical Review B, vol. 81, no. 12, Mar. 2010.
00020      !! -----
00021      !! Usage: This routine is called from a script for QSGW, ecalj/fpgw/exec/gwsc.
00022      !! which calls is as "echo 2|../exec/hsfp0_sc >lsc" when mode=2 (three times in the gwsc).
00023      !! -----
00024      !! mode= 1: exchange      mode SEx, the exchange part of the self-energy
00025      !! mode= 2: correlation mode SEc, the correlated part of the self-energy
00026      !! mode= 3: core exchange mode SExcore
00027      !! xxx mode= 4: plot spectrum function ---See manual ---> this is performed by echo 4|hsfp0
00028      !! -----
00029      !! iSigMode parameter which determines approximation for self-energy is given by GWinput file as iSigMode.
00030      !! iSigMode==0 SE_nn'(ef)+image integr:delta_nn'(SE_nn(e_n)-SE_nn(ef))
00031      !! iSigMode==1 SE_nn'(ef)+delta_nn'(SE_nn(e_n)-SE_nn(ef))
00032      !! xxx not support this mode now ... iSigMode==2 SE_nn'((e_n+e_n')/2)
00033      !! iSigMode==3 (SE_nn'(e_n)+SE_nn'(e_n'))/2 <--- this is mainly used
00034      !! iSigMode==5 delta_nn' SE_nn(e_n)
00035      !! Output file contain hermitean part of SE for energies to be real
00036      !! (for example, hermitean conjunction of SE_nn'(e_n) means SE_nn'n(e_n')^* )
00037      !! -----
00038      !! History: We learned so much from LMT0-ASA codeds developed by F.Aryasetiawan.
00039      !! -----
00040      use m_readfermi, only: readfermi, ef
00041      use m_readqg, only: readqg, readngmx
00042      use m_readeigen, only: init_readeigen, init_readeigen2, readeval, lowesteval
00043      use m_read_bzdata, only: read_bzdata,
00044      & nqzbz, nqibz, nqbwz, nteti, ntetf
00045      & n1, n2, n3, qbas, ginv, qbz, wibz, qibz, wibz, qbwz, idtetf, iblbz, idteti
00046      & nstar, irk, nstbz, ngrp2=>ngrp
00047      use m_genallcf_v3, only: genallcf_v3,
00048      & nclass, natom, nspin, nl, nn, ngrp,
00049      & nlmt0, nlnmx, nctot, niw, !nw_input=>nw,
00050      & alat, delta, deltaw, esmr, symgrp, clabl, iclass, !diw, dw,
```

```

00053      & invg, il,in,im,nlnm,
00054      & plat, pos,z,ecore, symgg, konf,nlnx, iantiferro
00055      use m_keyvalue,only: getkeyvalue
00056
00057 !! Base data to generate matrix elements zmel*. Used in "call get_zmelt".
00058      use m_rdp,only: rdp,      !"call rdp" generate following data.
00059      & nblocha,lx,nx,ppbrd,mdimx,nbloch,cgr
00060 !! Generate matrix element for "call get_zmelt".
00061      use m_zmel,only:      ! folloiwng data set are stored in this module in the main routin,
00062                          ! and used when call get_zmelt, get_zmelt2.
00063      & nband,itq,ngcmx,ngpmx,
00064      & mlat,tlat,shtvg, ntq, ppbir
00065 !! antiferro condition. only laf is used, after 'call anfcond()'
00066      use m_anf,only: anfcond,
00067      & laf
00068 !! subroutine only
00069      use m_sxcfcsc,only: sxcfcfal3_scz
00070 !! MPI
00071      use m_mpi,only:
00072      & mpi_initialize,mpi_real8send,mpi_real8recv,mpi_send_iv,mpi_recv_iv,mpi_sxcfc_rankdivider,
00073      & mpi_finalize,mpi_root,mpi_broadcast,mpi_rank,mpi_size,mpi_allreducesum,
00074      & mpi_consoleout,
00075      & mpi_barrier
00076
00077      implicit none
00078 !! -----
00079 !!      real(8),parameter :: ua = 1d0 ! constant in w(0)exp(-ua^2*w'^2) to take care of peak around w'=0
00080 c-----
00081 !!!      test switches to calculate the self-energy based on an another separation of \Sigma.
00082 !!!      \Sigma = \Sigma_{sx} + \Sigma_{coh} + \Sigma_{img axis} + \Sigma_{pole} by Hedin PR(1965)A785
00083 !!!      I found COH term has inevitably poor accuracy.
00084      logical ::tetra, tetra_hsfp0,
00085      & screen = .false.,      ! \Sigma_{sx} for mode 1 and
00086      & \Sigma_{img axis} + \Sigma_{pole} for mode 2
00087      & cohtest= .false.      ! \Sigma_{coh}. mode swich is not required.
00088 c      & , tetra = .false. ! test switch for tetrahedron method test.
00089 c      ! tetra=T is only effective for exchange=T case.
00090 c      ! Tetrahedron mehod for correlation is a bit
00091 ! difficult and I gave up for a while.
00092 ! If you want to calculate with tetra=T for exchange, you
00093 ! have to uncomment tetra related part in
00094 ! sxcfc.f, and a part calling sxcfc in this routine. Note wtet wtetef!
00095 ! They sometimes cause array destruction if you run tetra=T without comment them.
00096
00097 c      real(8) :: shtw
00098      integer::
00099      & ixc,iopen,ifhbed, nprecb,mrecb,mrece,nlmtot,nqbzt, !nband,
00100      & ibas,ibasx,nxx,ifgpnt,ifwd,
00101      & nprecx,mrecl,nblochpmx2,nwp,niwt, nqnum,nblochpmx, !mdimx,nbloch
00102      & noccxv,maxocc,noccx,ifvcfpout,igall,iaf, !ntq, !ifrcw,ifrcwi,
00103      & i,k,nsipinx, nq,is,ip,iq,idxk,ifoutsex,iclose,nq0i,ig,
00104      & mxkp,nqibzxx,ntet,nene,iqi, ix,iw,
00105      & nlnx4,invr,ivsum, ifoutsec, !niwx,
00106      & ifsec(2)
00107      & ,ifxc(2),ifsex(2), ifphiv(2),ifphic(2),ifec,ifexsp(2),
00108      & ifsex2(2),ifsec2(2),      !out S_nn'
00109      & ifsecmg(2),ndble=8
00110      real(8) :: pi,tpia,vol,voltot,rs,alpha,
00111      & qfermi,efx,vaIn,efnew,edummy,efz,qm,xsex,egex,edummyd(1),
00112      & zfac1,zfac2,dscdw1,dscdw2,dscdw,zfac
00113      logical :: lgall,laff,lntq
00114      real(8),allocatable :: q(:, :)
00115
00116      integer,allocatable ::
00117      & ngvecp(:, ), ngvecc(:, ),iqib(:, ),
00118      & kount(:, )
00119      real(8),allocatable:: vxcfp(:, :, ),
00120      & wqt(:, )q0i(:, ),
00121      & eqt(:, ),
00122      & ppbrdx(:, :, :, :, :, ),
00123      & eq(:, ),
00124      & eqx(:, :, ),eqx0(:, :, ),ekc(:, ),coh(:, )
00125      complex(8),allocatable:: zsec(:, :, )
00126 c
00127      logical :: legas
00128      real(8) :: rydberg,hartree
00129      real(8):: qreal(3), ntot,nocctotg2,trip1,xxx(3,3)
00130      logical ::nocore
00131
00132 c      space group infermation
00133      integer,allocatable :: iclasst(:), invgx(:)
00134 c      tetra
00135      real(8),allocatable :: qz(:, ),qbxzx(:, ),wbzxz(:, ),wtet(:, :, :, ),
00136      & eband(:, :, ), ene(:)
00137      integer,allocatable :: idtetx(:, ),idtet(:, ),ipq(:)
00138      & ,iene(:, :, ),ibzx(:)
00139      integer ::ib,ixq,igp,iii,ivsumxxx,isx,iflegas, iqpnnum

```

```

00140 c
00141 real(8),allocatable :: eexl(:,:),exspl(:,:),qqexl(:,:,:)
00142 integer,allocatable:: nspex(:,:),ieord(:),itexl(:,:,:)
00143 real(8) :: qqex(1:3), eex,exsp,eee, exwgt,deltax0
00144 integer :: itmx,ipex,itpex,itex,nspexmx,nnex,isig,iex,ifexsp
00145 & ,ifexspxx,ifefsm, nq0ix,ifemesh,nz
00146 character(3) :: charnum3
00147 character(12) :: filenameeex
00148 logical :: exspwrite=.false.
00149 character*8 xt
00150
00151 integer :: isigmode,ifinin ,idummy
00152
00153 real(8),allocatable:: omega(:)
00154 real(8) :: ebm(2)
00155 integer:: nbm(2)
00156
00157 real(8):: volwgt
00158
00159 integer:: incwfin
00160 real(8),allocatable::freqx(:),freqw(:),wwx(:)
00161
00162 integer:: ngpnl,mrecg,ngcnl
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:: ixend,ixini
00188 integer:: l2nl,igrp,kx,kr
00189 logical :: iprintx,tiii,timereversal, eibz4sig,tiiout
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 integer:: nw
00211 real(8)::dwdummy
00212 c-----
00213 call mpi__initialize() ! MIZUHO-IR
00214 call date_and_time(values=timevalues)
00215 write(6, '(a,9i5)') 'dateandtime1=',mpi__rank,timevalues(1:8)
00216 !TIME0_0000
00217 !TIME0_0010
00218 hartree=2d0*rydberg()
00219 hermitianw=.true.
00220 if(cohstest) then !currently not used (may need fixing if necessary)
00221 screen = .true.
00222 ixc = 2; nz=0
00223 open(671,file='COH')
00224 elseif(mpi__root) then
00225 write(6,*) ' --- Choose modes below -----'
00226 write(6,*) ' Sx(1) Sc(2) ScoreX(3) '

```

```

00227     write(6,*) ' [option --- (+ QPNT.{number} ?)] '
00228     write(6,*) ' Add 1000, eg, 1001 is diagonal only mode for one-shot Z=1'
00229     write(6,*) ' --- Put number above ! -----'
00230     call readin5(ixc,nz,idummy)
00231     write(6,*) ixc
00232   endif
00233   call mpi__broadcast(ixc)
00234   call mpi__broadcast(nz)
00235   if(mpi__root) call headver('hsfp0_sc',ixc)
00236   write(ixcc,('mode=',i4.4))ixc
00237
00238   if(ixc>1000) then          !selected QP
00239     ixc=mod(ixc,1000)
00240     selectqp=.true.
00241     diagonly=.true.
00242     hermitianw=.false.
00243     write(6,*) "---- Diagonal-only mode. jobsw=5; see description at the top of sxcf_fal2.sc.F."
00244     write(6,*) "---- This is the same as one-shot calculaiton with iSigMode5 in Gwinput."
00245   endif
00246
00247   call mpi__consoleout('hsfp0_sc'//trim(ixcc))
00248   write(6,*) ' ixc nz=',ixc, nz
00249   if(ixc=0) call rx( ' --- ixc=0 --- Choose computational mode!')
00250
00251  !! === readin BZDATA. See gwsrsrc/rwbzdata.f ===
00252  !! See use m_read_bzdata,only: at the top of this routine
00253   call read_bzdata()
00254   write(6,*)' nqbz =' ,nqbz
00255   write(6,*)' nqibz ngrp=' ,nqibz,ngrp2
00256   call pshprt(60)
00257
00258  !! === readin GWIN and LMT0, then allocate and set datas. ===
00259  !! See use m_genallcf_v3,only: at the top of this routine
00260  c      nwin = 0          !Readin nw from NW file
00261  c      efin=-999d0      !not readin EFERMI
00262  if(ixc=3) then; incwfin= -2 !core exchange mode
00263  else      ; incwfin= -1 !use 7th colmn for core at the end section of GWIN
00264  endif
00265  call genallcf_v3(incwfin) ! module m_genallcf_v3. See use m_genallcf in this routine
00266  if(ngrp/= ngrp2) call rx( 'ngrp inconsistent: BZDATA and LMT0 GWIN_V2')
00267  esmref=esmr
00268
00269  !! iSigMode
00270  call readd_isigma_en(ifynin,isigmode) !reading self-energy mode parameter from file 'Gwinput'
00271  if(diagonly) isigmode=5
00272
00273  !! Get maximums
00274  call getnemx8(nbmxx,ebmxx) !Get maximums takao 18June03
00275  !!      nbmx1 ebmx1: to set how many bands of <i|sigma|j> do you calculate.
00276  !!      nbmx2 ebmx2: to restrict num of bands of G to calculate G \times W
00277  !!      ebmx2 nbmx2 are not used. For safe, strange number is supplied here.
00278  nbmx(2)=9999999
00279  ebmx(2)=1d10
00280  write(6,(' nbmx ebmx from Gwinput=',i8,d13.5)) nbmx(1),ebmx(1)
00281
00282  !!Caution! WE ASSUME iclass(iatom)= iatom (because of historical reason)
00283  if (nclass /= natom ) call rx( ' hsfp0: nclass /= natom ')
00284  write(6,*)' hsfp0_sc: end of genallcf_v3'
00285  call pshprt(30)
00286  pi = 4d0*datan(1d0)
00287  tpia = 2d0*pi/alat
00288  c      call dinv33(plat,1,xxx,vol)
00289  c      voltot = dabs(vol)*(alat**3)
00290  voltot = abs(alat**3*tripl(plat,plat(1,2),plat(1,3)))
00291  c      shtw = 0d0
00292  tetra= tetra_hsfp0()
00293  !! if(esmr<1d-5) shtw=0.01d0 ! Ferdi's shift to avoid resonance effect(maybe), I used this until sep2012
00294
00295  c$$$!! ef is taken as rs for the empty-sphere test case of legas=T case
00296  c$$$!! HOMOGENIOUS GAS code. Usually not used. Need fixing if necessary.
00297  c$$$!! Keep this just as a memo.
00298  c$$$      legas = .false.
00299  c$$$      if(.false.) then
00300  c$$$          INQUIRE (FILE = 'LEGAS', EXIST = legas)
00301  c$$$          if(legas) then          !!! test for electron gas case.
00302  c$$$              write(6,*)' find LEGAS. legas = ',legas
00303  c$$$              iflegas = 2101
00304  c$$$              open (iflegas,file='LEGAS')
00305  c$$$              read(iflegas,*)rs
00306  c$$$              close(iflegas)
00307  c$$$              alpha = (9*pi/4d0)**(1d0/3d0)
00308  c$$$              qfermi = alpha/rs
00309  c$$$              efx = qfermi**2
00310  c$$$              valn = efx**1.5d0*voltot/3d0/pi**2
00311  c$$$              write (6,*)' ##### egas test mode legas=T ##### given rs = ',rs
00312  c$$$              write (6,*)' egas Exact Fermi momentum qf = ', qfermi
00313  c$$$              write (6,*)' egas Exact Fermi energy Ef = ', efx

```

```

00314 c$$$      if(tetra) call rx( 'legas You have to give ef of tetrahedron')
00315 c$$$      endif
00316 c$$$      endif
00317 c$$$!!
00318      if(ixc==1) then
00319          exchange=.true.
00320          write(6,*) ' --- Exchange mode --- '
00321          if(mpi__root) then
00322              ifxc(1) = iopen('XCU'//xt(nz),1,-1,0)
00323              ifsex(1) = iopen('SEXU'//xt(nz),1,-1,0)
00324              ifsex2(1)= iopen('SEX2U',0,-1,0) !out SEX_nn'
00325              if (nspin == 2) then
00326                  ifxc(2) = iopen('XCD'//xt(nz),1,-1,0)
00327                  ifsex(2) = iopen('SEXD'//xt(nz),1,-1,0)
00328                  ifsex2(2)= iopen('SEX2D',0,-1,0) !out SEX_nn'
00329              endif
00330          endif
00331 c          INQUIRE (FILE = 'EXspTEST', EXIST = exspwrite)
00332 c          if(exspwrite) then
00333 c              write(6,*)'--- Find EXspTEST ExspectrumWrite=',exspwrite
00334 c              write(6,*)'--- esmr is chosen to be 2d0 Ry'
00335 c              esmr= 2d0
00336 c              do is=1,nspin
00337 c                  ifexsp(is) = iopen('EXSP.'//char(48+is),1,-1,0)
00338 c              enddo
00339 c          endif
00340      elseif(ixc==2) then
00341          exchange=.false.
00342          write(6,*) ' --- Correlation mode --- '
00343          if(cohatest) write(6,*) ' COH calculation mode. Results in COH'
00344          if(mpi__root) then
00345              ifsec(1) = iopen('SECU'//xt(nz),1,-1,0) ! output files
00346              ifsec2(1)= iopen('SEC2U',0,-1,0) !out SEC_nn'
00347              if (nspin == 2)
00348              .   ifsec(2) = iopen('SECD'//xt(nz),1,-1,0)
00349              .   ifsec2(2)= iopen('SEC2D',0,-1,0) !out SEC_nn'
00350              endif
00351      elseif(ixc==3) then
00352          exchange=.true.
00353          esmr=0d0
00354          write(6,*) ' --- CORE Exchange mode --- '
00355          if(mpi__root) then
00356              ifsex(1) = iopen('SEXcoreU'//xt(nz),1,-1,0)
00357              ifsex2(1)= iopen('SEXcore2U',0,-1,0) !out SEXcore_nn'
00358              if (nspin == 2) then
00359                  ifsex(2) = iopen('SEXcoreD'//xt(nz),1,-1,0)
00360                  ifsex2(2)= iopen('SEXcore2D',0,-1,0) !out SEXcore_nn'
00361              endif
00362          endif
00363      !! spectrum function mode, we do not use ixc==4
00364 c      elseif(ixc==4) then
00365 c          write(6,*) ' --- Spectrum function Sigma(\omega) mode --- '
00366 c          exchange=.false.
00367 c          ifsecomg(1) = iopen('SEComgU'//xt(nz),1,-1,0) ! output files
00368 c          if (nspin == 2)
00369 c          .   ifsecomg(2) = iopen('SEComgD'//xt(nz),1,-1,0)
00370 c          else
00371 c              call rx( ' hsfp0: Need input (std input) 1(Sx) 2(Sc) or 3(ScoreX)!')
00372 c          endif
00373
00374 c--- Neglect core is NoCore exists -----
00375 c inquire(file='NoCore',exist=nocore)
00376 c if(nocore) nctot=0
00377
00378      write(6,*) ' --- computational conditions --- '
00379      write(6,*) '      deltax =',f13.6)') deltax
00380 c      write(6,*) '      ua      =',f13.6)') ua
00381      write(6,*) '      esmr      =',f13.6)') esmr
00382      write(6,*) '      alat voltot =',2f13.6)') alat, voltot
00383
00384      !! read dimensions of wc,b,hb
00385      ifhbed = ifile_handle() ! ifhbed = iopen('hbe.d',1,0,0)
00386      ! ifile_handle() search unused file handle
00387      open(ifhbed,file='hbe.d',status='old')
00388      read (ifhbed,*) nprecb,mrecb,mrece,nlmtot,nqbzt, nband,mrecg
00389      close(ifhbed) !isx = iclose ('hbe.d')
00390      if (nprecb == 4) call rx( 'hsfp0: b,hb in single precision')
00391      !!
00392      call init_readEigen(ginv,nspin,nband,mrece) !initialization of readEigen
00393      ! required for readEigen readchpi readgeig.
00394
00395      !! === Get space group information ===
00396      !! True class information in order to determine the space group,
00397      !! because the class in the generated GW file is dummy. (iclass(ibas)=ibas should be kept).
00398      ificlass=ifile_handle()
00399      open (ificlass,file='CLASS')
00400      allocate(iclass(natom),invgr(ngrp))

```



```

00401      & ,miat(natom,ngrp),tiat(3,natom,ngrp),shtvg(3,ngrp))
00402      write(6,*)' --- Readingin CLASS info ---'
00403      do ibas = 1,natom
00404          read(ificlass,*) ibasx, iclasst(ibas)
00405          write(6, "(2i10)") ibasx, iclasst(ibas)
00406      enddo
00407      close(ificlass)
00408  !! Get space-group transformation information. See header of mptaouof.
00409      call mptaouof(symgg,ngrp,plat,natom,pos,iclasst
00410      o ,miat,tiat,invvx,shtvg ) !note: miat,tiat,shtvg are defined in m_zmel.
00411      if(verbose())>=40 write (*,*)' hsf0.sc.m.F: end of mptaouof'
00412
00413  !! ==== Get array size to call rdpp can call rdpp to generate base data for get_zmel ====
00414      call getsrdpp2( nclass,nl,nxx)
00415      call readngmx('QGpsi',ngpmx)
00416      call readngmx('QGcou',ngcmx)
00417      write(6,*)' max number of G for QGpsi and QGcou: ngcmx ngpmx=',ngcmx,ngpmx
00418      allocate(ngvecp(3,ngpmx),ngvecc(3,ngcmx))
00419      call readqg('QGpsi',qibz(1:3,1),ginv, quu,ngpn1, ngvecp)
00420      call readqg('QGcou',qibz(1:3,1),ginv, quu,ngcn1, ngvecc)
00421      deallocate(ngvecp,ngvecc)
00422      write(6,*) ' end of read QGcou'
00423  !! ppbrd = radial integrals
00424  !! cgr = rotated cg coeffecients.
00425      call rdpp(nxx, nl, ngrp, nn, nclass, nspin, symgg,qbas)
00426  ! output: nblocha, lx, nx, ppbrd , mdimx, nbloch, cgr are stored in m_rdp.
00427      call pshprt(60)
00428
00429  !! Readin WV.d
00430      if(.not.exchange.or.(exchange.and.screen)) then !screen means screened exchange case
00431          ifwd=ifile_handle() ! ifwd = iopen('WV.d',1,-1,0)
00432  !direct access files WVR and WVI which include W-V.
00433          open(ifwd,file='WV.d')
00434          read (ifwd,*) nprecx,mrecl,nblochpmx,nwp,niwt, nqnum, nw_i
00435          write(6,('( Readin WV.d =', 10i8)") nprecx,mrecl,nblochpmx,nwp,niwt, nqnum, nw_i
00436          close(ifwd) !ifwd =iclose('WV.d')
00437          call checkeq(nprecx,ndble)
00438          nw = nwp-1
00439          if(niwt /= niw) call rx( 'hsfp0_sc: wrong niw')
00440
00441  !! Energy mesh; along real axis. Read 'freq_r'
00442  !! NOTE nw_i=nw for non-timereversal case.
00443  !! nw_i=0 for time-reversal case.
00444  !! NOTE: We assume freq_r(i) == -freq_r(-i) in this code. feb2006
00445  !! NOTE: this program assumes freq_r(iw)=freq_r(-iw). freq_r(iw <0) is redundant.
00446  c      write(6,('( niw nw dw =" ,2i6,f13.6)') niw,nw,dw
00447          ififr=ifile_handle()
00448          open(unit=ififr,file='freq_r')
00449          read(ififr,*)nwxx
00450          if(nwxx/= nw+1) call rx( ' freq_r nw /=nw')
00451          allocate(freq_r(nw_i:nw)) !freq_r(0)=0d0
00452          do iw= nw_i,nw
00453              read(ififr,*) freq_r(iw)
00454          enddo
00455          close(ififr)
00456          if(nw_i/=0) then
00457              if(nw/= -nw_i) call rx( "sxcf_fal3_scz: nw/=-nw_i")
00458              if(freq_r(0)/=0d0) call rx( "sxcf_fal3_scz: freq_r(0)/=0")
00459              if( sum(abs( freq_r(1:nw)+freq_r(-1:-nw:-1)))/=0)
00460              & call rx( "sxcf_fal3_scz: freq_r /= -freq_r")
00461          endif
00462      endif
00463
00464  !! efermi by tetrahedron. this can be overwritten
00465  c      ifief=ifile_handle()
00466  c      open(ifief,file='EFERMI')
00467  c      read(ifief,*) ef
00468  c      close(ifief)
00469      call readefermi()
00470
00471      if(tetra) goto 201 !tetra is experimental. usually =F.
00472
00473  !!== Determine Fermi energy ef for given valn (legas case), or corresponding charge given by z and konf.==
00474  !! When esmr is negative, esmr is given automatically by efsimplef.
00475  c      write(6, "(a,f12.6)")' --- READIN ef from EFERMI. ef=',ef
00476      legas=.false.
00477      call efsimplef2a(nspin,wibz,qibz,ginv,
00478      i nband,nqibz
00479      i ,konf,z,nl,natom,iclass,nclass
00480      i ,valn, legas, esmref, !!! valn is input for legas=T, output otherwise.
00481      i qbz,nqbz ! index_qbz, n_index_qbz,
00482      o ,efnew)
00483      if(ixc/=3) ef = efnew
00484      eftrue = efnew
00485
00486  !! ==== check total ele number ====
00487      ntot = nocctotg2(nspin, ef,esmr, qbz,wbz, nband,nqbz)

```



```

00488     write(6,*)' ef      =' ,ef
00489     write(6,*)' esmr    =' ,esmr
00490     write(6,*)' valn    =' ,valn
00491     write(6,*)' ntot    =' ,ntot
00492
00493 !! == Core-exchange case. ef means just below the valence eigenvalue (to take only core in sxcf).==
00494     if(ixc==3) then
00495         ef = lowesteval() -ld-3 !lowesteval(nspin,nband,qbz,nqbz) - ld-3 !lowesteb was
00496         call getkeyvalue("GWinput","EXonly",wex,default=0d0)
00497         if(wex==0d0) then
00498             exonly=.false.
00499         else
00500             exonly=.true.
00501             write(6,*)' exonly=T ecore shift: ecore---> ecore-100'
00502             ecore = ecore-100.0
00503         endif
00504         write(6,"(a)")' CoreEx mode: We change ef as ef=lowesteval-ld-3, slightly below the bottom of
valence.'
00505         write(6,"(a,f13.5,i5,i5)")' CoreEx mode: ef nspin nctot=' ,ef,nspin,nctot
00506         do ix=1,nctot
00507             write(6,"(i4,x,d13.5,x,d13.5)") ix,(ecore(ix,is),is=1,nspin)
00508         enddo
00509 c         if(maxval(ecore(:,1:nspin))>ef) then !ef is bottom of valence.
00510 c             call rx('hsfp0 ixc=3: ecore>evalence. ')
00511 c         endif
00512     endif
00513 201 continue
00514
00515     call init_readeigen2(mrecb,nlmt0,mrecg) !initialize m_readeigen
00516
00517 !! Read q-points and states
00518     nspinmx = nspin
00519     if(selectqp .and. mpi__root) then
00520         call getkeyvalue("GWinput",<QPNT> ,unit=ifqpnt,status=ret)
00521         lqall = .false.
00522         laff = .false.
00523         call readx(ifqpnt,10)
00524         read (ifqpnt,*) iqall,iaf
00525         if (iqall == 1) lqall = .true.
00526         if (iaf == 1) laff = .true.
00527         call readx(ifqpnt,100)
00528         if (lqall) then !all q-points case
00529             nq = nqibz
00530             allocate(q(3,nq))
00531             call dcopy(3*nqibz,qibz,1,q,1)
00532         else
00533             call readx(ifqpnt,100)
00534             read (ifqpnt,*) nq
00535             allocate(q(3,nq))
00536             do k = 1,nq
00537                 read (ifqpnt,*) i,q(1,k),q(2,k),q(3,k)
00538             enddo
00539         endif
00540         nspinmx = nspin
00541         if (laff) nspinmx =1
00542         close(ifqpnt)
00543     else
00544 ! q-points. bzcase()=1
00545         nq = nqibz
00546         allocate(q(3,nq))
00547         q(:,1:nq) = qibz(:,1:nq) !call dcopy (3*nqibz,qibz,1,q,1)
00548     endif
00549 !!
00550     call mpi__broadcast(nq)
00551     if(mpi__root) then
00552         do dest=1,mpi__size-1
00553             call mpi__real8send(q,3*nq,dest)
00554         enddo
00555     else
00556         call mpi__real8recv(q,3*nq,0)
00557     endif
00558 !! antiferro case. Only calculate up spin
00559     call anfcond()
00560     if(laf) nspinmx=1
00561     call mpi__broadcast(nspinmx)
00562
00563
00564 !! Determine ntq. See also in sxcf_fal.sc.F ntq should be common for all ixc modes.
00565 !! FIX NTQ during iteration by the file NTQ 15jun2015
00566 !!
00567 !! Determine nbandmx. Moved from sxcf_fal2.sc.F.
00568 !!!! count number of band to calculate.
00569 !! I think it it better to determine nbandmx in a manner within LDA
00570 !! (need to care degeneracy...).
00571     allocate(nbandmx(nq,nspinmx))
00572     if(mpi__root) then
00573         inquire(file='NTQXX',exist=Intq)

```

```

00574
00575     ifih = ifile_handle()
00576     open(ifih,file='NTQXX')
00577 !!   Get ntq
00578     if(lntq) then
00579         read(ifih,*) nband_r,nq_r,ntq
00580         if(nband_r/=nband.or.nq_r/=nq) then
00581             rewind ifih
00582             lntq=.false.
00583         endif
00584     endif
00585     if(.not.lntq) then
00586         ntq=0
00587         allocate(eqt(nband))
00588         do is = 1,nspin
00589             do ip = 1,nq
00590                 call readeval(qibz(1,ip),is, eqt)
00591                 do iband=1,nband
00592                     ntq = max(iband,ntq)
00593                     if(eqt(iband)-eftrue>ebmx(1)) exit
00594                 enddo
00595             enddo
00596         enddo
00597         ntq = min(ntq, nbmx(1))
00598         deallocate(eqt)
00599         write(ifih,"(3i10)") nband,nq,ntq
00600     endif
00601 !!   Get ntqxx(iq,isp) and nbandmx
00602     allocate(eqt(nband))
00603     do is = 1,nspinmx
00604         do ip = 1,nq
00605             call readeval(qibz(1,ip),is, eqt)
00606             if(lntq) then
00607                 read(ifih,*) ntqxx    ! ntqxx = ntq !jun2016
00608             else
00609                 ntqxx = 0
00610                 do i = 1,ntq
00611                     if(eqt(i)-eftrue<ebmx(1)) ntqxx =ntqxx  + 1
00612                 enddo
00613                 ntqxx = min(ntqxx, nbmx(1))
00614                 write(ifih,"(i10)") ntqxx
00615             endif
00616             if(ntqxx<nband) then ! reduce ntqxx when band tops are degenerated.
00617                 do i=ntqxx,1,-1
00618                     if(eqt(i+1)-eqt(i)<ld-2) then !ld-2 is a tol to check degeneracy.
00619                         ntqxx=i-1
00620                     else
00621                         exit
00622                     endif
00623                 enddo
00624             endif
00625             nbandmx(ip,is) = ntqxx !number of bands to be calculated
00626         enddo
00627     enddo
00628     deallocate(eqt)
00629     close(ifih)
00630 endif
00631 call mpi_broadcast(ntq)
00632 !!
00633 do is=1,nspinmx
00634     if(mpi_root) then
00635         print *, 'is nbandmx(:,is)=' ,is,nbandmx(:,is)
00636         do dest=1,mpi_size-1
00637             call mpi_send_iv(nbandmx(1:nq,is),dest)
00638         enddo
00639     else
00640         call mpi_recv_iv(nbandmx(1:nq,is),0)
00641     endif
00642 enddo
00643
00644 !! trivial case of itq itq(i)=i
00645     allocate (itq(ntq))
00646     do i = 1, ntq
00647         itq(i) = i !itq is used also in hsfp0.m.F
00648     enddo
00649     do iq=1,nq
00650         write(6,(' Target iq q=" ,i6,3f9.4')iq,q(:,iq)
00651     enddo
00652
00653 !! read LDA eigenvalues
00654 c     allocate(omega(ntq))
00655     allocate(exq(ntq,nq,nspin),eqx0(ntq,nq,nspin),eqt(nband))
00656     do is = 1,nspin
00657         do ip = 1,nq
00658             call readeval(q(1,ip),is,eqt)
00659             eqx0(1:ntq,ip,is) = eqt(itq(1:ntq))
00660             eqx(1:ntq,ip,is) = rydberg()*(eqt(itq(1:ntq))- eftrue)

```

```

00661         enddo
00662     enddo
00663     deallocate(eqt)
00664
00665     write (6,*) ' ***'
00666     write (6,6700) nspin,nq,ntq
00667 6700 format (1x,3i4,' nspin nq ntq')
00668     write (6,6501) is,nbloch,ngpnl,ngcnl,nqbz,nqibz,ef,deltaw,alat,ef,esmr
00669 6501 format (' spin =',i2,' nbloch ngp ngc=',3i4
00670 & ', nqbz =',i6,' nqibz =',i6,' ef=',f10.4,' Rydberg'
00671 & ',d23.16,' <= deltaw(Hartree)'
00672 & ',d23.16,' <= alat'
00673 & ',d23.16,' <= ef '
00674 & ',d23.16,' <= esmr')
00675 c    call winfo(6,nspin,nq,ntq,is,nbloch,ngpnl,ngcnl,nqbz,nqibz,ef,deltaw,alat,esmr)
00676 !!-----
00677 !!    LDA exchange-correlation
00678 !!-----
00679     if(ixc==1) then
00680         allocate( vxcfp(ntq,nq,nspin) )
00681         call rsexx(nspin,itq,q,ntq,nq, ginv, vxcfp) !add ginv july2011
00682         if(mpi__root) then
00683             do is = 1,nspinmx
00684                 write (ifxc(is),*) '=====
00685                 write (ifxc(is),")(' LDA exchange-correlation : is=',i3)")is
00686                 write (ifxc(is),*) '=====
00687                 call winfo(ifxc(is),nspin,nq,ntq,is,nbloch
00688 & ,ngpnl,ngcnl,nqbz,nqibz,ef,deltaw,alat,esmr)
00689                 write (ifxc(is),*) ' ***'
00690                 write (ifxc(is),")("a") ' jband iq ispin
00691 &qvec
00692 &eigen-Ef (in eV)
00693 &LDA XC (in eV)'
00694                 ifoutsex = ifxc(is)
00695                 write(6,*)
00696                 do ip = 1,nq
00697                     do i = 1,ntq
00698                         write(ifoutsex,"(3i5,3d24.16,3x,d24.16,3x,d24.16)")
00699 & itq(i),ip,is, q(1:3,ip), eqx(i,ip,is),
00700 & vxcfp(i,ip,is)
00701 & 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).
00702 & write(6,"(' j iq isp=' i3,i4,i2,' q=' ,3f8.4,
00703 & ' eig=' ,f10.4,' Sxc(LDA)=' ,f10.4)")
00704 & itq(i),ip,is, q(1:3,ip), eqx(i,ip,is),
00705 & vxcfp(i,ip,is)
00706                     endif
00707                 end do
00708             end do
00709             if(is==1) isx = iclose('XCU'//xt(nz))
00710             if(is==2) isx = iclose('XCD'//xt(nz))
00711         enddo ! end of spin-loop
00712     endif !MPI__root
00713     deallocate(vxcfp)
00714 endif
00715
00716 !! Offset Gamma point QOP
00717     write(6,*) 'reading QOP'
00718     ifiq0p=file_handle()
00719     open (ifiq0p,file='QOP')
00720     read (ifiq0p,"(i5)") nq0i
00721     if(.not.exchange) call checkeq(nqibz+nq0i-1,nqnum)
00722     write(6,*) ' *** nqibz nq0i_total=', nqibz,nq0i
00723     allocate( wgt(1:nq0i),q0i(1:3,1:nq0i) )
00724 c    read (101,"(d24.16,3x, 3d24.16)" ) ( wgt(i),q0i(1:3,i),i=1,nq0i)
00725     nq0ix = nq0i
00726     do i=1,nq0i
00727         read (ifiq0p,* ) wgt(i),q0i(1:3,i)
00728         if(wgt(i)==0d0 ) nq0ix = i-1
00729     enddo
00730     nq0i = nq0ix ! New nq0i July 2001
00731     write(6,*) ' Used k number in QOP =', nq0i
00732     write(6,"(i3,f14.6,2x, 3f14.6)" )(i, wgt(i),q0i(1:3,i),i=1,nq0i)
00733     close(ifiq0p)
00734     allocate( wgt0(nq0i,ngrp) )
00735     call getkeyvalue("GWinput","allq0i",allq0i,default=.false.) !S.F.Jan06
00736     call q0iwgt3(allq0i,symgg,ngrp,wgt,q0i,nq0i, !S.F.Jan06
00737 o wgt0) ! added allq0i argument
00738     if (nq0i/=0 ) write(6,*) ' *** tot num of q near 0 =', 1/wgt0(1,1)
00739     write(6,"(' sum(wgt0) from QOP=' ,d14.6)")sum(wgt0)
00740 c$$$ if(bzcase()==2) then
00741 c$$$ wgt0= wgt0*wgt0p()/dble(nqbz)
00742 c$$$ write(6,"('bzcase=2: sum(wgt0_modified )=' ,d14.6)")sum(wgt0)
00743 c$$$ endif
00744
00745 !! Pointer to optimal product basis
00746 c    allocate(imdim(natom))

```

```

00747 c      call indxmmdm (nblocha,nclass,iclass,natom,
00748 c      o imdim )          !in m_zmel
00749      if(niw/=0) then
00750 !! Generate gaussian frequencies x between (0,1) and w=(1-x)/x
00751      allocate(freqx(niw),freqw(niw),wwx(niw)) !,expa(niw))
00752      call freq0lx(niw,      !ua,
00753      o      freqx,freqw,wwx)          !,expa)
00754      endif
00755
00756 c$$$!! ----- write energy mesh for check -----
00757 c$$$      ifemesh = iopen('emesh.hsfp0'//xt(nz),1,-1,0)
00758 c$$$      deltax0 = 0d0
00759 c$$$      if(MPI__root) then
00760 c$$$      call writeemesh(ifemesh,freqw,niw,freq,nw,deltax0)
00761 c$$$      endif
00762
00763 !! === readin Vcoud and EPSwklm for newaniso()=T ===
00764      ifidmlx = iopen('EPSwklm',0,0,0)
00765      read(ifidmlx) nq0ix,lxklm
00766      if(nq0i/=nq0ix) then
00767      write(6,*)'nq0i from EPSwklm /= nq0i',nq0i,nq0ix
00768      call rx( 'nq0i from EPSwklm /= nq0i')
00769      endif
00770      allocate( dmlx(nq0i,9))
00771      allocate( epinvq0i(nq0i,nq0i) )
00772      allocate( wklm((lxklm+1)**2))
00773      read(ifidmlx) dmlx, epinvq0i
00774      read(ifidmlx) wklm
00775      ifidmlx = iclose('EPSwklm')
00776
00777 c----tetra block is experimental. unused usually. -----
00778      if(tetra) then
00779 c      --- get tetrahedron
00780 c      mxkp = n1*n2*n3
00781 c      allocate( qbzx(3*mxkp),wbzx(mxkp),ipq(mxkp) )
00782 c      call bzmesh (plat,qbasmc,n1,n2,n3,w(igrp),ngrp,ipq,
00783 c      .      qbzx,wbzx,nqibzx,mxkp)
00784 c      allocate(idtetx(0:4,mxkp*6))
00785 c      call tetirr(qbasmc,n1,n2,n3,ipq,nqibz,ntet,
00786 c      .      idtetx)
00787 c      allocate(idtet(0:4,ntet))
00788 c      idtet(0:4,1:ntet) = idtetx(0:4,1:ntet)
00789 c      deallocate(idtetx,qbzx,wbzx,ipq)
00790 c
00791 c      nene = ntq*nq*nspin ! for energy points.
00792 c      if(exchange) nene=0
00793 c      allocate(wtet(nband,nspin,nqibz,0:3*nene),
00794 c      &      eband(nband,nspin,nqibz), qz(3,nqibz),nstar(nqibz),
00795 c      &      iene(3*ntq,nq,nspin), ene(0:3*nene) ) ! pointer for
00796 c      allocate(wtet(nband,nspin,nqibz,0:0),
00797 c      &      eband(nband,nspin,nqibz), qz(3,nqibz) ) ! pointer for
00798 c      call dcopy(3*nqibz,qibz,1,qz,1)
00799 c      do is = 1,nspin      !Readin eband
00800 c      do iqi = 1,nqibz
00801 c      iq      = idxx (qz(1:3,iqi),qbz,nqibz)
00802 c      call rwdl (ifev(is), iq, nband, eband(:,is,iqi))
00803 c      call readeval(qz(1:3,iqi),is, eband(:,is,iqi))
00804 c      enddo
00805 c      enddo
00806 c      wtet(nband,nsp,nqibz,iene) where
00807 c      the energy pointer as iene(itp,ip,ispin) corresponding its energy value.
00808 c      ene(0) = ef
00809 c      if(.not.exchange) then
00810 c      ix = 0
00811 c      do is = 1,nspin
00812 c      do ip = 1,nq
00813 c      do i = 1,ntq
00814 c      do iw = -1,1
00815 c      ix = ix+1
00816 c      iene(3*i+iw-1,ip,is) = ix
00817 c      ene(ix) = eqx0(i,ip,is) + 2.d0*(dble(iw)-shtw)*deltaw
00818 c      enddo
00819 c      enddo
00820 c      enddo
00821 c      enddo
00822 c      endif
00823 c      do ix = 0,3*nene
00824 c      ene(ix) = ene(ix)-ld-15 ! to avoid coincidence
00825 c      call bzints2(n1,n2,n3,eband,wtet(:, :, :, ix),nqibz,nband,nband,
00826 c      .      nspin,edummy,edummy,edummy,1,ene(ix),2,ntet,idtet)
00827 c      enddo
00828 c      volwgt = (3d0 - nspin) / ntetf ! ntetf was =6*n1*n2*n3
00829 c      call bzints2x(volwgt,eband,wtet(:, :, :, 0),nqibz,nband,nband,
00830 c      .      nspin,edummy,edummy,edummy,1,ef,2,nteti,idteti)
00831 c      ntot= sum(wtet)
00832 c      if(legas) then
00833 c      write(6, "(' tetra=T ef ntot nexact ratio=',15f12.6) ) ef,ntot

```

```

00834 c      &      , ef**1.5d0/3d0/pi**2*voltot, ef**1.5d0 /3d0/pi**2*voltot/ntot
00835 c      else
00836 c      write(6, "(' tetra=T ef nvalence)=' ,15f12.6)") ef,ntot
00837 c      endif
00838 c      write(6, "(' tetra=T ef nvalence)=' ,15f12.6)") ef,ntot
00839 c      if(nspin==1) wtet = wtet/2d0
00840 c      do iq1 = 1,nqibz
00841 c      wtet(:, :,iq1, :) = wtet(:, :,iq1, :)/nstar(iq1)
00842 c      enddo
00843 c      deallocate( eband, qz, ene ) ! pointer for
00844 c -- ibzx denote the index of k{FBZ for given k{1BZ.
00845 c      allocate(ibzx(nqibz))
00846 c      call invkibzx(irk,nqibz,ngroup,nqibz,
00847 c      o ibzx)
00848 c      else
00849 c      allocate(wtet(1,1,1,1), iene(1,1,1,1)) !dummy
00850 c      endif
00851 c ---- end of tetra section -----
00852 c      iii=ivsumxxx(irk,nqibz*ngroup)
00853 c      write(6,*) " sum of nonzero iirk=",iii, nqibz
00854
00855
00856 !!-----
00857 !!      calculate the the self-energy SEx(ip) or SEc(ip)
00858 !!-----
00859 !! eibz4sig() is EIBZ symmetrization or not...
00860 c      if(eibz4sig()) then
00861 c      allocate(nwgt(nqibz,1:nq), igx(ngroup*2,nqibz,nq))
00862 c      allocate(igxt(ngroup*2,nqibz,nq), eibzsym(ngroup,-1:1,nq))
00863 c      iqxini=1
00864 c      iqxend=nq
00865 c      write(6, "('TimeRevesal switch = ',11)") timereversal()
00866 c      call eibzgen(nq,symgg,ngroup,q(:,iqxini:iqxend),
00867 c      & iqxini,iqxend,qbz,nqibz,timereversal(),ginv,iprintx,
00868 c      o nwgt,igx,igxt,eibzsym,tiiout)
00869 !! Check timereversal is required for symmetrization operation or not. If tiii=timereversal=F is enforced,
00870 !! the symmetrization procedure in x0kf_v4h becomes a little time-consuming.
00871 c      tiii=.false. !Enforce no time reversal. time reversal not yet...
00872 c      write(6,*) 'NOTE:TimeReversal not yet implemented in hsf0.sc.m.F'
00873 c      write(6, "('== goto eibzgen == used timereversal=',11)") tiii
00874 c      iprintx=.false.
00875 c      if(mpi__root) iprintx=.true.
00876 c      call eibzgen(nq,symgg,ngroup,q(:,iqxini:iqxend),
00877 c      & iqxini,iqxend,qbz,nqibz,tiii,ginv,iprintx,
00878 c      o nwgt,igx,igxt,eibzsym,tiiout)
00879 c      call PBindex(natom,lx,l2nl,nx) !all input. this returns requied index stored in arrays in m_pbindex.
00880 ! PBindex: index for product basis. We will unify this system; still similar is used in ppbaftp_v2.
00881 c      call readqgcou() !no input. Read QGcou and store date into variables.
00882 c      call Spacegroupprot(symgg,ngroup,plat,natom,pos) ! all inputs.
00883 c      do iq=iqxini,iqxini
00884 c      do ibz=1,200
00885 c      if(nwgt(ibz,iq)/=0) then
00886 c      write(6, "('yyyl: ',i8,2x,25(i3,i2))") ibz,(igx(i,ibz,iq),igxt(i,ibz,iq),i=1,nwgt(ibz,iq))
00887 c      endif
00888 c      enddo
00889 c      enddo
00890 c      endif
00891
00892 !! == irkip control paralellization ==
00893 !! We have to distribute non-zero irkip into processes (nrank).
00894 !! When irkip(nqibz,ngroup,nq,nspinmx)/=0, we expect grain-size
00895 !! for each job of (iqibz,igrp,iq,isp) is almost the same.
00896 !! Our pupose is to calculate zsec(itp,itpp,iq).
00897 !! Thus we need to set up communicator (grouping) MPI__COMM_iqisp(iq,isp) to do all_reduce.
00898 !! (for given zsec(iq,isp), we take sum on zsec for (iqibz,igrp) by all_reduce.)
00899 !! ---
00900 !! NOTE: in future, we will further extend irkip for itp and itpp
00901 c      allocate(irkip_all(nspinmx,nqibz,ngroup,nq)) !this is global
00902 c      allocate(nrkip_all(nspinmx,nqibz,ngroup,nq)) !this is global
00903 c      allocate(nrkip(nspinmx,nqibz,ngroup,nq)) !this is global
00904 c      if(eibz4sig()) then
00905 c      nrkip_all=0
00906 c      irkip_all=0
00907 c      is=1 ! not spin dependent
00908 c      do iqq=1,nq
00909 c      irkip_all(is, :, :,iqq)=irk
00910 c      do kx=1,nqibz
00911 c      do igrp=1,ngroup
00912 c      kr = irk(kx,igrp) !ip_all(is,kx,igrp,iqq) !kr is index for qbz (for example, nonzero # of kr
00913 is 64 for 4x4x4)
00913 c      if(kr==0) cycle
00914 c      if(nwgt(kr,iqq)/=0) then
00915 c      irkip_all(is,kx,igrp,iqq)= irk(kx,igrp)
00916 c      nrkip_all(is,kx,igrp,iqq)= nwgt(kr,iqq)
00917 c      endif
00918 c      write(6,*) ' iqq kr irk =',iqq,kr,irkip_all(is,kx,igrp,iqq),nrkip_all(is,kx,igrp,iqq)
00919 c      enddo

```

```

00920         enddo
00921     enddo
00922 C     do iqg=1,nq
00923 C         write(6,"('iq=' ,i4,' # of EIBZ: Used(TimeR 1 or -1)=' ,i3,'+',i3+')")
00924 C         & iqg,sum(eibzsym(:, :,iqg)),sum(eibzsym(:,1,iqg)),sum(eibzsym(:, -1,iqg))
00925 C         write(6,"('eibz: iqg sum(nrkip_all)=nqbz ' ,i3,3f11.5,3i8)")
00926 C         & iqg,q(:,iqg),sum(nrkip_all(is, :,iqg)),nqbz
00927 C         do kx=1,nqibz
00928 C             do igrp=1,nggrp
00929 C                 kr = irkip_all(is,kx,igrp,iqg) !kr is index for qbz
00930 C                 if(kr/=0) write(6,"(' ' ,i8,3f11.5,i8,2x,25(i4,i2))")
00931 C                 & kr,qbz(:,kr),nrkip_all(is,kx,igrp,iqg)
00932 C                 & , (igx(i,kr,iqg),igxt(i,kr,iqg),i=1,nwgt(kr,iqg))
00933 C             enddo
00934 C         enddo
00935 C! ! Probably partial group symmetrization is enough. But it may not reduce computational time so
much.
00936 C     enddo
00937 C     if(nspinmx==2) then
00938 C         irkip_all(2, :, :, :) = irkip_all(1, :, :, :)
00939 C         nrkip_all(2, :, :, :) = nrkip_all(1, :, :, :)
00940 C     endif
00941 C     else ! not eibz4sig
00942 C         do is = 1,nspinmx
00943 C             do iqg=1,nq
00944 C                 irkip_all(is, :, :, iqg) = irk
00945 C             enddo
00946 C         enddo
00947 C     endif
00948
00949 !! -- ppb= <Phi(SLn,r) Phi(SL'n',r) B(S,i,Rr)>
00950 allocate( ppbir(nlnmx*nlnmx*mdimx*nclass,nggrp,nspin))
00951 do irot = 1,nggrp
00952     do ispin = 1,nspin
00953         call ppbafp_v2(irot,nggrp,ispin,nspin,
00954             i il,in,im,nlnm, !w(i_mnl),
00955             i nl,nl,nclass,nlnmx,
00956             i mdimx,lx,nx,nxx, !Bloch wave
00957             i cgr, nl-1, !rotated CG
00958             i ppbrd, !radial integrals
00959             o ppbir(:,irot,ispin) !this is in m_zmel
00960     enddo
00961 enddo
00962
00963 !! MPI RankDivider for iqibz and irot cycle in sxcf.
00964 !! nrkip is weight corresponding to irkip for a node.
00965 allocate(irkip(nspinmx,nqibz,nggrp,nq)) !local
00966 call mpi__sxcf_rankdivider(irkip,irkip_all,nspinmx,nqibz,nggrp,nq) ! MIZUHO-IR
00967 nrkip = nrkip_all ! we don't need to change this for MPI case.
00968 ! It just need to distribute non-zero irkip.
00969 !!
00970 nlnx4 = nlnx**4 ! niwx = max0 (nw+1,niw) !nw --->nw+1 feb2006
00971 allocate( kount(nqibz,nq),zsec(ntq,ntq,nq), coh(ntq,nq) )
00972 !TIME1_0010 "main:before2000loop"
00973 do 2000 is = 1,nspinmx
00974 !TIME0_0020
00975 if(mpi__root) then
00976     if(exchange) then
00977         write(ifsex2(is)) nspin, nq, ntq,nqibz,nqibz, nl,n2,n3
00978         write(ifsex(is),*) '=====
00979         write(ifsex(is),"('Self-energy exchange SEx(q,t): is=',i3)") is
00980         write(ifsex(is),*) '=====
00981         call winfo(ifsex(is),nspin,nq,ntq,is,nbloch,ngpn1,
00982             & ngcn1,nqibz,nqibz,ef,deltaw,alat,esmr)
00983         write (ifsex(is),*) ' *** '
00984         write (ifsex(is),"(a)") ' jband iq ispin '//
00985         & ' qvec eigen-Ef (in eV) exchange (in eV)'
00986     elseif(ixc==2) then
00987         write(ifsec2(is)) nspin, nq, ntq ,nqibz,nqibz ,nl,n2,n3
00988         write(ifsec(is),*) '=====
00989         write(ifsec(is),"('Self-energy correlated SEc(qt,w): is=',i3)") is
00990         write(ifsec(is),*) '=====
00991         call winfo(ifsec(is),nspin,nq,ntq,is,nbloch,ngpn1,
00992             & ngcn1,nqibz,nqibz,ef,deltaw,alat,esmr)
00993         write (ifsec(is),*) ' *** '
00994         write (ifsec(is),"(a)") ' jband iq ispin '//
00995         & ' qvec eigen-Ef (in eV) '//
00996         & 'Re(Sc) 3-points (in eV) '//
00997         & ' In(Sc) 3-points (in eV) Zfactor(=1)'
00998     endif
00999 endif
01000 zsec = 0d0
01001 coh = 0d0
01002 kount = 0
01003 if(ixc==3.and.nctot==0) goto 2001 !make dummy SEXcore
01004 !! dummy to overlaid -check bounds sep2014
01005 if(size(ecore)==0) then

```

```

01006         deallocate(ecore)
01007         allocate(ecore(1,2))
01008     endif
01009
01010     !!== ip loop to spedify external q ==
01011     c      do 1001 ip = 1,nq
01012     c          if(sum(irkip(is,:,:),ip))==0) cycle
01013     c      call sxcf_fal3_scz(kount,q,itq,ntq,ef,esmr,
01014     i      nspin,is,
01015     i      qbas,qinv,qibz,qbz ,wbz, nstbz,
01016     i      irkip(is,:,:),nrkip(is,:,:),
01017     i      freq_r,nw_i,nw, freqx,wxw,
01018     i      dwdummy,
01019     i      ecore(:,is),
01020     d      nlmto,nqibz,nqibz,nctot,
01021     d      nbloch,ngpr,niw,nq,
01022     i      nblochpmx, ngpmx,ngcmx,
01023     i      wgt0,nq0i,q0i,symgg,alat,
01024     i      nband,             !shtvg,
01025     i      ifvcfpout,
01026     i      exchange, screen, cohtest, ifexsp(is),
01027     i      nbmx,ebmx,
01028     i      wkml,lxklm,
01029     i      eftrue,
01030     i      jobsw = isigmode, nbandmx=nbandmx(1:nq,is), !nbandmx is input mar2015
01031     i      hermitianw=hermitianw,
01032     o      zsec=zsec)
01033     c 1001 continue
01034     !TIME1_0020 "main:endofsxcf_fal3_scz"
01035     c      call date_and_time(values=timevalues)
01036     c      write(6,'(a,9i5)') 'dateandtime2=',MPI__rank,timevalues(1:8)
01037     c      call cpu_time(time_red1)
01038
01039     !! CAUTION! Allreduce wait all cpu jobs done here.
01040     !! Before nov2013, MPI__sxcf_rankdivider was stupid---> half of cores assigned for isp=2
01041     !! was just waiting here!
01042     c      call MPI__AllreduceMax( nbandmx(:,is), nq ) ! MIZUHO-IR
01043     c      call cpu_time(time_red2)
01044     c      write(6,*) MPI__rank,'time(MPI__AllreduceMax)=',time_red2-time_red1
01045
01046     c$$$$!! electron gas bare exchange (exact)
01047     c$$$$         if (legas.and.exchange) then
01048     c$$$$             efz=(ntot*3*pi**2/voltot)**(2d0/3d0) ! ef is calculated from ntot.
01049     c$$$$             pi
01050     c$$$$                 = 4.d0*datan(1.d0)
01051     c$$$$             tpia
01052     c$$$$                 = 2.d0*pi/alat
01053     c$$$$             qfermi= dsqrt(efz)
01054     c$$$$             alpha = (9*pi/4d0)**(1d0/3d0)
01055     c$$$$             write (6,*) ' --- exact electron gas bare exchange --- '
01056     c$$$$             write (6,*) ' density parameter rs= ', alpha/qfermi
01057     c$$$$             write (6,*) ' kf= ',qfermi
01058     c$$$$             do
01059     c$$$$                 ip = 1,nq
01060     c$$$$                 greal = tpia*q(1:3,ip)
01061     c$$$$                 qm = dsqrt ( sum(greal**2) )
01062     c$$$$                 xsex = hartree * egex (qm,efz)
01063     c$$$$                 write (6,*)
01064     c$$$$                 write (6, "(' True qm-ef Sx=',2f14.6,' q/qf=',f14.6)")
01065     c$$$$                 & rydberg()*(qm**2-efz), xsex, qm/qfermi
01066     c$$$$                 write (6, "(' Num qm-ef Sx=',2f14.6)")
01067     c$$$$                 & eqx(1,ip,is), hartree*dreal(zsec(1,1,ip)) !sf 21May02
01068     c$$$$                 write (6, "(' == diff =',2f14.6)")
01069     c$$$$                 & rydberg()*(qm**2-efz)-eqx(1,ip,is)
01070     c$$$$                 & xsex - hartree*dreal(zsec(1,1,ip)) !sf 21May02
01071     c$$$$                 write (661, "(' qm True qm-ef Sx=',3f14.6)")
01072     c$$$$                 & qm,rydberg()*(qm**2-efz), xsex
01073     c$$$$                 write (662, "(' qm Num qm-ef Sx=',3f14.6)")
01074     c$$$$                 & qm,eqx(1,ip,is), hartree*dreal(zsec(1,1,ip)) !sf 21May02
01075     c$$$$                 write (ifsex(is),6600) greal(1),greal(2),greal(3),xsex
01076     c$$$$                 write (6,6600) greal(1),greal(2),greal(3),xsex
01077     c$$$$                 6600 format (' greal =',3f8.4,' SEx(q) =',d13.5)
01078     c$$$$                 write (663,"(2f14.6)") qm/qfermi, qfermi
01079     c$$$$             end do
01080     c$$$$         endif
01081     c 2001 continue
01082
01083     !! eibz4sig symmetrization. MPI__AllreduceSum in zsecsym.
01084     if(eibz4sig()) then
01085         !TIME0_0030
01086         call zsecsym(zsec,ntq,nq,nband,nbandmx,nspinmx, eibzsym,ngpr,tiii,q,is)
01087         !TIME1_0030 'zsecsym'
01088     endif
01089     !TIME0_0040
01090     call mpi__allreducesum( zsec,ntq*ntq*nq )
01091     !TIME1_0040 'MPI__AllreduceSumzsec'
01092
01093     if(mpi__root) then
01094         if(exchange) then
01095             ifoutsex=ifsex(is)

```

```

01093         write(6,*)
01094         do ip = 1,nq
01095             do i = 1,ntq
01096                 write(ifoutsex,"(3i5,3d24.16,3x,d24.16,3x,d24.16)")
01097                 & itq(i),ip,is, q(1:3,ip), eqx(i,ip,is),
01098                 & hartree*dreal(zsec(i,i,ip)) !sf 21May02
01099                 if( eqx(i,ip,is)<1d20.and.abs(zsec(i,i,ip))/=0d0 ) then !takao june2009
01100                     write(6,"(' j iq isp=' i3,i4,i2,' q=' ,3f8.4,' eig=' ,f10.4,' Sx=' ,f10.4)")
01101                     & itq(i),ip,is, q(1:3,ip), eqx(i,ip,is),
01102                     & hartree*dreal(zsec(i,i,ip)) !sf 21May02
01103                 endif
01104             end do
01105             write(ifsex2(is)) is, q(1:3,ip), zsec(1:ntq,1:ntq,ip) !SEC_nn' out
01106         end do
01107         elseif(ixc==2) then
01108             ifoutsec=ifsec(is)
01109             do ip = 1,nq
01110                 do i = 1,ntq
01111                     if( eqx(i,ip,is)<1d20.and.abs(zsec(i,i,ip))/=0d0 ) then !takao june2009
01112                         write(6,"(' j iq isp=' i3,i4,i2,' q=' ,3f8.4,' eig=' ,f8.4,' Re(Sc) =' ,f8.4,' Img(Sc)
= ,f8.4 )")
01113                         & itq(i),ip,is, q(1:3,ip), eqx(i,ip,is),
01114                         & hartree*dreal(zsec(i,i,ip)),
01115                         & hartree*dimag(zsec(i,i,ip))
01116                     endif
01117                     write(ifoutsec,"(3i5,3d24.16,3x,d24.16,3x,d24.16, 3x,d24.16)")
01118                     & itq(i),ip,is, q(1:3,ip), eqx(i,ip,is),
01119                     & hartree*dreal(zsec(i,i,ip)),
01120                     & hartree*dimag(zsec(i,i,ip))
01121                 end do
01122                 write(ifsec2(is)) is, q(1:3,ip), zsec(1:ntq,1:ntq,ip) !SEC_nn' out
01123             end do
01124         endif
01125     endif
01126     !ixc
01127     !MPI__root
01128
01129 2000 continue
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178

```



```

01179 c$$$      filenameex = 'EXSS'//charnum3(ipex)//charnum3(itpex)
01180 c$$$      &      //'.'//char(48+is)
01181 c$$$      ifexspxx=4112
01182 c$$$      open(ifexspxx,file=filenameex)
01183 c$$$
01184 c$$$      do i=1,nnex
01185 c$$$          write(ifexspx, "(2d14.6, i4, 3f14.6)")
01186 c$$$      &          eex1 (i,itpex,ipex), exspl (i,itpex,ipex),
01187 c$$$      &          itex1 (i,itpex,ipex), qqex1 (1:3,i,itpex,ipex)
01188 c$$$      enddo
01189 c$$$c
01190 c$$$      eee =-1d99
01191 c$$$      exwgt= 0d0
01192 c$$$      do i=1,nnex
01193 c$$$          if(eex1(i,itpex,ipex) > eee+1d-4 .or. i==nnex) then
01194 c$$$              if(i/=1) write(ifexspxx, "(2d23.15)")
01195 c$$$      &              eee, exwgt*hartree
01196 c$$$              eee = eex1(i,itpex,ipex)
01197 c$$$              exwgt= exspl (i,itpex,ipex)
01198 c$$$          else
01199 c$$$              exwgt= exwgt + exspl (i,itpex,ipex)
01200 c$$$          endif
01201 c$$$      enddo
01202 c$$$c
01203 c$$$      deallocate( ieord )
01204 c$$$      close(ifexspx)
01205 c$$$      close(ifexspxx)
01206 c$$$      enddo
01207 c$$$      enddo
01208 c$$$      deallocate( eex1, exspl, nspex, itex1, qqex1 )
01209 c$$$      enddo
01210 c$$$      write(6,*)' End of ExSpectrum section ---'
01211 c$$$      endif
01212 c      isx = iclose ('wc.d')
01213 c      isx = iclose ('wci.d')
01214 c      isx = iclose ('hbe.d')
01215 call cputid(0)
01216 write(6,*) '--- end of hsfp0_sc --- irank=',mpi__rank
01217 call flush(6)
01218 call mpi__finalize
01219 !TIME1_0000 "main:totalofhsfp0_sc"
01220 !TIMESHOW
01221 if(ixc==1 ) call rx0( ' OK! hsfp0_sc: Exchange mode')
01222 if(ixc==2 ) call rx0( ' OK! hsfp0_sc: Correlation mode')
01223 if(ixc==3 ) call rx0( ' OK! hsfp0_sc: Core-exchange mode')
01224 end program hsfp0_sc
01225
01226
01227
01228 subroutine zsecsym(zsec,ntq,nq,nband,nbandmx,nspinx, eibzsym,ngrp,tiii,q,is)
01229 !! --- symmetrize zsec for eibz4sig mode. -----
01230 !! Read a file lmfgw_kdivider, which contains info for vxc and evec (they are in separated files in MPI)
01231 !!
01232 c      use m_mpi,only: MPI__AllreduceSum
01233 c      use m_readeigen,only: readeval
01234 implicit none
01235 complex(8),intent(inout)::zsec(ntq,ntq,nq)
01236 integer,intent(in)::ntq,nq,nspinx,nband,nbandmx(nq,nspinx),is
01237 integer,intent(in):: ngrp,eibzsym(ngrp,-1:1,nq)
01238 logical,intent(in):: tiii !time reversal switch
01239 real(8),intent(in):: q(3,nq)
01240
01241 complex(8),allocatable::zsect(:,:)
01242 integer:: ifile_handle,iqq
01243 integer:: procid,nrankv,ifvxc_,ifevec_,ifiproc,iqqxx,
01244 & ispx,ixx,ixxx,nqixx,nspxx,ispxx,iqbz,i,igrp,iq
01245 character*256:: extn,ext
01246 character*256,allocatable:: extp(:)
01247 integer,allocatable:: ifevec_(:),ifvxc_(:),iproqc(:,:)
01248
01249 integer:: nsym,nhdim,it,nblk,iband,napw,ldim,ierr,ispix,nbsize,nbsizemx
01250 & ,iblk1,iblk2,ii1,ii2,ie1,ie2,ne1,ne2,iqxx, ndimhx, nspix,nnnx
01251 integer,allocatable:: iblki(:),iblke(:)
01252 complex(8),allocatable:: evec(:,:),evec_inv(:,:),evecrot(:,:),rmatjj(:,:,:)
01253 real(8),allocatable::evaliq(:)
01254 real(8)::tolry=1d-4,qqqx(3),qtarget(3),tolq=1d-8
01255 complex(8),allocatable:: ovl(:,:)
01256 integer::nev,j
01257 !TIME0_0100
01258 write(6,*)'zsecsym:'
01259 allocate( zsect(ntq,ntq) )
01260 !! === readin lmfgw_kdivider, and get extensions === apr2013
01261 ifiproc=ifile_handle()
01262 open(unit=ifiproc,file='lmfgw_kdivider',status='old')
01263 read(ifiproc,*) ext
01264 read(ifiproc,*) nqixx, nspxx, nrankv
01265 if(allocated(iprocq)) deallocate(iprocq)

```

```

01266     allocate(iprocq(nqixx,nspxx))
01267     do isp=1,nspxx
01268         do iqq=1,nqixx
01269             read(ifiproc,*) iqqxx, ispxx, ixxx
01270             if(iqqxx/=iqq) call rx( 'iqqxx/=iqq')
01271             if(ispxx/=isp) call rx( 'ispxx/=isp')
01272             iprocq(iqq,isp) = ixxx
01273             write(6, "('iqq isp irank=',i8,i2,i6)") iqq,isp, iprocq(iqq,isp)
01274         enddo
01275     enddo
01276     close(ifiproc)
01277 !! for multiple files.
01278 c     if(allocated(extp)) deallocate(extp,ifvxc_,ifevec_)
01279     allocate(extp(0:nrankv-1),ifvxc_(0:nrankv-1),ifevec_(0:nrankv-1))
01280     extp(0) = trim(ext)
01281     write(6, "(' 0 ext= ',a,a)") trim(extp(0)), ' -----'
01282     do procid=1,nrankv-1
01283         write(extn,"(i10)") procid
01284         extp(procid)=trim(adjustl(ext))//'_ '//trim(adjustl(extn))
01285         write(6, "(i3, ' ext= ',a,a)") procid,trim(extp(procid)), ' -----'
01286     enddo
01287     do procid=0,nrankv-1
01288         ifvxc_(procid) = ifile_handle()
01289         open( ifvxc_(procid), file='vxc'//extp(procid),form='unformatted')
01290         ifevec_(procid)= ifile_handle()
01291         open( ifevec_(procid), file='evc'//extp(procid),form='unformatted')
01292     enddo
01293     ifvxc_ = ifvxc_(0) !0 is root
01294     ifevec_ = ifevec_(0)
01295     read(ifevec_) ndimhx, nsp,nnnx
01296     read(ifvxc_) !skip ndimh, nsp,nnn
01297     allocate(evaliq(nband),iblki(nband),iblke(nband))
01298 !TIME1_0100 "zsecsym:endof_allocate_zsect"
01299 !TIME0_0110
01300     iqq=0 !iqq is to read multiple vxc.* evec.*
01301     do 3020 iq=1,nq !nq means iq for which we will calculate sigma
01302         iqq=iqq+1
01303         do 3030 ispx=1,nspinmx !ispx loop is to find isx=is
01304             ifvxc_ = ifvxc_(iprocq(iqq,ispx))
01305             ifevec_ = ifevec_(iprocq(iqq,ispx))
01306             if(ispx==is) then
01307 !this if-block is due to evec and v_xc file-->they shall be divided into spin files.
01308                 read(ifvxc_) nhdim,ldim
01309                 read(ifvxc_)
01310                 allocate( evec(nhdim,nhdim),evecrot(nhdim,nhdim))
01311                 read(ifevec_) qqxx(1:3), evec(1:nhdim,1:nhdim),nev !nev number of true bands nov2015
01312                 zsect = 0d0
01313             else !skip isx/=is. Need to get access sequential files evec and v_xc.
01314                 read(ifvxc_)
01315                 read(ifvxc_)
01316                 read(ifevec_)
01317                 cycle
01318             endif
01319             do i=1,nnnx !nq !qqqx from evec v_xc.
01320                 if(sum(abs(qqqx-q(:,i)))<tolq) then
01321                     iqxq=i
01322                     goto 3011
01323                 endif
01324             enddo
01325             deallocate(evec,evecrot)
01326             call rx( 'hsfp0_sc: bug:qqqx can not find ...')
01327 3011 continue
01328             if(tiii) call rx( 'timereversal is not yet implemented')
01329         enddo
01330 !! evec_inv(ibl,iww)= \sum_ib2 ovlinv(ibl,ib2)*dconjg(evec(iww,ib2)) nov2015, we introduce nev. iww is
for PMT basis. ib for band index.
01331 !! This is for converting rotated evec (=evecrot(ib)) in the representation of original evec(ib).
01332     allocate(ovl(nev,nev))
01333 c     print *, 'nnnnnnnnnn zsecsym: nband=',nhdim,nband,nev
01334     do i=1,nev
01335         do j=1,nev
01336 c         write(6,*) 'evec orth=',i,j,sum(dconjg(evec(:,i)*evec(:,j)))
01337             ovl(i,j)=sum(dconjg(evec(:,i))*evec(:,j))
01338         enddo
01339     enddo
01340     call matcinv(nev,ovl) !ovl --> ovlinv
01341     allocate(evec_inv(nev,nhdim))
01342     evec_inv = matmul(ovl(1:nev,1:nev),dconjg(transpose(evec(:,1:nev)))) !note ovl means ovlinv
01343     deallocate(ovl)
01344 c     evec_inv = evec
01345 c     call matcinv(nhdim,evec_inv)
01346     call readeval(q(:,iqxx), is, evaliq)
01347     nsym = sum(eibzsym(:, :,iqxx))
01348     do it=1,1 !no-time reversal yet !it=1,-1,-2 !c.f. x0kf_v4h
01349         do igrp=1,ngrp !A-rotator
01350             if( eibzsym(igrp,it,iqxx)==0) cycle
01351             nblk=0

```

```

01352         iblki=0
01353         iblke=0
01354         iblki(1)=1
01355 !! degeneracy divider for evaliq. See How to apply EIBZ to
01356 !! Is this procedure really make speed up so much?
01357         tolry= 0.2d0 !Degeneracy tol. if tolry is large,
01358 !! larger tolry is safer, although a little inefficient.
01359 !! If tolry is too small to divide degenerated values to different blocks --> then we have wrong results.
01360 !(NOTE that Hamiltonian can be not so symmetric in some reasons)
01361         nbsizemx=0
01362         do iband=2,nbandmx(iqxx,is)
01363 ! nbandmx is the number of bands for which we calculate self-energy.
01364 ! We assume nbandmx(iqxx,is) is well separated for degeneracy.
01365         if(evaliq(iband) > evaliq(iband-1)+tolry
01366         & .or. iband==nbandmx(iqxx,is)) then
01367             nblk=nblk+1
01368             if(nblk>=2) iblki(nblk)=iblke(nblk-1)+1
01369             if(iband==nbandmx(iqxx,is)) then
01370                 iblke(nblk)=iband
01371             else
01372                 iblke(nblk)=iband-1
01373             endif
01374             nbsize = iblke(nblk)- iblki(nblk)+1
01375             if( nbsize>nbsizemx ) nbsizemx = nbsize
01376         endif
01377         enddo ! iband
01378 !! rotation of evec. Generate evecrot. (Within degenerated block, evec are mapped).e
01379         allocate(rmatjj(nbsizemx,nbsizemx,nblk))
01380         napw=nhdim-ldim
01381         do iblk1=1,nblk
01382             iil=iblki(iblk1)
01383             iel=iblke(iblk1)
01384             nel=iel-iil+1
01385             call rotwvigg(igrp,q(:,iqxx),q(:,iqxx),nhdim,
01386             & napw,nel,evec(:,iil:iel),evecrot(:,iil:iel),ierr )
01387             rmatjj(1:nel,1:nel,iblk1) =
01388             & matmul(evec_inv(iil:iel,:),evecrot(:,iil:iel))
01389         enddo ! iblk1
01390         do iblk1=1,nblk
01391             do iblk2=1,nblk
01392                 iil=iblki(iblk1)
01393                 iel=iblke(iblk1)
01394                 nel=iel-iil+1
01395                 ii2=iblki(iblk2)
01396                 ie2=iblke(iblk2)
01397                 ne2=ie2-ii2+1
01398                 zsect(iil:iel,ii2:ie2)= zsect(iil:iel,ii2:ie2)
01399             & + matmul( dconjg(transpose(rmatjj(1:nel,1:nel,iblk1))),
01400             & matmul(zsect(iil:iel,ii2:ie2,iqxx),
01401             & rmatjj(1:ne2,1:ne2,iblk2)) )
01402             enddo ! iblk2
01403         enddo ! iblk1
01404         deallocate(rmatjj)
01405         enddo ! igrp
01406         enddo ! it
01407         deallocate(evec, evec_inv, evecrot)
01408         zsec(:, :, iqxx) = zsect(:, :)/dble(nsym)
01409 c         call MPI_AllreduceSum( zsec(:, :, iqxx), ntq*ntq ) ! MIZUHO-IR
01410 3030 continue ! ispx
01411 3020 continue ! iq
01412         do procid=0,nrankv-1
01413             close(ifvxc__(procid) )
01414             close(ifevec__(procid))
01415         enddo
01416         deallocate(iblki, iblke, evaliq)
01417         deallocate(zsect, extp, ifevec__, ifvxc__, iprocq)
01418 !TIME1_0110 "sub_zsecsym"
01419         end subroutine zsecsym

```

## 4.31 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 [zgesvdsn2](#) (no, nn, nmx, epsmx, pmat, nnn)
- subroutine [mkb0](#) (q, lxx, lx, nxx, nx, aa, bb, nrr, nrx, rprodx, alat, bas, nbas, nbloch, b0mat)

#### 4.31.1 Function/Subroutine Documentation

4.31.1.1 subroutine [checkagree](#) ( [real\(8\)](#), dimension(3) *a*, [real\(8\)](#), dimension(3) *b*, [character](#)\*(\*) *char* )

Definition at line [1286](#) of file [hvccfp0.m.F](#).

4.31.1.2 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 [1412](#) of file [hvccfp0.m.F](#).

4.31.1.3 program [hvccfp0](#) ( )

Definition at line [1](#) of file [hvccfp0.m.F](#).

Here is the call graph for this function:

4.31.1.4 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 [1467](#) of file [hvccfp0.m.F](#).

Here is the caller graph for this function:

4.31.1.5 subroutine [mkradmatch](#) ( [real\(8\)](#), dimension(1:2, 1:nxdim) *p*, [integer\(4\)](#) *nxdim*, [real\(8\)](#), dimension(1:nxdim,1:nxdim) *rdmatch* )

Definition at line [1296](#) of file [hvccfp0.m.F](#).

Here is the call graph for this function:

Here is the caller graph for this function:

4.31.1.6 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 [1365](#) of file [hvccfp0.m.F](#).

Here is the caller graph for this function:

4.31.1.7 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 1388 of file hvccfp0.m.F.

Here is the caller graph for this function:

4.31.1.8 subroutine zgesvddn2 ( integer(4) no, integer(4) nn, integer(4) nnmx, real(8) epsmx, complex(8), dimension(no,nn) pmat, integer(4) nnn )

Definition at line 1429 of file hvccfp0.m.F.

## 4.32 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 nbq0.
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 m_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,lncjg,lncxg,
00024      &   nkq,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,trip1,alat0,epsx,
00027      &   tol,as,tpiba,qb0(3,3),vol0,rdist0,qdist0,radd,qadd,
00028      &   a0,awald,alat1,toll,r0,q0,awald0,qg(3),   absqg2,aaa,aaal2
00029      integer(4),allocatable :: jcg(:),indxcg(:),
00030      &   lx(:),kmx(:),nblocha(:),nr(:),ificrb(:),
00031      &   nx(:,:),ngvecp(:,:),ngvecc(:,:),ngveci(:,:),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(:,:),sgpp(:,:),
00037      &   sgpb(:,:),fouvvp(:,:),vcoul0(:,:),
00038      &   foubv(:,:),fouvvp(:,:),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,igl,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,ixq,ipl1,ipl2,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) :: ixmini, ixqend,imode
00062      logical :: allochk=.false. !paralellx0=.true.,
00063
00064      complex(8),allocatable:: hhl(:,:),ool(:,:)
00065      integer(4):: nqnumc,ifigc !bzcase,
00066 c      character(5) :: charnum5
00067 c      integer(4),allocatable:: iqok(:)
00068      real(8):: qq(3),qpgcut_cou      !,qq(3)
00069
00070      integer(4),allocatable:: ngvecc0(:,:)
00071      integer(4):: ngc0
00072
00073      real(8):: ginv(3,3),quu(3)
00074
00075 c---
00076      real(8),allocatable :: rkpr(:,:,:),rkmr(:,:,:),rofi(:,:)
00077      real(8):: eee,ees, 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,nq0iadd
00096      logical:: is_mix0vec,wvcc !,newaniso
00097      character(128):: vcoudfile
00098      real(8),allocatable:: wqfac(:),qbwzww(:,:)
00099      integer:: ifiwqfac,igbz,igbzx,nnn,ixyz
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,ixqini,ixqend)
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          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 c      elseif(imode==0) then
00126 c      elseif(imode==3) then
00127 c      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(ixqini< 2) paralellx0=.false.
00132 c      if(paralellx0) then
00133 c          write(6, "(' PARALELL.X0 mode: ixqini ixqend=',2i3) ")
00134 c      & ixqini, ixqend
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, ngbz, nbas, nband
00140      if(allochk)
00141      & write(*,*) 'allocate(qbz(3,ngbz),bas(3,nbas),rmax(nbas))'
00142      allocate(qbz(3,ngbz),bas(3,nbas),rmax(nbas))
00143      read(ifhvccfp) qbz, bas,rmax
00144      read(ifhvccfp) ngibz
00145      if(allochk)
00146      & write(*,*)'allocate(qibz(3,ngibz),iqibzx(ngibz))'
00147      allocate(qibz(3,ngibz),iqibzx(ngibz))
00148      read(ifhvccfp) qibz(1:3,1:ngibz)
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,ngibz)
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,ngibz)
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 ngbz=',nnn,ngbz
00171 c$$$c      if(nnn/=ngbz) stop 'hvccfp0_sc: readin nnn WQFAC/= ngbz'
00172 c$$$      allocate( wqfac(nnn),qbwzww(3,nnn) )
00173 c$$$      read(ifiwqfac) wqfac,qbwzww
00174 c$$$      ifiwqfac = iclose('WQFAC')
00175
00176 c$$$c-----
00177 c$$$      iqibzx=0
00178 c$$$      do iqibz=1,ngibz
00179 c$$$          do iq =1,ngbz
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$$$      & write(*,*) 'allocate( ngcn(ngbz), ngvecci(3,ngcmx,ngibz))'
00193 c$$$      allocate( ngcn(ngbz), ngvecci(3,ngcmx,ngibz))
00194 c$$$      do iq=1, ngbz
00195 c$$$          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$$$      allocate( ngcn(ngbz), ngvecci(3,ngcmx,ngibz), ngvecc(3,ngcmx),iqok(ngibz))
00214 c$$$      iqok=1
00215 c$$$      do iq=1, nqnumc
00216 c$$$          read (ifiqgc) qqg, ngc
00217 c$$$          read (ifiqgc) ngvecc(1:3,1:ngc)
00218 c$$$          do iqibz=1,ngibz
00219 c$$$              if( sum(abs(qibz(:,iqibz)-qqg))<1d-8 ) then
00220 c$$$                  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),ngveccc(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, ngvecc(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,filenamee,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      ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00289      cccc TEST ccccccccccccccccccccccccccccccccccccccccccccccccc
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      ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00295
00296
00297
00298
00299
00300      ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00301 c TEST ccccccccccccccccccccccccccccccccccccccccccccccccccccc
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. ccccccccccccc
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 also 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,igx)+ matmul(qlat, ngvecci(1:3,ig1,igx)))
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(1)* 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),1,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), aaal2 )
00375              rprodx(1:nr(ibas),n2,l,ibas) = rprodx(1:nr(ibas),n2,l,ibas)
00376      & - aaal2*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 ccccccccccccccccccccccccccccccccccc
00386      ccccccccccccccccccccccccccccccccccccccccccccccccccc
00387
00388
00389      nbloch = sum(nblocha)
00390      nblochpmx = nbloch + ngcmx
00391
00392  c --- CG coefficientnets. <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 c          allocate(cg(lnjcg),jcg(lnjcg),indxcg(lnxcg))
00416 c          call scg(lmxcg,cg,indxcg,jcg)
00417 c          if(allochk) write(6,*)' end of scg: cg coefficients generated.'
00418
00419
00420      call minv33(qlat,ginv)
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*lx    !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 cccccccccccccccccccccccccccccccccccccccccc
00453 c takao
00454 c      toll= tol*alat**(lmax+1) *0.01
00455 cccccccccccccccccccccccccccccccccccccccccc
00456      toll= toll*alat**(lmax+1)
00457 c      if(allochk) write(*,*) 'allocate(dlv, qlv, work) '
00458 c      allocate(dlv(3,nkdmx), qlv(3,nkqmx), work(max0(nkdmx,nkqmx)) )
00459 c      call lctoff(a0,vol0,lmax,toll,r0,q0)
00460 c      nkdest =4.18879*(r0+radd)**3/vol0+.5
00461 c      nkrest =4.18879*(q0+qadd)**3*vol0+.5
00462 c      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 c          .      ' awald=',0p,f7.4,' v0=',f10.3/' alat1=',f9.5,
00465 c          .      ' estimates: nkdest',i6,' nkrest',i6)
00466 c      call lgen(plat,r0+radd,nkd,nkdmx,dlv,work)
00467 c      write(6,342) r0,r0*alat,radd,nkd
00468 342 format(' r0=',f9.4,' rc=',f9.4,' radd=',f9.4,' nkd=', i7)
00469 c      call lgen(qb0,q0+qadd,nkq,nkqmx,qlv,work)
00470 c      write(6,341) q0,q0*tpiba,qadd,nkq
00471 341 format(' q0=',f9.4,' qc=',f9.4,' qadd=',f9.4,' nkq=', i7)
00472 c      if(allochk) write(*,*) 'deallocate(work)'
00473 c      deallocate(work)
00474
00475 c... readin r0c
00476 c      if(newaniso()) then
00477 c          eee=screenfac() !takao feb2012
00478 c      elseif(imode==101.or.imode==102) then
00479 c          eee = ees()
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--- bessell and hankel for the expansion of exp(-r/r_0)/r.
00494 c bessell and hankel is renormarized so that its behaves as r^1 and r^{-1-1} near r=0.
00495 c rkpr means r^1*r for e=0 (r0c =infinity) case
00496 allocate(rkpr(nrx,0:lx, nbas), rkmr(nrx,0:lx, nbas), rofi(nrx, nbas))
00497 do ibas=1, nbas
00498 call genjrh(eee, nr(ibas), aa(ibas), bb(ibas), lx(ibas), nrx, lx,
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:lx, nbas), sgbb(nxx, nxx, 0:lx, nbas))
00505 c do ibas = 1, nbas
00506 c call mkjb( lx, lx(ibas), nxx, nx(0:lx, 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:lx, nbas), sgbb(nxx, nxx, 0:lx, nbas))
00512 do ibas = 1, nbas
00513 call mkjb_4( lx, lx(ibas), nxx, nx(0:lx, 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= (lx+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,*) nq0i, idummy, nq0iadd
00539 if(allochk) write(*,*) 'allocate( wgt(1:nq0i), q0i(1:3, 1:nq0i) )'
00540 allocate( wgt(1:nq0i+nq0iadd), q0i(1:3, 1:nq0i+nq0iadd) )
00541 read (101,*) ( wgt(i), q0i(1:3, i), ixyz, i=1, nq0i+nq0iadd)
00542 write (6, "(d13.5, 3x, 3d13.5)" ) ( wgt(i), q0i(1:3, i), i=1, nq0i+nq0iadd)
00543 close(101)
00544 write(6,*) ' *** goto do iq nqibz+nq0iadd nq0i=', nqibz, nq0i+nq0iadd
00545
00546 C --- Check PARALELL.X0
00547 c INQUIRE (FILE = 'PARALELL.X0', EXIST = paralellx0)
00548 c$$$ wvcc=.true.
00549 c$$$ if(newaniso()) wvcc=.false.
00550 wvcc=.false.
00551 write(6, '(a)') " Mix0vec.XXX is not empty only when"
00552 & /*" the corresponding q is in Q0P with zero weight."
00553 c if(paralellx0) then
00554 c if(wvcc) ifvcfpout = iopen( "VCCFP." //xxt(ixxini, ixxend), 0, -1, 0)
00555 c ifgb0vec = iopen ( "Mix0vec." //xxt(ixxini, ixxend), 1, 3, 0)
00556 c ifgb0vec1 = iopen ( "Mix0vec1." //xxt(ixxini, ixxend), 1, 3, 0)
00557 c else
00558 c ixxend = nqibz + nq0i+nq0iadd
00559 c if(wvcc) ifvcfpout = iopen('VCCFP', 0, -1, 0)
00560 c ifgb0vec = iopen( "Mix0vec", 1, 3, 0)
00561 c ifgb0vec1 = iopen( "Mix0vec1", 1, 3, 0)
00562 c endif
00563
00564 if(imode==202) then
00565 ixxini= nqibz + 1
00566 c elseif(paralellx0) then
00567 c & !skip
00568 c$$$ elseif(bzcase()==1) then
00569 c$$$ ixxini = 2
00570 c$$$ ixxini = 1 !oct2005
00571 else

```

```

00572         iqxini = 1
00573     endif
00574 !!
00575 c         if(newaniso().and.imode==0) then
00576         if(imode==0) then
00577             iqxini=1
00578 c         iqxend=nqibz ! comment out at 18nov2012
00579         endif
00580         write(6,*)'iqxini iqxend=',iqxini,iqxend
00581 c qibz loop
00582 c         epsx = 0.01d0
00583 c         if(bzcase()==1) then
00584         if(abs(sum(qibz(:,1)**2))/=0d0) call rx( 'hvccfp0: sum(q**2)==0d0')
00585 c         endif
00586         if(wvcc) write(ifvcfpout) nqibz, nblochpmx
00587 c         if(imode==101) then
00588             read(ifvcfporg) nqibz_in, nblochpmx_in
00589 c         if(nqibz /= nqibz_in) stop 'nqibz /= nqibz_in VCCFP.ORG'
00590 c         if(nblochpmx /= nblochpmx_in)
00591 c         & stop 'nblochpmx /= nblochpmx_in VCCFP.ORG'
00592 c         endif
00593
00594 C... Readin PRODMT into prodmt. oct2005
00595         smbb = smbasis()
00596         write(6,*) ' smooth mixed basis=',smbb
00597         if(smbasis()) then
00598             allocate( prodmt(2,nxx,0:1xx,nbas))
00599             allocate( nx_r(0:1xx))
00600             do ibas =1,nbas
00601                 filenamep = 'PRODMT_'//charnum3(ibas)
00602                 ifprodmt = iopen(filenamep,0,-1,0)
00603                 read(ifprodmt) nl_r
00604                 if( 2*(nl_r-1) /= 1xx ) then
00605                     write(6,*) 2*(nl_r-1),1xx
00606                     call rx( '2*nl_r-1 /= 1xx ')
00607                 endif
00608                 read(ifprodmt) nxx_r
00609                 write(6, "(' nxx =' ,100i3)")nxx_r
00610                 if(nxx_r>nxx) call rx( 'nxx_r>nxx')
00611                 read(ifprodmt) nx_r(0:1xx)
00612                 write(6, "(' nx_r=' ,100i3)") nx_r(0:1xx)
00613                 lx_ = lx(ibas)
00614                 if(sum(abs(nx(0:lx_,ibas)-nx_r(0:lx_))) /=0) then
00615                     write(6,*)' debug: nx  =' ,nx(0:lx_,ibas)
00616                     write(6,*)' debug: nx_r=' ,nx_r(0:lx_)
00617                     call rx( 'nx /=nx_r')
00618                 endif
00619                 read(ifprodmt) prodmt(1:2, 1:nxx_r, 0:1xx, ibas)
00620                 write(6,*)' sumcheck prodmt=',sum(abs(prodmt(:, :, ibas)))
00621                 isx = iclose(filenamep)
00622             enddo
00623
00624 C... Check write for radial part of the product basis
00625         if(.false.) then
00626             do ibas= 1,1 !1,nbas
00627                 do l = 0,1x(ibas)
00628                     open(1011,file='ProdOld_ibas'//charnum3(ibas)//'_1'//charnum3(1))
00629 c                 open(2011,file='ProdNew_ibas'//charnum3(ibas)//'_1'//charnum3(1))
00630                     nxdim = nx(1,ibas)
00631                     do ix=1,nxdim
00632                         write(1011,"(' -- -- -- ',3i3,' --- ' )") ix,1,ibas
00633 c                     write(2011,"(' -- -- -- ',3i3,' --- ' )") ix,1,ibas
00634                         do ir =1,nr(ibas)
00635                             write(1011,"(d13.5,2x,2d18.8)")
00636                             & rofi(ir,ibas), rprodx(ir,ix,1,ibas)
00637                             & , rprodx(ir,ix,1,ibas) /rofi(ir,ibas)
00638 c                             write(2011,"(d13.5,2x,2d18.8)")
00639 c                             & rofi(ir,ibas), sum(rprodx(ir,1:nxdim,1,ibas)*rdmatch(1:nxdim,ix,1,ibas))
00640 c                             & , sum(rprodx(ir,1:nxdim,1,ibas)*rdmatch(1:nxdim,ix,1,ibas))/rofi(ir,ibas)
00641                         enddo
00642                     enddo
00643                     close(1011)
00644 c                     close(2011)
00645                     enddo
00646                     enddo
00647 c                 stop 'text end'
00648             endif
00649 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00650 !
00651             allocate( rdmatch(nxx,nxx,0:1xx,nbas) )
00652             do ibas= 1, nbas
00653                 do l = 0, 1x(ibas)
00654                     nxdim = nx(1,ibas)
00655                     if(nxdim<=1)write(6,*)'hvccfp0:smbasis case error nxdim <=1'
00656 !                     pval = prodmt(1, 1:nxdim, 1,ibas)
00657 !                     pslo = prodmt(2, 1:nxdim, 1,ibas)
00658 !                     prod(r, inew) = \sum_iold rrmrmat(inew,iold) * prod(r,iold)

```

```

00659         write(6, "('goto mkradmatch ibas lnxdim =',3i4)")ibas,l,lnxdim
00660         call mkradmatch(prodmt(1:2, 1:lnxdim, 1,ibas), lnxdim,
00661             o          rdmatch(1:lnxdim,1:lnxdim,1,ibas) )
00662         enddo
00663     enddo
00664
00665
00666
00667 ! index (mx,nx,lx,ibas) ordering: taken from voul_4
00668 allocate(ibl(-lxx:lxx,nxx,0:lxx,nbas))
00669 ibl1 = 0
00670 ibl=999999
00671 do ibas= 1, nbas
00672     do l = 0, lx(ibas)
00673         do n = 1, nx(1,ibas)
00674             do m = -1, 1
00675                 ibl1 = ibl1 + 1
00676                 ibl(m,n,1,ibas) = ibl1
00677 !         write(6,*)ibl1,n,1,m,lmb1(ibl1)
00678             enddo
00679         enddo
00680     enddo
00681 enddo
00682 if(ibl1/= nbloch) then
00683     write(6,*)' ibl1 nbloch',ibl1, nbloch
00684 Cstop2rx 2013.08.09 kino stop ' hvccfp0:smbasis mode error ibl1/= nbloch'
00685 call rx( ' hvccfp0:smbasis mode error ibl1/= nbloch')
00686 endif
00687 ! index (mx,nx,lx,ibas) ordering
00688 cttttt
00689 nnr = 2 ! =2 new
00690 ! =0 equivalence with original mixed basis
00691 write(6,*)' sss:nbas lx=',nbas,lx(1:nbas)
00692
00693 nbln=0
00694 do ibas= 1, nbas
00695     do l = 0, lx(ibas)
00696         write(6, "('sss: nx=',3i4)") ibas,l,nx(1,ibas)
00697         if(nx(1,ibas)<=0) cycle
00698 Cstop2rx 2013.08.09 kino if(nx(1,ibas)==1) stop 'nx(1,ibas) =1'
00699         if(nx(1,ibas)==1) call rx( 'nx(1,ibas) =1')
00700 ccccccccccccccccd
00701 cttttt
00702 c          nnr = 2 ! =2 new
00703 c          ! =0 equivalence with original mixed basis
00704 c          if(l<=3) nnr=0
00705 ccccccccccccccccd
00706         nbln = nbln + (2+l+1)*(nx(1,ibas)-nnr)
00707     enddo
00708 enddo
00709 allocate( pmat(nbloch+ngcmx, nbln+ngcmx) )
00710 pmat=0d0
00711 ibln = 0
00712 do ibas= 1, nbas
00713     do l = 0, lx(ibas)
00714 ccccccccccccccccd
00715 cttttt
00716 c          nnr = 2 ! =2 new
00717 c          ! =0 equivalence with original mixed basis
00718 c          if(l<=3) nnr=0
00719 ccccccccccccccccd
00720         do nn = nnr+1, nx(1,ibas) !nn=1 and nn=2 corresponds to non-zero val sol
00721             do m = -1, 1
00722                 ibln = ibln +1
00723                 nxdim = nx(1,ibas)
00724                 pmat( ibl(m,1:nxdim,1,ibas), ibln)
00725                 & = rdmatch(1:nxdim, nn, 1,ibas)
00726 cttttt
00727 c          pmat( ibl(m,nn,1,ibas), ibln)
00728 c          & = 1d0
00729 ccccccccccccccccdcccccccccccccccccccccccccccccccccccccd
00730         enddo
00731     enddo
00732 enddo
00733 enddo
00734 C... Store matting matrix (imatchn,imatcho,pmat)
00735 ifpomat = iopen('Pomat',0,-1,0)
00736 c          write(6,*)'ttt= sumchk pmat(b)=' ,sum(abs(pmat(1:nbloch, 1:nbln)))
00737     endif
00738 !! === open file Vcloud ===
00739 !! This contains E(\nu,I), given in PRB81,125102
00740
00741 !! == main loop for iqx ==
00742 call mpi_getrange( mpi_iini, mpi_iend, iqxini, iqxend )
00743 do 1001 iqx = mpi_iini, mpi_iend ! q=(0,0,0) is omitted!
00744 c$$$ do 1001 iqx = iqxini, iqxend ! q in IBZ. avoid q=0 case for iqx=1
00745     write(6, "('#### do 1001 start iqx=',i5)")iqx

```

```

00746      vcoudfile='Vcoud.'//charnum5(igx) !this is closed at the end of do 1001
00747      ifvcoud = iopen(trim(vcoudfile),0,-1,0)
00748      if(igx > nqibz) then !
00749          q = q0i(:,igx-nqibz)
00750 c      qq = 0d0
00751      else !
00752          q = qibz(:,igx)
00753 c      qq = q
00754      endif
00755      cccccccccccccccccccccccc
00756 c      if(imode==202) then !for igx>nqibz
00757 c          qq=q
00758 c      endif
00759      cccccccccccccccccccccccc
00760
00761 c$$$      if(.not. newaniso() ) then !this is for fe_epsPP_lmfm_chipm feb2012
00762 c$$$      if(sum(q**2)<ld-12) q=(/ld-4,0d0,0d0/) !takao oct2006
00763 c$$$      endif
00764 !! ==== q+G vector ====
00765      call readqg('QGcou',q,ginv, quu,ngc, ngvecc ) !qq-->q
00766      ngb = nbloch + ngc !it was ngcnn(ig)
00767      write(6,(' igx q ngc = ',i5,3f10.4,i5')) igx,q,ngc
00768
00769 c$$$      if(newaniso()) then
00770 c$$$      continue
00771 c$$$      elseif(bzcase()==1.and.igx==1) then
00772 c$$$      goto 1101
00773 c$$$      endif
00774
00775 c      ngc = ngcn(ig)
00776 c      ngvecc(1:3,1:ngc) = ngvecci(1:3,1:ngc,ig)
00777 c      write(6,*)' ig ngc=' ,ig, ngc
00778      cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00779 c      q test
00780 c      q=(/ 0.09d0,0.09d0,0.09d0/)
00781 c      q = q+(/ 0.01d0,0.01d0,0.01d0/)
00782 c      q=q/4
00783      cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00784
00785 C--- strxq structure factor.
00786      if(allochk) write(6,*)' goto strxq'
00787      if(allochk) write(*,*) 'allocate( strx(nlxx,nbas,nlxx,nbas))'
00788      allocate( strx(nlxx,nbas,nlxx,nbas))
00789      do ibas1 =1,nbas
00790          do ibas2 =1,nbas
00791              p = bas(:,ibas2)-bas(:,ibas1)
00792              phasep =exp(img*2*pi*sum(q*p))
00793              nlx1 = (lx(ibas1)+1)**2
00794              nlx2 = (lx(ibas2)+1)**2
00795              if(allochk) write(*,*) 'allocate( s(nlx1,nlx2))'
00796              allocate( s(nlx1,nlx2),sd(nlx1,nlx2)) !kino add sd----but sd is dummy
00797 c          call strxq(1,0d0,q,p,nlx1,nlx2,nlx1,alat,voltot,
00798                  call strxq(1,eee,q,p,nlx1,nlx2,nlx1,alat,voltot,
00799                      i      awald,nkd,nkq,dlv,qlv,
00800                      i      cg,indxcg,jcg,
00801                      o      s,sd)
00802              strx(1:nlx1,ibas1,1:nlx2,ibas2) = fpi*s      !!! *phasep
00803              if(allochk) write(*,*)'deallocate( s )'
00804              deallocate( s,sd )
00805          enddo
00806      enddo
00807      cccccccccccccccccccccccccccc
00808 c      strx=0d0
00809      cccccccccccccccccccccccccccc
00810
00811 C--- onsite integrals <j(e=0)|P^(q+G)_L> and <B|v(onsite)|B>
00812 c$$$      if(.true.) then !==New version without sgpp and fouvp allocation June2004=====
00813          if(allochk) write(*,*)'allocate(rojp,sgpb,fouv)'
00814          allocate( rojp(ngc, nlxx, nbas),
00815                  &      sgpb(ngc, nxx, nlxx, nbas),
00816                  &      fouv(ngc, nxx, nlxx, nbas))
00817 c          &      sgpp(ngc, ngc, nlxx, nbas),
00818 c          &      fouvp(ngc, ngc, nlxx, nbas) )
00819          do ibas = 1,nbas
00820              if(allochk) write(6,*)' --- goto mkjp_4',ibas
00821              call mkjp_4(q,ngc, ngvecc, alat, qlat,
00822                  i      lxx, lx(ibas),nxx, nx(0:lxx,ibas),
00823                  i      bas(1,ibas),aa(ibas),bb(ibas),rmax(ibas),
00824                  i      nr(ibas), nrx, rprodx(1,1,0,ibas),
00825                  i      eee, rofi(1,ibas), rkpr(1,0,ibas), rkmr(1,0,ibas),
00826                  o      rojp(1,1,ibas), sgpb(1,1,1,ibas),
00827                  o      fouv(1,1,1,ibas))
00828 c              call mkjp3(q,ngc, ngvecc, alat, qlat,
00829 c              i      lxx, lx(ibas),nxx, nx(0:lxx,ibas),
00830 c              i      bas(1,ibas),aa(ibas),bb(ibas),rmax(ibas),
00831 c              i      nr(ibas), nrx, rprodx(1,1,0,ibas),
00832 c              o      rojp(1,1,ibas), sgpb(1,1,1,ibas),

```

```

00833 c      o      fouv(1,1,1,ibas))
00834      enddo
00835
00836 C--- the Coulomb matrix
00837      if(allochk) write(6,*)' goto vcoulq_4'
00838      call vcoulq_4(q, nbloch, ngc,
00839      i          nbas, lx,lxx, nx,nxx,
00840      i          alat, qlat, voltot, ngvecc,
00841      i          strx, rojp,rojb, sgbb,sgpb, fouv, !sgpp,fouv,
00842      i          nblochpmx, bas,rmax,
00843      i          eee, aa,bb,nr,nrx,rkpr,rkmr,rofi,
00844      o          vcoul)
00845 c      call vcoulq2(q, nbloch, ngc,
00846 c      i          nbas, lx,lxx, nx,nxx,
00847 c      i          alat, qlat, voltot, ngvecc,
00848 c      i          strx, rojp,rojb, sgbb,sgpb, fouv, !sgpp,fouv,
00849 c      i          nblochpmx, bas,rmax,
00850 c      o          vcoul)
00851      if(allochk) write(6,*)' end of vcoulq_4'
00852      deallocate( strx, rojp,sgpb,fouv)
00853
00854 c$$$      else !===old version (allocation of sgpp and fouv are required) ===
00855 c$$$
00856 c$$$      if(allochk) write(6,*) 'allocate(rojp,sgpb,sgpp,fouv,fouv)'
00857 c$$$      allocate( rojp(ngc,      nlxx, nbas),
00858 c$$$      &          sgpb(ngc, nx, nlxx, nbas),
00859 c$$$      &          fouv(ngc, nx, nlxx, nbas),
00860 c$$$      &          sgpp(ngc, ngc, nlxx, nbas),
00861 c$$$      &          fouv(ngc, ngc, nlxx, nbas) )
00862 c$$$      do ibas = 1,nbas
00863 c$$$      write(6,*)' xxx goto mkjp',ibas
00864 c$$$      call mkjp2(q,ngc, ngvecc, alat, qlat,
00865 c$$$      i          lxx, lx(ibas),nxx, nx(0:lxx,ibas),
00866 c$$$      i          bas(1,ibas),aa(ibas),bb(ibas),rmax(ibas),
00867 c$$$      i          nr(ibas), nr, rprod(1,1,0,ibas),
00868 c$$$      o          rojp(1,1,ibas), sgpb(1,1,1,ibas),
00869 c$$$      o          fouv(1,1,1,ibas),
00870 c$$$      o          sgpp(1,1,1,ibas),fouv(1,1,1,ibas) )
00871 c$$$      enddo
00872 c$$$c--- the Coulomb matrix
00873 c$$$      write(6,*)' goto vcoulq'
00874 c$$$      call vcoulq(q, nbloch, ngc,
00875 c$$$      i          nbas, lx,lxx, nx,nxx,
00876 c$$$      i          alat, qlat, voltot, ngvecc,
00877 c$$$      i          strx, rojp,rojb, sgbb,sgpb,sgpp, fouv,fouv, nblochpmx,
00878 c$$$      o          vcoul)
00879 c$$$      if(allochk)
00880 c$$$      & write(6,*)'deallocate(strx, rojp,sgpb,sgpp, fouv,fouv)'
00881 c$$$      deallocate( strx, rojp,sgpb,sgpp, fouv,fouv)
00882 c$$$
00883 c$$$      endif !=====
00884
00885 c----check write
00886      trwv = 0d0
00887      do i = 1,nbloch
00888      trwv = trwv + vcoul(i,i)
00889      enddo
00890      write(6,(' " vcoul trwv=" ,i6,2d22.14')) iqx,trwv
00891      write(6,(' "### sum vcoul(1:ngb,      1:ngb) ",2d22.14,2x,d22.14'))
00892      & sum(vcoul(1:ngb,1:ngb)), sum(abs(vcoul(1:ngb,1:ngb)))
00893      write(6,(' "### sum vcoul(1:nbloch,1:nbloch) ",2d22.14,2x,d22.14'))
00894      & sum(vcoul(1:nbloch,1:nbloch)),sum(abs(vcoul(1:nbloch,1:nbloch)))
00895      write(6,*)
00896 ccccccccccccccccccccccccccccccccccc
00897 c      vcoul(:, nbloch+1:ngb)=0d0
00898 c      vcoul(nbloch+1:ngb,:)=0d0
00899 ccccccccccccccccccccccccccccccccccc
00900
00901 1101      continue
00902      ngbo=ngb
00903 C... Generate ppmt mattix oct2005 .....
00904      if(smbasis()) then
00905      allocate( ppmt(2,(lxx+1)**2,nbas,ngc) )
00906      ppmt = 0d0
00907      call mkppmt(alat,plat,qlat, q,
00908      i          ngc, ngvecc,
00909      i          rmax, nbas, bas, lx, lxx,
00910      o          ppmt) ! ppmt contains value and slope of e(i q+G r) at MT boundaries.
00911      ! ppmt(2,lxxaa,nbas)
00912 ccccccccccccccccccccccccccccccccccc
00913 c      write(6,*) 'lxx ppmtsum=',lxx, sum(abs(ppmt))
00914      write(6,*) 'nbln ngc',nbln,ngc
00915 ccccccccccccccccccccccccccccccccccc
00916
00917 C... Matching matrix pmtch. ppmt and prodmt
00918      pmat(:, nbln+1:nbln+ngc)=0d0
00919 c      write(6,*) 'sss nbln ngc',nbln,ngc

```

```

00920      do igc=1,ngc
00921 c      write(6,*) 'igc=',igc
00922      pmat(nbloch+igc, nbln+igc) = 1d0
00923      do ibas= 1, nbas
00924      do l = 0, lx(ibas)
00925      do m = -1, 1
00926 c      write(6,*) 'ibas l m=',ibas,l,m
00927      pval= ppmt(1, l**2 + l+1 +m, ibas,igc)
00928      pslo= ppmt(2, l**2 + l+1 +m, ibas,igc)
00929      do n = 1,nx(1,ibas)
00930      if(n==1.and.debug) write(6,('ttt2: '))
00931      pmat(ibl(m,n,l,ibas), nbln+igc)
00932      &      = rdmatch(n,l,l,ibas) * pval
00933      &      + rdmatch(n,2,l,ibas) * pslo
00934      if(debug.and.abs(pmat(ibl(m,n,l,ibas), nbln+igc))/=0d0)
00935      &      write(6,('ttt2: i1 i2 pmat=',2i5,2d13.5))
00936      &      ibl(m,n,l,ibas), nbln+igc, pmat(ibl(m,n,l,ibas), nbln+igc)
00937      enddo
00938      enddo
00939      enddo
00940      enddo
00941      enddo
00942      deallocate(ppmt)
00943      nn = nbln +ngc ! number for new smooth mixed basis.
00944      no = nbloch+ngc ! number for original size of mixed basis.
00945      if(debug) write(6,*) 'end of pmat'
00946
00947 C... oo(no,no). The original overlap matrix.
00948      allocate( pomat(nn,no) )
00949      allocate( ppovl(ngc,ngc),oo(no,no))
00950      call mkppovl2(alat,plat,qlat,
00951      i      ngc, ngvecc,
00952      i      ngc, ngvecc,
00953      i      nbas, rmax, bas,
00954      o      ppovl)
00955      oo = 0d0
00956      do ipl1 = 1,nbloch
00957      oo(ipl1,ipl1) = 1d0
00958      enddo
00959      do ix= 1,ngc
00960      do iy= 1,ngc
00961      oo(nbloch+ix, nbloch+iy) = ppovl(ix,iy)
00962      enddo
00963      enddo
00964      if(debug) write(6,*) 'end of oo'
00965
00966
00967 C... oon(nn,nn) is the overlap matrix with new basis
00968      allocate(oon(nn,nn))
00969      oon = matmul( dconjg(transpose(pmat(1:no,1:nn)))
00970      &      ,matmul(oo,pmat(1:no,1:nn)) )
00971
00972
00973 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00974 c Reduction of pmat by SVD ... not meaningful
00975 c      nnmx = 1000000
00976 c      epsmx= 1D20
00977 cc      write(6,*) ' sumchk pmat=',sum(abs(pmat(1:no,1:nn)))
00978 c      call zgesvdnn2(nn,nn, nnmx,epsmx,
00979 c      o      oon, ! pmat is reduced to pmat(1:no,1:nnn) by SVD.
00980 c      o      ngcnn)
00981 cc      call zgesvdnn2(no,ngc, nnmx,epsmx,
00982 cc      i      pmat(1:no,nbln+1:nbln+ngc), ! pmat is reduced to pmat(1:no,1:nnn) by SVD.
00983 cc      o      ngcnn)
00984 cc      nn= nbln+ngcnn
00985 c      write(6,*) ' svd ngc ngcnn=',ngc, ngcnn
00986 c      stop 'test end xxxxxx'
00987 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00988
00989 ccccccccccccccccccccccccccccccccccc
00990      if(.false.) then
00991      open(3011,file='oontest'//charnum5(igx))
00992      do ix=nbln+1,nn
00993      igc=ix-nbln
00994      qqx(1:3) = (q(1:3)+ matmul(qlat, ngvecc(1:3,igc)))
00995      absqq = sqrt(sum(qqx(1:3)**2))
00996 c      absqg2x(ix) =sum( (2*pi/alat *q0i(1:3,nq0i))**2)
00997      do iy=nbln+ 1,nn
00998      igc2=iy-nbln
00999      if(ix==iy) then
01000      write(3011,('on : ',2i8,3i3,2x,3i3,f13.5,3x,2f20.10))
01001      &      ix,iy, ngvecc(1:3,igc),ngvecc(1:3,igc2),absqq, oon(ix,iy)
01002      else
01003      write(3011,('off:', 2i8,3i3, 2f20.10))ix,iy,
01004      &      ngvecc(1:3,igc)-ngvecc(1:3,igc2), oon(ix,iy)
01005      endif
01006      enddo

```



```

01007         enddo
01008         close(3011)
01009     endif
01010 ccccccccccccccccccccccccccccccccccc
01011
01012
01013 C... Generat pomat
01014 !      zmult_new(K, ij) = \sum_I pomat(K,I)* zmult(I, ij)
01015 !      means <psi_i psi_j | K> where |K> denote new mixed basis.
01016 !      See sxcf_fal2 and x0kf.
01017 !      Be carefull its transpose procedure---it is a little confusing...
01018         call pmatorth(oo,oon, pmat(1:no,1:nn), no, nn,
01019             o      pomat)
01020
01021 ccccccccccccccccccccccccccccccccccc
01022 ctttt
01023 c      pomat=0d0
01024 c      do ix= 1,ngb
01025 c          pomat(ix,ix)=1d0
01026 c      enddo
01027 c      do ix= 1,ngb
01028 c          do iy= 1,ngb
01029 c              if(pmat(ix,iy)/=0d0 )
01030 c                  & write(6,"(' ttt: pomat=',2i3,2d13.6)")
01031 c                  & ix,iy,pmat(ix,iy)
01032 c              enddo
01033 c          enddo
01034 c          write(6,"(' ttt:sumchk=',2d13.6,2i4)")
01035 c          & sum(pomat(:,:)), no,nn
01036 ccccccccccccccccccccccccccccccccccc
01037
01038         if( iqx <= ngibz ) deallocate(oon)
01039         deallocate(ppovl,oo)
01040 C... Store matching matrix
01041         write(ifpomat) q,nn,no,iqx
01042         write(ifpomat) pomat
01043         deallocate(pomat)
01044     endif
01045
01046 c$$$      if(newaniso()) then
01047 c$$$          continue
01048 c$$$      elseif(bzcase()==1.and.iqx==1)then
01049 c$$$          cycle
01050 c$$$      endif
01051
01052 !! == Write out VCCFP ==
01053         if(debug) write(6,*) 'write out vcoul'
01054         if(smbasis()) then
01055             ngb= nn
01056             allocate(vcoulnn(ngb,ngb))
01057             vcoulnn= matmul(transpose(dconjg(pmat(1:no,1:nn)))
01058                 & ,matmul(vcoul(1:no,1:no),pmat(1:no,1:nn)))
01059             vcoul(1:ngb,1:ngb)= vcoulnn
01060             deallocate(vcoulnn)
01061         endif
01062         if(wvcc) then
01063             write(ifvcfpout) ngb
01064             write(ifvcfpout) vcoul(1:ngb,1:ngb),q
01065         endif
01066         write(6,"(' ngc ngb/ngbo=',6i6)") ngc,ngb,ngbo
01067
01068 c Mix0vec -----
01069 !! diagonalize the Coulomb matrix
01070         if(.true.) then
01071             if( iqx > ngibz .or. iqx==1) then !feb2012 add iqx==1 for newansio()=T
01072                 if(allochk) write(*,*) 'allocate( ppovl(ngc,ngc))'
01073                 allocate( oo(ngb,ngb) )
01074                 allocate( ppovl(ngc,ngc) )
01075                 call mkppovl2(alat,plat,qlat,
01076                     &      ngc, ngvecc,
01077                     &      ngc, ngvecc,
01078                     &      nbas, rmax, bas,
01079                     o      ppovl)
01080                 if(smbasis()) then
01081                     oo = oon
01082                     deallocate(oon)
01083                 else
01084                     oo = 0d0
01085                     do ipl1=1,nbloch
01086                         oo(ipl1,ipl1) = 1d0
01087                     enddo
01088                     do ix=1,ngc
01089                         do iy=1,ngc
01090                             oo(nbloch+ix, nbloch+iy) = ppovl(ix,iy)
01091                         enddo
01092                     enddo
01093                 endif

```

```

01094
01095 allocate( oox(ngb,ngb) )
01096 oox = oo
01097 write(6,*)' --- goto eigen check1 --- '
01098 allocate( vcoul0(ngb,ngb) )
01099 vcoul0 = vcoul(1:ngb,1:ngb)
01100 if(allochk)
01101 & write(*,*) 'allocate(hh(ngb,ngb),oo(ngb,ngb),oox,zz,eb,zzr)'
01102 allocate(hh(ngb,ngb),zz(ngb,ngb),eb(ngb),zzr(ngb))
01103 hh = - vcoul0
01104 c nmx = 15
01105 nmx = ngb
01106 call diagcv(oo,hh,zz,ngb, eb,nmx,ld99,nev)
01107 do ipl1=1,nev
01108 if(ipl1==1) write(6,*)' ... '
01109 if(ipl1>10.and.ipl1<nev-5) cycle
01110 write(6,'(i4,d23.16)')ipl1,-eb(ipl1)
01111 enddo
01112 write(6,"(' nev ngv q=',2i5,3f10.6)")nev,ngb,q
01113
01114 c$$$!! Modify -eb
01115 c$$$ if(ixq<=nqibz) then
01116 c$$$ do igbz=1,nqbz !! check
01117 c$$$ if(sum(abs(qbzwww(:,igbz)-q))<1d-6) then
01118 c$$$ igbzx=igbz
01119 c$$$ goto 888
01120 c$$$ endif
01121 c$$$ enddo
01122 c$$$ stop 'hvccfp0:sum(abs(qbzwww(:,iq)-qbz(:,iq)))>1d-6'
01123 c$$$ 888 continue
01124 c$$$ if(abs((-eb(1)+eb(2))/eb(2))<1d-2) then
01125 c$$$!! Center. touching case. Respect smoothness when we change nln2n3 division.
01126 c$$$ eb(1)=eb(1)*wqfac(igbzx)
01127 c$$$ eb(2)=eb(2)*wqfac(igbzx)
01128 c$$$ else
01129 c$$$ eb(1)=eb(1)*wqfac(igbzx)
01130 c$$$ endif
01131 c$$$ endif
01132
01133 ! ! == save zz == apr2012takao
01134 c if( newaniso().and.ixq==1 ) then
01135 c if(sum(q**2)>1d-10) then
01136 c stop ' hvccfp0: sanity check. |q(ixq)| /= 0'
01137 c endif
01138 write(ifvcoud) ngb
01139 write(ifvcoud) q
01140 write(ifvcoud) -eb
01141 write(ifvcoud) zz
01142
01143 c$$$$cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01144 c$$$ write(6,*)' dddddd q=',q
01145 c$$$ do ix=1,ngb
01146 c$$$ do iy=1,ngb
01147 c$$$ aaaa= sum( dconjg(zz(1:ngb,ix))*matmul( oox,zz(1:ngb,iy)) )
01148 c$$$ if(ix==iy .and. abs(aaaa-ld0) >1d-8 ) then
01149 c$$$ write(*,*)' dddd zcousum check',ix,iy,aaaa
01150 c$$$ endif
01151 c$$$ if(ix/=iy .and. abs(aaaa) >1d-8 ) then
01152 c$$$ write(*,*)' dddd zcousum check',ix,iy,aaaa
01153 c$$$ endif
01154 c$$$ enddo
01155 c$$$ enddo
01156 c$$$$cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01157
01158
01159
01160 write(6,*)
01161 write(6,'(" eig0 must be equal to the largest =", 2d24.16)')
01162 & sum( dconjg(zz(1:ngb,1))*matmul( vcoul0,zz(1:ngb,1)) )
01163 write(6,'(" zz norm check=",d24.16)')
01164 & sum( dconjg(zz(1:ngb,1))*matmul(oox,zz(1:ngb,1)) )
01165 write(6,*)
01166 c write(6,'(" --- vcoul(exact no eee)=",d14.6," absq2=",d24.16)')
01167 c & fpi*voltot/(sum(tpiba**2*q(1:3)**2))
01168 c & , (sum(tpiba**2*q(1:3)**2))
01169 write(6,'(" --- vcoul(exact)=",d14.6," absq2=",d24.16)')
01170 & fpi*voltot/(sum(tpiba**2*q(1:3)**2)-eee)
01171 & , (sum(tpiba**2*q(1:3)**2)-eee)
01172 write(6,'(" --- vcoul(cal ) =",2d14.6)')
01173 & sum( dconjg(zz(1:ngb,1))*matmul( vcoul0,zz(1:ngb,1)) )*voltot
01174 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01175 c do igc=1,ngb
01176 c qqx(1:3) = (q(1:3)+ matmul(qlat, ngvecc(1:3,igc)))
01177 c write(6,'(" --- vcoul(exact) xxx =",d14.6," absq2=",d24.16)')
01178 c & fpi*voltot/(sum(tpiba**2*(qqx(1:3)**2)-eee))
01179 c & , (sum(tpiba**2*(qqx(1:3)**2)-eee))
01180 c write(6,'(" --- vcoul(cal ) xxx =",2d14.6)')

```

```

01181 c      &      sum( dconjg(zz(1:ngb,igc))*matmul( vcoul0,zz(1:ngb,igc)) ) *voltot
01182 c      enddo
01183 c      ccccccccccccccccccccccccccccccccccccccccccc
01184 c      deallocate(vcoul0)
01185
01186 c      if( igx-nqibz>=1 ) then
01187 c          if( wqt(igx-nqibz)==0d0) then ! MIZUHO-IR
01188
01189 c --- To get the vector <Mixed basis| q=0> -----
01190 cki          if(is_mix0vec()==0) then      !used original befor oct2006
01191 c          if(.not.is_mix0vec()) then      !used original befor oct2006
01192 c ! See switch.F ---> this is not used now.
01193 c          ifgb0vec_a =ifgb0vec1
01194 c          ifgb0vec_b =ifgb0vec
01195 cki          elseif(is_mix0vec()==1) then !oct2006 new case
01196 c          else
01197 c ! ismix0vec=1 is to avoid problem at BZ boundary when is_mix0vec()==0.
01198 c          ifgb0vec_a =ifgb0vec
01199 c          ifgb0vec_b =ifgb0vec1
01200 c      endif
01201 c1... Case1 to write ifgb0vec -----
01202 c      write(6,*)' voltot=',voltot
01203 c      if(ngc==0) then
01204 c          continue
01205 c      else
01206 c          do igc=1,ngc
01207 c              if( sum(abs( ngvecc(1:3,igc) ))==0 ) then
01208 c                  igc0=igc
01209 c                  exit
01210 c              endif
01211 c          enddo
01212 c          write(6,*)' igc0=',igc0,ngvecc(1:3,igc0)
01213 c          zzr(nbloch+1:nbloch+ngc) = ppovl(1:ngc,igc0)
01214 c      endif
01215
01216 c      allocate( gbvec(ngb), b0mat(nbloch) )
01217 c      write(6,*)' goto mkb0'
01218
01219 c ... get a vector <Product Basis| q+0>
01220 c      call mkb0( q, lxx,lx,nxx,nx, aa,bb,nr,nrx,rprodx,
01221 c          i      alat,bas,nbas,nbloch,
01222 c          o      b0mat)
01223 c      zzr(1:nbloch) = b0mat(1:nbloch)
01224 c      ccccccccccccccccccccccccccccccc
01225 c      do igc=1,ngb
01226 c          write(6, "('ssss: ',i5,2d14.6)") igc, zzr(igc)
01227 c      enddo
01228 c      ccccccccccccccccccccccccccccccc
01229 c      allocate(ooxi(ngb,ngb))
01230 c      ooxi=oox
01231 c      call matcinv(ngb,ooxi)
01232 c      gbvec = matmul(ooxi, zzr)
01233
01234 c      ccccccccccccccc
01235 c      do igc=1,ngb
01236 c          write(6, "('ssss: ',i5,2d14.6)") igc, gbvec(igc)
01237 c      enddo
01238 c      ccccccccccccccc
01239 c      deallocate(ooxi)
01240 c      dnorm = sqrt( sum(dconjg(gbvec)*zzr) )
01241 c ! remove /dnorm at 14June2008. See main/hx0fp0.
01242 c ! dnorm corresponds to volume (or sum of MT volume if no IPW).
01243 c      gbvec = gbvec /dnorm
01244 c      zzr = zzr /dnorm
01245 c ! Not dnorm=1 at 14June2008. See main/hx0fp0.
01246 c      dnorm=1
01247 c      write(ifgb0vec_a,"(3d24.16,2i10,d24.16)") q, ngb,igc0,dnorm
01248 c      write(ifgb0vec_a,"(4d24.16)") (gbvec(i),zzr(i),i=1,ngb)
01249 c      deallocate( gbvec, b0mat)
01250 c1-----
01251
01252 c2... --- Case2 to write ifgb0vec c2 is problematic at BZ boundary...-----
01253 c      dnorm = 1d0
01254 c      zzr(:) = matmul(oox, zz(:,1))
01255 c      igc0 = 999999 !dummy now
01256 c phasex ---just to clean. this is irrelevant
01257 c      phasex =1d0
01258 c      do i=1,ngb
01259 c          if(abs(zz(i,1)) > 1d-3) phasex = abs(zz(i,1))/zz(i,1)
01260 c      enddo
01261 c      do i=1,ngb
01262 c          zz(i,1)= phasex * zz(i,1)
01263 c          zzr(i) = phasex * zzr(i)
01264 c      enddo
01265 c      write( ifgb0vec_b,"(3d24.16,2i10,d24.16)") q, ngb,igc0,dnorm
01266 c      write( ifgb0vec_b,"(4d24.16)") (zz(i,1),zzr(i),i=1,ngb)
01267 c      endif

```

```

01268         endif ! MIZUHO-IR
01269         if(allochk) !bugfix ---this was in inside or above if 7Feb2006
01270             & write(*,*)'deallocate(hh,oo,zz,eb,oox,zzr)'
01271             deallocate(hh,oo,zz,eb,oox,zzr)
01272             deallocate(ppovl)
01273 c2-----
01274         endif
01275         idummy=iclose(trim(vcloudfile))
01276 1001 continue
01277         deallocate(ngvecc)
01278         call cputid(0)
01279         call flush(6)
01280         call mpi_finalize
01281         if(imode==202) call rx0( ' OK! hvccfp0 imode=202 only for Q0P')
01282         if(imode==0) call rx0( ' OK! hvccfp0 imode=0')
01283         if(imode==3) call rx0( ' OK! hvccfp0 imode=3')
01284         end
01285
01286         subroutine checkagree(a,b,char)
01287             real(8):: a(3),b(3)
01288             character*(*) :: char
01289             if(sum(abs(a-b))>1d-6) then
01290                 write(6,*)' Error in checkagree:',char
01291 Cstop2rx 2013.08.09 kino stop ' Error in checkagree:'
01292                 call rx( ' Error in checkagree:')
01293             endif
01294         end
01295
01296         subroutine mkradmatch( p, nxdim,
01297             o rdmatch)
01298 C- make rdmatch
01299 C-----
01300 Ci p(1,i): phi at mt for i-th basis
01301 Ci p(2,i): dphi/dr at mt for i-th basis
01302 Co rdmatch(nxdim,nxdim)
01303 C-----
01304 Cr phinew_j(r) =sum_i phi_i(r)* rdmatch(i,j)
01305 Cr phinew_l(rmt) =1 phinew_2(rmt) =0
01306 Cr d phinew_l(rmt)/dr =0 d phinew_2(rmt)/dr=1
01307 Cr for k >=3
01308 Cr phinew_k(rmt) =0
01309 Cr d phinew_k(rmt)/dr =0
01310 C-----
01311         implicit none
01312         integer(4):: nxdim,lbas,i,il,i2,ix
01313         real(8):: p(1:2, 1:nxdim), rdmatch(1:nxdim,1:nxdim)
01314         real(8):: pd,p1,pld,p2,p2d,s,t, eps=1d-3,delta
01315 Cr old new
01316 c write(6,('mkradmatch: nxdim=',i4)) nxdim
01317         if(nxdim <=0) return
01318 Cstop2rx 2013.08.09 kino if(nxdim ==1) stop 'mkradmatch err nxdim==1'
01319         if(nxdim ==1) call rx( 'mkradmatch err nxdim==1')
01320         rdmatch=0d0
01321 C... pivot--- get better set of phi for augmentation
01322         do
01323             i1= nxdim
01324             i2= nxdim-1
01325             p1 = p(1, i1)
01326             p2 = p(1, i2)
01327             pld= p(2, i1)
01328             p2d= p(2, i2)
01329             write(6,('mkradmatch: i1 p1 pld=',i3,2d13.6)) i1,p1,pld
01330             write(6,('mkradmatch: i2 p2 p2d=',i3,2d13.6)) i2,p2,p2d
01331             delta = p1*p2d-p2*pld
01332             if(abs(delta) <eps*p1*p2) then
01333                 if(i2==1) then
01334                     write(6,(' i1 i2=',2i5,2d13.6)) i1,i2,pld/p1,p2d/p2
01335 Cstop2rx 2013.08.09 kino stop 'mkradmatch: err poor linear dep'
01336                     call rx( 'mkradmatch: err poor linear dep')
01337                 endif
01338                 i2=i2-1
01339             endif
01340             exit
01341         enddo
01342 C...
01343         call phimatch(ld0,0d0, p1,pld,p2,p2d, s,t)
01344         rdmatch(i1, 1)= s
01345         rdmatch(i2, 1)= t
01346         write(6,('mkradmatch: 1 0 st=',2d13.5)) s,t
01347         call phimatch(0d0,ld0, p1,pld,p2,p2d, s,t)
01348         rdmatch(i1, 2)= s
01349         rdmatch(i2, 2)= t
01350         write(6,('mkradmatch: 0 1 st=',2d13.5)) s,t
01351
01352         ix=2
01353         do i= 1,nxdim
01354             if(i==i1.or.i==i2) cycle

```

Generated on Fri Nov 11 2016 17:43:19 for ecalj/fpgw/ code document by Doxygen

```

01442      lwork=4*no
01443      allocate(work(lwork),rwork(5*no),pmatx(no,nn))
01444      pmatx =pmat
01445      call zgesvd('A','A',no,nn,pmat,no,ss,uu,no,vt,nn,work,lwork,rwork,info)
01446      nnn=-999
01447      do i=1,nn
01448          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)
01449      !      vtt(i,:)=ss(i)*vt(i,:)
01450      !      if(nnn== -999.and.ss(i)<epsmx) nnn = i-1
01451      !      enddo
01452      c      write(6,*) 'nnn=',nnn
01453      Cstop2rx 2013.08.09 kino      if(nnn==0) stop 'strange: nnn=0'
01454      if(nnn==0) call rx('strange: nnn=0')
01455      if(nnn>nnmx) nnn=nnmx
01456      pmat=pmatx
01457      c      pmat(:,1:nnn) = uu(:,1:nnn)
01458      !      write(6,"('sumcheck zzz zzz-uu*s*vt=',d13.5,d13.5)")
01459      !      & sum(abs(zw0bk)), sum(abs(zw0bk - matmul(uu,vtt)))
01460      !      if(abs(sum(abs(zw0bk - matmul(uu,vtt))))>1d-8*sum(abs(zw0bk)))
01461      !      & stop 'sumcheck zzz zzz-uu*s*vt= error'
01462      !      deallocate(vtt)
01463      end
01464
01465
01466      c-----
01467      subroutine mkb0( q, lxx,lx,nxx,nx, aa,bb, nrr,nrx,rprodx,
01468      i      alat,bas,nbas,nbloch,
01469      o      b0mat)
01470      C--make the matrix elementes < B_q | exp(iq r)>
01471      use m_lldata,only: ll
01472      implicit none
01473      integer(4) :: nlx,l,n,m,nr,ir,lm,ibl1,ibas,nrx,nbloch
01474
01475      integer(4) :: nbas,lxx, lx(nbas), nx, nx(0:lxx,nbas),nrr(nbas)
01476      real(8) :: rprodx(nrx,nxx,0:lxx,nbas),aa(nbas),bb(nbas),
01477      & phi(0:lxx),psi(0:lxx), bas(3,nbas),
01478      & alat,
01479      & pi,fpi,tpiba,qgl(3),q(3),absqg,r2s,a,b
01480      c
01481      complex(8) :: b0mat(nbloch),img=(0d0,1d0) ,phase
01482      c
01483      integer(4),allocatable:: ibasbl(:), nbl(:), lbl(:), lmb1(:)
01484      real(8),allocatable:: ajr(:,:),rofi(:),rob0(:,:,:)
01485      real(8),allocatable:: cy(:),yl(:)
01486      complex(8),allocatable:: pjyl(:,:)
01487      c$$$#ifndef COMMONLL
01488      c$$$      integer(4) ll(51**2)
01489      c$$$      common/llblock/ll
01490      c$$$#else
01491      c$$$      integer(4) ll
01492      c$$$#endif
01493
01494      c-----
01495      write(6,*) 'mkb0:'
01496      pi = 4d0*datan(1d0)
01497      fpi = 4*pi
01498      nlx = (lxx+1)**2
01499      c
01500      tpiba = 2*pi/alat
01501      qgl(1:3) = tpiba * q(1:3)
01502      absqg = sqrt(sum(qgl(1:3)**2))
01503      c
01504      allocate(ajr(1:nrx,0:lxx), pjyl(nlx,nbas),rofi(nrx),
01505      & ibasbl(nbloch), nbl(nbloch), lbl(nbloch), lmb1(nbloch),
01506      & cy(nlx),yl(nlx),rob0(nxx,0:lxx,nbas))
01507      c
01508      call sylmnc(cy,lxx)
01509      call sylm( qgl/absqg,yl,lxx,r2s) !spherical factor Y( q+G )
01510      c
01511      do ibas = 1,nbas
01512          a = aa(ibas)
01513          b = bb(ibas)
01514          nr= nrr(ibas)
01515          rofi(1) = 0d0
01516          do ir = 1, nr
01517              rofi(ir) = b*( exp(a*(ir-1)) - 1d0)
01518              call bessl(absqg**2*rofi(ir)**2,lx(ibas),phi,psi)
01519              do l = 0,lx(ibas)
01520      c ...  bessell function
01521              ajr(ir,l) = phi(l)* rofi(ir) **(l+1 )
01522              ! ajr = j_l(sqrt(e) r) * r / (sqrt(e))**l
01523              enddo
01524          enddo
01525
01526      c ... Coefficients for j_l yl on MT in the expansion of of exp(i q r).
01527      phase = exp( img*sum(qgl(1:3)*bas(1:3,ibas))*alat )

```

```

01528         do lm = 1,(lx(ibas)+1)**2
01529             l = ll(lm)
01530             pjyl(lm,ibas) = fpi *img**l *cy(lm)*yl(lm) *phase  *absqg**l
01531         enddo
01532 c ... rob0
01533         do l = 0,lx(ibas)
01534             do n = 1,nx(l,ibas)
01535                 call gintxx( ajr(l,l), rprodx(l,n,l,ibas), a,b,nr,
01536                     o rob0(n,l,ibas) )
01537             enddo
01538         enddo
01539     enddo
01540
01541 c ... index (mx,nx,lx,ibas) order.
01542     ibl1 = 0
01543     do ibas= 1, nbas
01544         do l = 0, lx(ibas) ! write(6,'(" l ibas nx =",3i5)') l,nx(l,ibas),ibas
01545             do n = 1, nx(l,ibas)
01546                 do m = -l, l
01547                     ibl1 = ibl1 + 1
01548                     ibasbl(ibl1) = ibas
01549                     nbl(ibl1) = n
01550                     lbl(ibl1) = l
01551                     lbl1(ibl1) = l**2 + l+1 +m ! write(6,*)ibl1,n,l,m,lbl1(ibl1)
01552                 enddo
01553             enddo
01554         enddo
01555     enddo
01556 c ... pjyl * rob0
01557     do ibl1= 1, nbloch
01558         ibas= ibasbl(ibl1)
01559         n = nbl(ibl1)
01560         l = lbl(ibl1)
01561         lm = lbl1(ibl1)
01562         b0mat(ibl1) = pjyl(lm,ibas) * rob0(n,l,ibas)
01563     enddo
01564     deallocate(ajr, pjyl,rofi,
01565         & ibasbl, nbl, lbl, lbl1,
01566         & cy,yl,rob0)
01567     end

```

## 4.33 main/hx0fp0.m.F File Reference

### Functions/Subroutines

- program [hx0fp0](#)
- real \*8 function [eclda\\_bh](#) (rs)
- real \*8 function [eclda\\_pz](#) (rs)
- subroutine [wecqw](#) (ifcor,
- subroutine [getsqovlp](#) (q, ngc, ngb, sqovlp)
- subroutine [tr\\_chkwrite](#) (tagname, zw, iw, freqq, nblochpmx, nbloch, ngb, iq)
- complex(8) function, dimension(1) [matchinvf](#) (a)
- subroutine [diagno00](#) (nbloch, wpvc, eval)

### 4.33.1 Function/Subroutine Documentation

4.33.1.1 subroutine [diagno00](#) ( integer *nbloch*, complex(8), dimension(nbloch,nbloch) *wpvc*, real(8), dimension(nbloch) *eval* )

Definition at line [2372](#) of file [hx0fp0.m.F](#).

4.33.1.2 real\*8 function [eclda\\_bh](#) ( real(8) *rs* )

Definition at line [2204](#) of file [hx0fp0.m.F](#).

Here is the caller graph for this function:

#### 4.33.1.3 `real*8 function eclda_pz ( real(8) rs )`

Definition at line [2213](#) of file [hx0fp0.m.F](#).

Here is the caller graph for this function:

#### 4.33.1.4 `subroutine getsqovlp ( real(8), dimension(3) q, integer ngc, integer ngb, complex(8), dimension(ngb,ngb) sqovlp )`

Definition at line [2254](#) of file [hx0fp0.m.F](#).

Here is the call graph for this function:

#### 4.33.1.5 `program hx0fp0 ( )`

Definition at line [5](#) of file [hx0fp0.m.F](#).

Here is the call graph for this function:

#### 4.33.1.6 `complex(8) function, dimension(1) matcinvf ( complex(8), dimension(:, :) a )`

Definition at line [2347](#) of file [hx0fp0.m.F](#).

#### 4.33.1.7 `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 [2308](#) of file [hx0fp0.m.F](#).

#### 4.33.1.8 `subroutine wecqww ( ifcor )`

Definition at line [2223](#) of file [hx0fp0.m.F](#).

Here is the caller graph for this function:

## 4.34 `hx0fp0.m.F`

```
00001 !! Calculate x0, \epsilon, spin susceptibility.
00002 !!
00003 !! eps_lmf_cphimp mode is now commented out; you may need to recover this if necessary
00004 !! (only epsPP_lmf_chimp mode works).
00005 program hx0fp0
00006   use m_readfermi, only: readfermi, ef
00007   use m_readqg, only: readqg, readngmx
00008   use m_readeigen, only: readeval, init_readeigen, init_readeigen2
00009   use m_read_bzdata, only: read_bzdata,
00010   & ngrp2=>ngrp, ngbz, ngibz, ngbzw, nteti, ntetf, n1, n2, n3, qbas, ginv,
00011   & dq_, qbz, wbz, qibz, wibz, qbzw,
00012   & idtetf, iblbz, idteti,
00013   & nstar, irk, nstbz
```



```

00014     use m_genallcf_v3,only: genallcf_v3,
00015     & nclass,natom,nspin,nl,nm, ngrp,
00016     & nlmtol,nlrmx, nctot,niw, !nw_input=>nw,
00017     & alat, delta,deltaw,esmr,symgrp,clabl,iclass, !diw,dw,
00018     & invg, il,in,im,nlrm,
00019     & plat, pos,ecore, symgg
00020     use m_keyvalue,only: getkeyvalue
00021     use m_pbindx,only: pbindx !,norbt,l_tbl,k_tbl,ibas_tbl,offset_tbl,offset_rev_tbl
00022     use m_readqgcou,only: readqgcou
00023     use m_mpi,only: mpi_hx0fp0_rankdivider2,mpi_task,mpi_initialize,mpi_finalize,mpi_root,
00024     & mpi_broadcast,mpi_dblecomplexsend,mpi_dblecomplexrecv,mpi_rank,mpi_size,
00025     & mpi_ranctab,mpi_consoleout,mpi_barrier
00026 !! Base data to generate matrix elements zmel*. Used in "call get_zmelt".
00027     use m_rdpp,only: rdpp, !NOTE: "call rdpp" generate following data.
00028     & nblocha,lx,nx,ppbrd,mdimx,nbloch,cgr
00029 !! Generate matrix element for "call get_zmelt".
00030     use m_zmel,only: !NOTE: these data set are stored in this module, and used when
00031     & nband,itq,ngcmx,ngpmx, ppovlz,
00032     & ppbir,shtvg, mlat,tiat , ntq
00033 !! frequency
00034     use m_freq,only: getfreq, !NOTE: call getfreq generate following data.
00035     & frhis,freq_r,freq_i, nwhis,nw_i,nw,npm,wiw !, frhis0,nwhis0 !output of getfreq
00036 !! tetwt
00037     use m_tetwt,only: tetdeallocate,gettetwt, !followings are output of
    'L871:call gettetwt')
00038     & whw,ihw,nhw,jhw,ibjb,nbnbx,nhwtot,n1b,n2b,nbnb
00039 !! w0 and w0i (head part at Gamma point)
00040     use m_w0w0i,only: w0w0i,
00041     & w0,w0i
00042
00043     use m_lldata,only: ll
00044     implicit none
00045 !! -----
00046 !! We calculate chi0 by the folloing three steps.
00047 !! gettetwt: tetrahedron weights
00048 !! x0kf_v4h: Accumulate Im part of the Lindhard function. Im(chi0) or Im(chi0*+-)
00049 !! dpsion5: calculate real part by the Hilbert transformation from the Im part
00050 !! eibz means extented irreducible brillowin zone scheme by C.Friedlich. (not so efficient in cases).
00051 !!-----
00052
00053 ccccc this may be wrong or correct ccccccccccc
00054 cr be careful for the indexing...
00055 cr     a routine idxlrmc(nindxv,nindxc,... in index.f
00056 cr     specifies the order of the(core wave)+(argumentation wave) in each mt.
00057 cr     the total number of the wave are mnl(ic)= mnlc(ic) + mnlv(ic).
00058 cr     the indexing starts with core first and then valence on top of core
00059 cr     so n-index in "in" for valence electron is different from "inv".
00060 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00061     real(8):: q(3), qgbn(3),qx(3)
00062     real(8):: ua=ld0 ! this is a dummy.
00063     integer:: ifrb(2),ifcb(2),ifrbh(2),ifchb(2) !,ifev(2)
00064     integer:: ndble=8
00065     integer:: nword
00066     real(8),allocatable:: vxcfp(:,,:),
00067     & wgt(:,), wgt0(:,,:),q0i(:,)
00068     integer,allocatable:: ngvecpb(:,,:),ngveccb(:,) !,ngveccB(:,,:),)
00069     & , ngvecp(:,), ngvecc(:,), !,ngpn(:,),ngcni(:,),iqib(:,)
00070     & ifppb() !ongveccBr(:,,:),nx(:,),nblocha(:,),lx(:,)
00071     complex(8),allocatable:: geigb(:,,:),geig(:,),vcoul(:,),
00072     & zw(:,),zw0(:,),
00073     & zxq(:,),zxqi(:,),)
00074     real(8),allocatable:: eqt(:,) !ppbrd (:,:,,:,:),cgr(:,,:),)
00075     & ppbrdx(:,,:),aaa(:,),symope(:,),
00076     & ppb(:,),pdb(:,),dpb(:,),ddb(:,),
00077     & qbze(:,),qibze(:,) !,ecore(:,) freqr(:,)freqi(:,) !rw(:,),cw(:,) ---->zw
00078     complex(8),allocatable:: trwv(:,),trwv2(:,),rcxq(:,),)
00079 c     & ,rcxqmean(:,),rcxqmean(:,),) !now rcxqmean is treated as a case of rcxq(nmbas,nmbas)
00080
00081 ! tetrahedron method
00082     logical:: tetra !,tmpwwk=.true.! If tmpwwk=.true., this use a temporary file tmp.wwk
00083     ! so as to reduce the memory usage.
00084     complex(8) :: fff,img=(0d0,ld0)
00085     complex(8),allocatable:: wwkc(:,)
00086     integer,allocatable::
00087     & noccxvv() !nlb(:,),n2b(:,),nbnb(:,),nbnbt(:,),
00088     real(8) ::qbzx(3),anfvec(3)
00089     logical:: debug
00090     integer,allocatable:: ibasf()
00091     real(8),allocatable:: transaf(:,)
00092     logical:: realomega, imagomega
00093     complex(8),allocatable:: epsi(:,),gbvec(:,),zrz(:,),x0mean(:,),zrz0()
00094     complex(8) :: epxxx,vcmean, vcmmmm
00095     complex(8),allocatable:: vcmmmm()
00096     character*11 fileps
00097     character*11 fileps23
00098     character*16 filepsnolfc
00099     character*11 filele

```

```

00100     character(5) :: charnum5
00101     character(20):: xxt
00102
00103     real(8) :: emin, emax,emin2,emax2
00104     real(8) :: omg2max,omg1max,wemax
00105     real(8), allocatable :: freqr2(:) , ekxxx(:,,:)
00106
00107     logical::imagonly=.false.,realonly=.false. !,readgwinput
00108     integer::iopen,maxocc2,iclose,
00109     & ixc,ixxini,ixxend,ixxendx,
00110     & ifhbe,
00111     & nprecb,mrecb,mrece,nlmtot,nqgbzt,!nband,
00112     & nq0i,i,nq0ix,neps,ngrpmx,mxx,nqbze,nqibze,ini,ix,ngrpx !ngcmx,
00113     & ,nblochpmx,ndummy1,ndummy2,ifcphi,is,nwp, !ifvcfpout,,mdimx,nbloch
00114     & ifepscond,nxx !,ifvxcput,ifgb0vec
00115     & ,nw0,iw,ifinin,iw0,ifwwk,noccxv,noccx
00116     & ,nprecx,mrecl,ifwd,ifrcwi,ifrcw,nspinmx,ifianf,ibas
00117     & ,ibas1,irot,iq,ngb,ixxc2,ifepsdatnolfc,ifepsdat,ngbin,igc0dummy
00118     & ,kx,isf,kqxx,kp,job,noccxvx(2)=-9999,nwmax !,ifev1,ifev2 nbnbx,nhwtot,
00119     & ,ihis,jhwtot,ik,ibib,ibl,ib2,ichkhis,ihww,j,imode
00120 c    & ,ngpmx !, ifchipmlog
00121
00122     real(8):: dum1,dum2,dum3,wqtsum,epsrng,dnorm,
00123     & dwry,dwh,omg_c,omg2
00124
00125     integer:: incwfin, verbose
00126
00127     integer:: ngc,mrecg !bzcase,
00128     real(8):: quu(3), deltaq(3)!,qq(3) !,qqq(3)=0d0
00129     logical:: omitqbz=.false., noq0p
00130
00131     logical,allocatable :: iwtg(:,,:,:)
00132     complex(8),allocatable:: wgt(:,,:,:)
00133
00134     real(8),allocatable:: qbz2(:,,:)
00135     logical :: qbzreg !if true, we use off-gamma mesh.
00136     integer:: nbcut,nbcut2
00137
00138     integer,allocatable:: nstibz(:) !Nov2004 Miyake's tote
00139     real(8),allocatable:: ecgw(:,,:) !,wiw(:)
00140     real(8) :: erpaqw, trpvqw, trlogqw,rydberg,hartree
00141     & ,pi,efz,qfermi,alpha,rs,voltot,ecelgas,efx,vaIn
00142     integer:: igbz,igindx,iflegas,nmx
00143     & ,ifcor,nqitot,isx,ntot,ieclg,iww,iqq,ieceig,ecorr_on=-1
00144     real(8) :: eclda_bh,eclda_pz,wk4ec,faca
00145     real(8),allocatable:: eval1(:)
00146     complex(8),allocatable:: ovlpc(:,:),evecc(:,:)
00147     integer:: nev !, ifdpin
00148
00149     real(8),allocatable:: ecut(:),ecuts(:) ,totexc(:), trpv(:),trlog(:)
00150     integer:: necut,iecut
00151
00152     integer:: ifv,lxx,ibasx,ilmx,ilm_r,nx_r,lb,nb,mb
00153     integer,allocatable:: nxx_r(:)
00154     real(8),allocatable:: svec(:,:),spinvec(:,:),consvec(:,:),cvec(:,:)
00155     character*3:: charnum3
00156     character*4:: charnum4
00157     complex(8),allocatable:: jcoup(:,:), mcm(:,:,:)
00158     real(8)::chg1,chg2,spinmom,schi=1d0
00159 c$$$#ifdef COMMONLL
00160 c$$$     integer::ll(51**2)
00161 c$$$     common/llblock/ll
00162 c$$$#else
00163 c$$$     integer :: ll
00164 c$$$     external ll
00165 c$$$#endif
00166     complex(8),allocatable:: ovlp(:,:),evec(:,:),ovlpi(:,:)
00167     real(8),allocatable::eval(:)
00168     integer:: new,nmxx,ii,iy,ipl1,ixx
00169
00170     complex(8),allocatable :: ppovl(:,:),oo(:,:),x0meanx(:,:),x0inv(:,:),ppovlzinv(:,:)
00171     real(8)::qxx(3),ssm
00172 ! svd. not used now
00173     real(8),allocatable::ss(:),rwork(:),ss0(:)
00174     complex(8),allocatable:: uu(:,:),vt(:,:),work(:),zw0bk(:,:),ddd(:,:)
00175     & ,vtt(:,:),zzz(:,:),sqsvect(:),ooo(:,:),ppo(:,:), !,sqgovlp(:,:),sqgovlpi(:,:)
00176     integer::lwork,info,imin,ifzxq
00177     complex(8)::x0mx
00178     complex(8),allocatable:: uu0(:,:),vt0(:,:)
00179
00180     logical :: chipm=.false.,nolfc=.false. !sergeyv only ngczero=.false.,
00181     & ,epsmode=.false.,normalm=.false., eigr=.false.
00182     integer:: ife, idum4 !ifchipmn,ifchipm,
00183     real(8):: qs,qt,ww,muu, ddq(3)
00184     character*11 ::ttt
00185     integer:: nmxx,nomx
00186

```

```

00187 ! Feb2006 time-reversal=off case
00188     logical :: timereversal, testtimer, onceww
00189     integer :: jpm, ncc
00190     real(8) :: frr
00191
00192     integer :: ipm, nrecoff
00193
00194     real(8), allocatable :: ebb(:)
00195     logical :: evaltest !for a debug test
00196     character*300 :: aline
00197     integer :: istat, nmbas, imb, imbl, imb2, nmbas_in
00198     integer, allocatable :: imbas(:), imbas_s(:), iibas(:)
00199
00200     complex(8), allocatable :: am1(:), am2(:), mmat(:,:),
00201     & x0mat(:,:), x0matinv(:,:), eigrm(:)
00202     integer :: ifchipm_mat, ifchipm_fmat !, ifchipm_mat
00203     integer :: ifstoner, ifx, il
00204     real(8) :: istoner, zz1, zz2, zz3, zz4, istoner0, jzero2, dumml, dummm2
00205     complex(8) :: trr, trr0, trr1, zzzx(4,4), zzzz(4,4), trrx, mmatx(4,4), denom(4,4)
00206     real(8), allocatable :: eee(:), mnmnorm(:),
00207     & asvec(:,:), ssv(:,:), sproj(:,:), sprojx(:,:), momsie(:)
00208     real(8) :: eex(4), eey(4), qvv(3)
00209
00210 c     logical :: newaniso, newaniso2, newanisox !, zloffd
00211     integer :: ngb0, ifvcoud, idummy, ifepstin, igb1, igb2, ngb_in, nmbas1, nmbas2, iq0, ifisk, iqx, ig, nmbaslx,
00212     ifiss, iq0x
00213     complex(8), allocatable :: zcousq(:,:), epstin(:,:), epstilde(:,:), zcousqrsum(:,:), zcousqr(:,:)
00214     real(8), allocatable :: vcousq(:)
00215     real(8) :: fourpi, sqfourpi, tpioa, absq, vcoul, vcoulsq
00216
00217 c     !! Eq.(40) in PRB81 125102
00218 c     complex(8), allocatable :: sk(:,:), sks(:,:), ski(:,:), sksi(:,:),
00219     & w_k(:,:), w_ks(:,:), w_ki(:,:), w_ksi(:,:), llw(:,:), llwi(:,:),
00220     complex(8), allocatable :: sks(:), ski(:), sksi(:),
00221     & w_k(:), w_ks(:), w_ki(:), w_ksi(:), s_vc(:), vw_k(:), vw_ks(:)
00222     complex(8), allocatable :: llw(:,:), llwi(:,:), aaamat(:,:)
00223     integer :: lxxklm, nlxxklm, ifrcwx, iq0xx, ircw, nini, nend, iwxx, nw_ixxx, nwxxx, niwxxx, iwx, iccl, icc2
00224     complex(8) :: vclvc2
00225     integer, allocatable :: neibz(:), nwgwt(:), ngrpt(:), igx(:,:), igxt(:,:), eibzsym(:,:)
00226
00227     real(8), allocatable :: aik(:,:),
00228     integer, allocatable :: aiktimer(:)
00229     integer :: l2nl
00230     logical :: eibz4x0, tiini, iprintx, symmetrize, eibzmode
00231     real(8) :: qread(3), imagweight
00232
00233     character(128) :: vcoudfile
00234     integer :: src, dest
00235     logical :: crpa, lqall
00236     integer, allocatable :: iclasst(:), invgx(:)
00237     integer :: ificlass, ifile_handle, k
00238     complex(8), allocatable :: ppovl_(:,:)
00239
00240     logical :: readw0w0itest=.false.
00241
00242     real(8) :: ebmxx
00243     integer :: nbmxx, mtet(3)
00244     real(8), allocatable :: ekxx1(:,:), ekxx2(:,:)
00245
00246     !! -----
00247     call mpi_initialize()
00248     call mpi_consoleout('hx0fp0')
00249     call cputid(0)
00250     hartree = 2d0*rydberg()
00251     pi = 4d0*atan(1d0)
00252     fourpi = 4d0*pi
00253     sqfourpi = sqrt(fourpi)
00254 c     !! computational mode select
00255     c     takao keeps only the sergey mode.
00256     write(6, "(a)") '--- Type numbers #1 #2 #3 [#2 and #3 are options] ---'
00257     write(6, "(a)") ' #1:run mode'
00258     write(6, "(a)") ' 11 : normal Sergey'
00259     write(6, "(a)") ' 202 : epsNoLFC Sergey'
00260     write(6, "(a)") ' 203 : eps Sergey'
00261     write(6, "(a)") ' 222 : chi^+- NoLFC Sergey'
00262     write(6, "(a)") ' 223 : chi^+- Sergey'
00263     write(6, "(a)") ' 12 : total energy Miyake Sergey'
00264     write(6, "(a)") ' -9999: just show version num'
00265 c     write(6, "(a)") ' #2=ixxini #3=ixxend'
00266 c     write(6, "(a)") ' 10222 : <e^{iqr}|chi^+-|e^{iqr}> nolfc'
00267     write(6, "(a)") '-----'
00268     if (MPI__root) then
00269         read(5, *) icx
00270     endif
00271     call MPI__Broadcast(icx)
00272     call headver('hx0fp0', icx)
00273     call cputid(0)

```

```

00273      crpa=.false.
00274 .or..or.      if(ixc<=6ixc==22ixc==23) then
00275         write(6,*) 'these modes are removed now'
00276         call rx( 'these modes are not supported')
00277 ! Sergey (Hilbert-transformation) modes
00278         elseif(ixc==11) then; write(6,*) " OK ixc=11 normal mode "
00279         normalm=.true.
00280         elseif(ixc==111) then; write(6,*) " OK ixc=111 normal mode. fullband"
00281         normalm=.true.
00282         elseif(ixc==10011) then; write(6,*) " OK ixc=10011 crpa mode "
00283         normalm=.true.
00284         crpa=.true.
00285 !      -- eps mode NoLFC
00286         elseif(ixc==202) then
00287             write(6,*) " OK ixc=202 sergey's eps mode only nolfc "
00288             realonly=.true.
00289 c         iepsmode=202
00290             omitqbz=.true.
00291 !      -- eps mode with LFC
00292         elseif(ixc==203) then
00293             write(6,*) " ok ixc=203 sergey's eps mode with LFC "
00294             realonly=.true.
00295 c         iepsmode=203
00296             omitqbz=.true.
00297 ! Total energy modes
00298         elseif(ixc==12) then
00299             write(6,*) " ixc=12 Miyake's total energy sergey--->need to fix this mode"
00300             call rx( " ixc=12 miyake's total energy Sergey--->need to fix this mode")
00301             imagonly=.true.
00302             ecorr_on=901
00303 !      -- chipm mode NoLFC
00304         elseif(ixc==222) then
00305             write(6,*) " OK ixc=222 chipm sergey's "
00306             realonly=.true.
00307             omitqbz=.true.
00308             eiqr =.false. ! .true. aug2012
00309 !      -- chipm mode NoLFC
00310 c         elseif(ixc==10222) then
00311             write(6,*) " ok ixc=10222 <q|chipm_0|q> sergey"
00312 c             sergeyv=.true.
00313 c             realonly=.true.
00314 c             omitqbz=.true.
00315 c             eiqr =.true.
00316 !      -- eps mode with LFC
00317         elseif(ixc==223) then
00318             write(6,*) " ixc=223 chipm with lfc sergey's -->commented out not. need to fix this mode if
necessary."
00319             call rx( " ixc=223 chipm with LFC sergey's -->commented out not. need to fix this mode if
necessary.")
00320             realonly=.true.
00321             omitqbz=.true.
00322             eiqr =.true.
00323         else
00324             call rx( ' hx0fp0: mode ixc is not appropriate')
00325         endif
00326
00327 .or..or..or.      if(ixc==202ixc==203ixc==222ixc==223) then
00328         epsmode = .true.
00329 .or.      if(mod(ixc,200)==22mod(ixc,200)==23) chipm =.true.
00330         if(mod(ixc,10)==2) nolfc=.true.
00331     endif
00332
00333 C ... files for RPA correlation energy mode.
00334 if(ecorr_on > 0) then
00335     ieclog = 8155
00336     if(ecorr_on==901) then
00337         ieceig=8156
00338         open(ieceig,file='rpa_eigen.chk')
00339         close(ieceig,status='delete')
00340     endif
00341     open(ieclog, file='ecorr.chk')
00342 endif
00343 !! ====newaniso2====
00344 c$$$      newaniso2=.false.
00345 c$$$      if(newaniso()) then
00346 c$$$          newaniso2=.true.
00347 c$$$      endif
00348
00349 !! naraga says this cause a stop in ifort --->why???
00350 c      write(6,*) 'Timereversal=',Timereversal()
00351
00352 !! Readin BZDATA. See m_read_bzdata in gwsrsrc/rwbzdata.f
00353      call read_BZDATA()
00354
00355 !! read bzdata; See use m_read_bzdata,only:
00356 !! Use off-regular mesh for qbzreg( )=F See hx0fp0.m.sc.F also.
00357 !! This must be consistent with qg4gw.F-mkqg.F

```

```

00358 .not.      if(qb2reg()) then
00359      deltaq= qbas(:,1)/n1 + qbas(:,2)/n2 +qbas(:,3)/n3
00360      do i=1,nqbz
00361          qb2(:,i) = qb2(:,i) - deltaq/2d0
00362          write(6, "('i qb2=',i3,f8.4)") i,qb2(:,i)
00363      enddo
00364      endif
00365      write(6, "(' nqbz nqibz ngrp=',i3)") nqbz,nqibz,ngrp
00366
00367 C --- Use regular mesh even for bzcase==2 and qb2reg()==T
00368 ! A little confusing...
00369 c      ddq = 0d0
00370 c      if(bzcase()==2) ddq= dq_
00371 c      do iq = 1, nqbz
00372 c          qb2(1:3,iq) = qb2(1:3,iq) + ddq
00373 c          ! This new qb2 is regular mesh, which are identical in the both bzcase.
00374 c      enddo
00375 .not.c      if(qb2reg()) then ! off-regular mesh case
00376 c          do i=1,nqbz
00377 c              qb2(:,i) = qb2(:,i) - dq_
00378 c          enddo
00379 c      endif
00380      if(MPI__root) then
00381      do i=1,nqbz
00382      .or.          if(i<10i>nqbz-10) write(6, "('i qb2=',i8,f8.4)") i,qb2(:,i)
00383      .and.         if(i==10nqbz>18) write(6, "('... ')")
00384      enddo
00385      write(6,*)' nqbz nqibz =',nqbz,nqibz
00386      endif
00387
00388 c$$$!!- oct2005 not implimented cases.
00389 .and.c$$$      if(smbasis())chipm) then
00390 c$$$          write(6,*)' smbasis=T & chipm=T is not implimented yet.'//
00391 c$$$          &      ' Supply consistent MixSpin for smbasis! '//
00392 c$$$          &      ' MixSpin should be converted at the end of hvccfp0.'
00393 c$$$          call rx( ' smbasis=T & chipm=T is not implimented yet.')
00394 c$$$          endif
00395
00396 c      call getkeyvalue("gwinput","scaledgapx0",sciss,default=1d0)
00397 c      write(6, "(' ScaledGapX0=',f5.3)") sciss
00398
00399 !! === Readin by genallcf ===
00400 !! See "use m_genallcf_v3" at the begining of this routine
00401 !! We set basic data.
00402
00403 c$$$      if(epsmode) then
00404 c$$$          nwin = -999
00405 c$$$          else
00406 c$$$          nwin = 0      !Readin nw from NW file
00407 c$$$          endif
00408      incwfin= 0 !use ForX0 for core in GWIN
00409 c      efin = 0d0 !readin EFERMI
00410 cc--- EFERMI
00411 c      ifief=ifile_handle()
00412 c      open(ifief,file='EFERMI')
00413 c      read(ifief,*) ef
00414 c      close(ifief)
00415      call readfermi()
00416      write(6, "(a,f12.6)")' --- READIN ef from EFERMI. ef=',ef
00417      call genallcf_v3(incwfin) !in module m_genallcf_v3
00418      if(ngrp/= ngrp2) call rx( 'ngrp inconsistent: BZDATA and LMTO GWIN_V2')
00419      tpioa=2d0*pi/alat
00420
00421 .and.      if(chipmnspin==1) call rx( 'chipm mode is for nspin=2')
00422      debug=.false.; if(verbose())>=100) debug=.true.
00423      if(debug) write(6,*)' end of genallc'
00424 c      write(6, "(' ncore=',i4)") ncore
00425 c      write(6,*) 'nw_input delta=',nw_input,delta
00426
00427 !!!! WE ASSUME iclass(iatom)= iatom !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
00428 !!!! We assume nclass = natom. !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
00429      if(nclass /= natom) call rx( ' nclass /= natom ')
00430
00431 !! --- tetra or not
00432      if(delta <= 0d0) then
00433 c          tetra = .true.
00434 c          delta = -delta
00435 c          write(6,*)' hx0fp0: tetrahedron mode delta=',delta
00436 c          else
00437 c          tetra = .false. ! switch for tetrahedron method for dielectric functions
00438 c          call rx(' hx0fp0: only tetra=T support')
00439 c          endif
00440
00441 !! --- read dimensions of h,hb
00442      ifhbe = iopen('hbe.d',1,0,0)
00443      read (ifhbe,*) nprec,mrecb,mrece,nlmtot,nqbzt,nband,mrecg !warn nband is in m_zmel
00444      is = iclose('hbe.d')

```

```

00445         if(nlmtot/=nlmtot) call rx('hx0fp0: nlmtot/=nlmtot in hbe.d')
00446 c         if(nqbz /=nqbzt ) call rx('hx0fp0: nqbz /=nqbzt in hbe.d')
00447
00448 !! --- Readin Offset Gamma -----
00449         if(debug) write(6,*) 'reading QOP'
00450         open (l0l,file='QOP')
00451         read (l0l,"(i5)") nq0i
00452         write(6,*) ' ### nqibz nq0i=', nqibz,nq0i
00453         allocate( wqt(1:nq0i),q0i(1:3,1:nq0i) )
00454         do i=1,nq0i
00455             read (l0l, * ) wqt(i),q0i(1:3,i)
00456         enddo
00457         nq0ix = nq0i
00458         do i=1,nq0i
00459             if(wqt(i)==0d0 ) then
00460                 nq0ix = i-1
00461                 exit
00462             endif
00463         enddo
00464         neps = nq0i - nq0ix ! number of zero weight q0p which are used for ixc=2 or 3 mode.
00465         write( 6,*) ' num of zero weight q0p=',neps
00466         write(6,"(i3,f14.6,2x, 3f14.6)" )(i, wqt(i),q0i(1:3,i),i=1,nq0i)
00467         close(l0l)
00468 .not.c$$$         if(newaniso2) then
00469 c$$$             wqtsum = sum(abs(wqt(1:nq0i)))
00470 c$$$             call getkeyvalue("gwinput","testnoq0p",noq0p,default=.false.)
00471 .and..not.c$$$         if(normalmabs(wqtsum-ld0) >ld-10(noq0p))
00472 c$$$             & call rx( ' wqtsum of QOP /=1 ' )
00473 c$$$             endif
00474
00475 C --- readin by rdpp ; Radial integrals ppbrd and plane wave part
00476         call getsrdpp2( nclass,nl,nxx)
00477         call readngmx('QGpsi',ngpmx)
00478         call readngmx('QGcou',ngcmx)
00479         write(6,*)' ngcmx ngpmx=',ngcmx,ngpmx
00480 ! qibze(3,nqibze) qbze(3,nqibze)
00481         nqibze = nqibz *(1 + nq0i)
00482         nqibze = nqibz + nq0i
00483         allocate( qbze(3, nqibze), qibze(3, nqibze))
00484         qbze(:,1:nqibz) = qibz(:,1:nqibz)
00485         qibze(:,1:nqibz) = qibz(:,1:nqibz)
00486         do i = 1,nq0i
00487             qibze(:,nqibz+i) = q0i(:,i)
00488             ini = nqibz*(1 + i -1)
00489             do ix=1,nqibz
00490                 qbze(:,ini+ix) = q0i(:,i) + qbze(:,ix)
00491                 if( abs(qbze(1,ini+ix)+0.1d0)+abs(qbze(2,ini+ix)+0.1d0)<1d-6 ) then
00492                     write(6,"('aaaaaa qbze=',i8,3f18.14,2x,3f14.10)") ini+ix,qbze(:,ini+ix),q0i(:,i)
00493                 endif
00494             enddo
00495         enddo
00496         ngrpx = 1
00497         l2nl=2*(nl-1)
00498         allocate(symope(3,3))
00499         symope(1:3,1) = (/1d0,0d0,0d0/)
00500         symope(1:3,2) = (/0d0,1d0,0d0/)
00501         symope(1:3,3) = (/0d0,0d0,1d0/)
00502         ificlass=ifile_handle()
00503         open (ificlass,file='CLASS')
00504         allocate(iclass(natom),invvx(ngrp))
00505         & ,miat(natom,ngrp),tiat(3,natom,ngrp),shtvg(3,ngrp))
00506         write(6,*)' --- Readingin CLASS info ---'
00507         do ibas = 1,natom
00508             read(ificlass,*) ibasx, iclasst(ibas)
00509             write(6, "(2i10)") ibasx, iclasst(ibas)
00510         enddo
00511         close(ificlass)
00512 !! Get space-group transformation information. See header of mptauof.
00513         call mptauof(symope,ngrp,plat,natom,pos,iclassst
00514 o ,miat,tiat,invvx,shtvg ) !note: miat,tiat,shtvg are defined in m_zmel.
00515         if(verbose())>=40) write (*,*)' hsf0.sc.m.F: end of mptauof'
00516 !! ppbrd = radial integrals, cgrr = rotated cg coefficients.
00517         call rdpp(nxx, nl, ngrpx, nn, nclass, nspin, symope,qbas)
00518         ntq = nbnd
00519         allocate(itq(ntq)) !itq=i for i=1,ntq. a dummy. c.f. hsf0.sc.F
00520         do i=1,ntq
00521             itq(i)=i
00522         enddo
00523 !! Pointer to optimal product basis
00524 c         allocate(imdim(natom))
00525 c         call indxmnm (nblocha,nclass,iclass,natom,
00526 c o imdim ) !use in m_zmel
00527 .not.c         if(smbasis()) nblochpmx = nbloch + ngcmx
00528         nblochpmx = nbloch + ngcmx
00529         allocate(ngveccB(3,ngcmx)) ! work array
00530
00531 !! ... for legas test (not used so often. To compare homogeneous electron gas).

```

```

00532 c      legas = .false.
00533 c      INQUIRE (FILE = 'LEGAS', EXIST = legas)
00534 !!
00535      iqxend = nqibz + nq0i
00536      write(6,*) ' nqibz nqibze=',nqibz,nqibze
00537
00538 !! Initialization of readEigen !readin m_hamindex
00539 cccccccccccccccccccccccccccccccccccc
00540      ginv=transpose(plat)
00541 cccccccccccccccccccccccccccccccccccc
00542      call init_readeigen(ginv,nspin,nband,mrece)!EVU EVD are read in init_readeigen
00543      call init_readeigen2(mrech,nlmt0,mrecg)
00544      if(verbose()>50) print *, 'eeee exit of init_readeigen2'
00545 !! We get frhis,freq_r,freq_i, nwhis,nw,npm,wiw by getfreq
00546      call findemaxmin(nband,qbze,nqbze,nspin, emax,emin)
00547      if (nctot > 0) Emin=minval(ecore(:,1:nspin))
00548      omg2max = (Emax-Emin)*.5d0+.2d0
00549      ! (in Hartree) covers all relevant omega, +.2 for margin
00550      if(MPI__root) write(6,(' emin emax omega2max=',3f13.5)) emin, emax, omg2max
00551      realomega = .true.
00552      imagomega = .true.
00553      tetra = .true.
00554      if(imagonly) then !WVI only for imagonly for ixc==12
00555          realomega = .false.
00556          imagomega = .true.
00557      endif
00558      if(realonly) then !epsPP noLFC mode for ixc==13
00559          realomega = .true.
00560          imagomega = .false.
00561      endif
00562 !! getfreq return date given at " use m_freq,only:".
00563      lqall=.false.
00564 .not.      if(ixc==11) then
00565          lqall=.true.
00566      endif
00567 .not.      if(epsmode) call getwemax(lqall,wemax) !wemax is to determine nw !real axis divisions
00568      call getfreq(epsmode,realomega,imagomega,tetra,omg2max,wemax,niw,ua,MPI__root)
00569      if(MPI__root) write(6,(' nw=',i5)) nw
00570      nwp = nw+1
00571 .not.      if(imagomega) niw=1
00572 !! ... get eigenvector corresponds to exp(iqr) (q is almost zero).
00573      if(epsmode) then !iepsmode/=0) then ; write(6,*) ' read in Mix0vec'
00574          allocate(eps(nw_i:nw,neps)) !5July2005 nwp should be used after it is defined!
00575      endif
00576
00577 !! Miyake tote mode Nov2004. Need fixing.
00578      if(ecorr_on>0) then !it was bzcage()==2 Was it bug?
00579          allocate(nstibz(nqibz))
00580          do iq=1,nqibz
00581              iqbz = iqindx(qibz(:,iq),ginv,qbz,nqbz)
00582              nstibz(iq) = nstibz(iqbz)
00583 c      write(6,(' iq qibz nstibz=',i5,3f9.4,i5))iq,qibz(:,iq),nstibz(iq)
00584          enddo
00585      endif
00586
00587 !! tetra init
00588 c      call getkeyvalue("gwinput","tmpwwk",tmpwwk,default=.false.)
00589 c      if(tetra) then
00590 c          allocate( !wgt(nband+nctot,nband,nqbz), !noccxv(nw+niw),
00591 c      &      nbnbtt(nqbz,npm), ekxx1(nband,nqbz), ekxx2(nband,nqbz)) !!! nband=nlmt0
00592 c      if(tmpwwk) ifwwk = iopen('tmp.wwk',0,-1,0)
00593 c      endif ; if(debug) write(6,*)' xxx1:'
00594      noccxv = maxocc2 (nspin,ef, nband, qbze,nqbze) ! maximum no. occupied valence states
00595      if(noccxv>nband) call rx( 'hx0fp0: all the bands filled! too large Ef')
00596      noccx = noccxv + nctot
00597
00598 C allocate( ppb(nlnmx*nlnmx*mdimx*nclass,nspin) )
00599 c$$$C ... This is just to get nblochpmx
00600 c$$$      if(smbasis()) then
00601 c$$$          call getngbpomat(nqibz+nq0i, nnmx,nomx)
00602 c$$$          nblochpmx = nnmx
00603 c$$$      endif
00604
00605      nprecx = ndble !We use double precision arrays only.
00606      mrecl = nprecx*2*nblochpmx*nblochpmx/nword()
00607      if (MPI__root) then
00608          ifwd = iopen('WV.d',1,-1,0)
00609          write (ifwd,('lx,l0il4')) nprecx,mrecl,nblochpmx,nwp,niw,nqibz + nq0i-1,nw_i
00610          ifwd = iclose('WV.d'); ifwd=0
00611      endif
00612      allocate( zw(nblochpmx,nblochpmx) )
00613      nspinmx = nspin
00614
00615 !!... these are used x0k
00616      call getkeyvalue("gwinput","nbcutlow",nbcut, default=0 )
00617      call getkeyvalue("gwinput","nbcutlowto",nbcut2, default=0 )
00618      write(6,(' nbcut nbcutlowto=',2i5)) nbcut,nbcut2

```



```

00619
00620 !! -- ppb= <Phi(SLn,r) Phi(SL'n',r) B(S,i,R(r))>
00621 !! This is general for rotated CG coefficient; but hx0fp0 mode is only for ngrpx=1 (not rotated).
00622 !! Compare usage in hsfp0 modes.
00623 irot=1
00624 allocate( ppbir(nlnmx*nlnmx*mdimx*nclass,irot,nspin))
00625 do is = 1,nspin
00626     call ppbafp_v2 (irot,ngprx,is,nspin,
00627         i   il,in,im,nlnm,
00628         i   nl,nn,nclass,nlnmx,
00629         i   mdimx,lx,nx,nxx,      !Bloch wave
00630         i   cgr, nl-1,            !rotated CG
00631         i   ppbrd,                !radial integrals
00632         o   ppbir(:,irot,is))      ! this is in m_zmel, used to generate <phi|phi B>
00633     enddo
00634     if(debug)write(6,*) ' end of ppbafp_v2'
00635
00636 !! Set iqxini
00637     if(omitqbz) then
00638         iqxini= nqibz + 1
00639     else
00640         iqxini= 1
00641     endif
00642
00643 !! check write 1st part for Ec mode to ecorr.chk Nov2004
00644     if(ecorr_on>0) then
00645         !!!!!!!!!!!!!!! this path is under developing. !!!!!!!!!!!!!!!
00646         call rx(' ! hx0fp0: need to fix this path. check subroutine getwk and so on in this path')
00647         !!!!!!!!!!!!!!!
00648         .and..not.      if(ecorr_on >0 (imagomega) )
00649         .and..not.      & call rx(' hx0fp0: ecorr_on (imagomega)')
00650         write(ieclog, "(' iq          q          wk')")
00651         do iqq = iqxini,ixend
00652             call getwk(iqq, wibz, wqt,nqbz,nqibz,nstibz,nq0i, wk4ec)
00653             write(ieclog,"(i5,3x,3f12.8, f15.5)") iqq, q, wk4ec
00654         enddo
00655         write(ieclog,*)
00656         write(ieclog,"(' iw omega(Ry)      wiw')")
00657         do iww=1,niw
00658             write(ieclog,"(i5, f10.5, f10.5)") iww,2d0*freq_i(iww),wiw(iww)
00659         enddo
00660         write(ieclog,*)
00661         write(ieclog,"(' Note:IntWgt=wk*wiw.',
00662         & ' Ec =\sum_{k,iw} IntWgt(k,iw)*ecqw(k,iw)')")
00663         close(ieclog)
00664         open(ieclog,file="ecorr.chk",access='append')
00665
00666         call getkeyvalue("gwinput","necut_p",necut, default=1 )
00667         allocate(totexc(necut),trpv(necut),trlog(necut))
00668         totexc = 0d0
00669         trpv   = 0d0
00670         trlog  = 0d0
00671     else
00672         necut=1
00673     endif
00674 !!
00675     allocate(ecut(necut),ecuts(necut))
00676     call getkeyvalue("gwinput","ecut_p",ecut, necut,default=(/ld10/) )
00677     call getkeyvalue("gwinput","ecuts_p",ecuts,necut,default=(/ld10/) )
00678 !!
00679     if( chipm ) then
00680         nmbas=natom
00681         allocate(imbas(nmbas),imbas_s(nmbas))
00682         istat=-9999 ! istat=-9999 means nnumber of readin arguments is returened in istat.
00683         call getkeyvalue("gwinput","magatom",
00684         &         imbas,nmbas,status=istat)
00685         nmbas = istat
00686         write(6,*)
00687         write(6,"('Readin MagAtom nmbas =',i3,' imbas= ',l0i3)") nmbas,imbas(1:nmbas)
00688         imbas_s(1:nmbas) = imbas(1:nmbas)
00689         imbas(1:nmbas) = abs(imbas(1:nmbas))
00690         allocate(jcoup(nw_i:nw,neps) )
00691         allocate( svec(nbloch,nmbas) ) !sep2006
00692         svec=0d0
00693         allocate( cvec(nbloch,nmbas),momsite(nmbas),
00694         &         mnmnorm(nmbas))          !May2007
00695         cvec=0d0
00696         do imb=1,nmbas
00697             ibas= imbas(imb)
00698             ifv = iopen ('MixSpin.'//charnum3(ibas),1,3,0)
00699             read(ifv,*) ibasx,lxx
00700             allocate(nxx_r(0:lxx))
00701             do i=0,lxx
00702                 read(ifv,*) nxx_r(i) ! write(6,"(2i5,d13.6)") nxx_r(i)
00703             enddo
00704             allocate(spinvec((lxx+1)**2,maxval(nxx_r)))
00705             allocate(consvec((lxx+1)**2,maxval(nxx_r)))

```



```

00706         spinvec=0d0
00707         do ilm = 1, (lxx+1)**2
00708             lb = 1l(ilm) ) ! write(6,*)' lb=',lb,lxx,ilmx
00709             do ix = 1, nxx_r(lb) ! write(6,*)' nn=',nn,nxx_r(lb)
00710                 if(ilm==1) then
00711                     read(ifv,*) ilm_r, nx_r, spinvec(ilm,ix),chgl,chg2
00712             &         ,consvec(ilm,ix)
00713                 else
00714                     read(ifv,*) ilm_r, nx_r, spinvec(ilm,ix),dumm1,dumm2
00715             &         ,consvec(ilm,ix)
00716                 endif
00717 !             write(6,"(2i5,d13.6)") ilm, ix, spinvec(ilm,ix)
00718             enddo
00719         enddo
00720 !! Calculate ChipM. So sign of omega should be correct.
00721         if(imb=1) then !determine spin direction with respect to ibas=imbas(imb=1)
00722             spinmom=(chgl-chg2)
00723             schi=ld0
00724             if(spinmom<0d0) then
00725                 schi = -ld0 ! This affects to dpsion. Obtained results
00726                             ! should be the same in both mode.
00727             endif
00728         endif
00729 !! ReOrdering of spinvec in natom ordering...
00730         i=0
00731         if(ibas>1) i= sum(nblocha(1:ibas-1))
00732         do lb = 0, lx (ibas)
00733             do nb = 1, nx (lb,ibas)
00734                 do mb = -lb, lb
00735                     i = i+1
00736                     ilm = lb**2+ lb+ mb +1
00737                     svec(i,imb) = spinvec(ilm,nb)
00738                     cvec(i,imb) = consvec(ilm,nb)
00739                     write(6,"(' i lb mb svec svec**2=',3i4,2d13.5)")
00740             &         i,lb,mb,svec(i,imb),svec(i,imb)**2
00741                 enddo
00742             enddo
00743         enddo
00744         deallocate(nxx_r,spinvec,consvec)
00745         close(ifv)
00746         mnmorm(imb) = sqrt(sum(svec(:,imb)**2))
00747         momsie(imb) = chgl-chg2
00748 c         write(6,"(' svecsum=',e23.15)") sum(svec(:,imb)**2)
00749 c         write(ifchipmlog,"(2e23.15,' ! mmom mnmorm')")momsie(imb),mnmorm(imb)
00750         write(6,"( 'mmom mnmorm= ',2f14.10)") mnmorm(imb),momsie(imb)
00751     enddo
00752 endif
00753
00754 ! I assume 1 is for majority for eigr case.
00755 c         if(ix==10222) then
00756 cc             schi=ld0 !ld0 means Majority is isp=1. If Majority is isp=2, use schi=-ld0.
00757 cc             allocate(jcoup(nw_i:nw,neps))
00758 c             mnmorm=ld0
00759 c         endif
00760 c
00761 c! nmbas_in is for rcxqmean
00762 .and..and.c         if(chipm nolfco) then ! ix/=10222) then
00763 c             nmbas_in = nmbas
00764 c         else
00765 c             nmbas_in = 1
00766 c         endif
00767 .and.c         if(epsmodenolfco) then
00768 c             allocate( rcxqmean(nwhis,npm,nmbas_in,nmbas_in))
00769 c             if(debug) write(6,"('fff:',3i5)") nwhis,npm,nmbas_in
00770 c         else
00771 c             allocate( rcxqmean(1,1,1,1)) !dummy
00772 c         endif
00773 c
00774 c         if(chipm) allocate(eigrm(nmbas))
00775 c
00776 cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00777 cctakao
00778 c$$$         allocate( x0meanx(nmbas,nmbas) )
00779 c$$$         allocate( x0mat(nmbas,nmbas),x0matinv(nmbas,nmbas) )
00780 c$$$         do 1101 iq = ixini,ixend ! q=(0,0,0) is omitted!
00781 c$$$             if(iq==ixini+2) exit
00782 c$$$             q = qibze(:,iq)
00783 c$$$             write(6,*)'aaaaaaaaaa q=',q
00784 c$$$             read(ifgb0vec,*) qgbin(1:3),ngbin,igc0,dnorm
00785 c$$$             if(sum(abs(q))=0d0)then
00786 c$$$                 if(sum(qgbin**2) >ld-7)stop'qgbin=0 xxx See hvccfp0'
00787 c$$$                 elseif(sum(abs(qgbin(1:3)-q)) >ld-8)then
00788 c$$$                     stop'qgbin inconsistent'
00789 c$$$             endif
00790 c$$$             write(6,"(' --- Readin Mix0vec: ',3d13.6,2i5,d18.8)")
00791 c$$$             &         qgbin(1:3),ngbin,igc0,dnorm
00792 c$$$c             if(ngb/=ngbin) stop 'hx0fp0: ngb/=ngbgin'

```

```

00793 c$$$      ngb=ngbi
00794 c$$$      write(6,(' ngb nwp niw=',3i8))ngb,nwp,niw
00795 c$$$      nmbas_in=1
00796 c$$$      allocate( gbvec(ngb),zxr(ngb,1),x0mean(nw_i:nw,1,1))
00797 c$$$      x0mean=0d0
00798 c$$$      do i=1,ngb
00799 c$$$          read(ifgb0vec,"(4d24.15)") zxl,zz2,zz3,zz4
00800 c$$$          gbvec(i)= dcmplx(zxl,zz2)
00801 c$$$          zxr(i,1)= dcmplx(zz3,zz4)
00802 c$$$      enddo
00803 c$$$      write(6,(' normchk=',255e23.15)) sum( dconjg(gbvec)*zxr(:,1) )
00804 c$$$      & ,sum(abs(gbvec(:))), sum(abs(zxr(:,1)))
00805 c$$$      allocate(eiqrm(nmbas))
00806 c$$$      do imb=1,nmbas
00807 c$$$          eqirm(imb)= sum( dconjg(gbvec(1:nblock))*svect(1:nblock,imb) )
00808 c$$$          write(6,(' <eqir|m> ',255e23.15)) eqirm(imb)
00809 c$$$          if( imbas_s(imb)<-1) eqirm(imb)= -eqirm(imb)
00810 c$$$      enddo
00811 c$$$      write(6,('<eqir|m>:Set \pm in GWinput(for stuggard chi)'))
00812 c$$$      iqixc2 = iq- (nqibz+nq0ix)
00813 c$$$      ifx = iopen ('StonerNLFC.dat',1,3,0)
00814 c$$$      read(ifx,*) jzero2
00815 c$$$      ifx= iclose('StonerNLFC.dat')
00816 c$$$
00817 c$$$      ifchipm2=iopen(
00818 c$$$      &         'ChiPM'//charnum4(iqixc2)//'.nolfc.mat',1,3,0)
00819 c$$$      do iw=1,10
00820 c$$$          read(ifchipm2,
00821 c$$$      &         '(36x,2x,20x,2x,255e23.15)') x0meanx(:,:)
00822 c$$$          write(6,('xxx x0mat=",255dl3.5)') x0meanx
00823 c$$$          x0matinv=x0meanx
00824 c$$$          call matcinv(nmbas,x0matinv)
00825 c$$$          do i=1,nmbas
00826 c$$$              x0matinv(i,i)= x0matinv(i,i) - jzero2 ! (chipm_0^+-)^-1 - I
00827 c$$$          enddo
00828 c$$$          x0mat = x0matinv
00829 c$$$          do i=1,nmbas
00830 c$$$              x0mat(i,i) = x0mat(i,i)+ img*ld-30 ! to avoid inversion error.
00831 c$$$          enddo
00832 c$$$
00833 c$$$          call matcinv(nmbas,x0mat) !this is full x0_+
00834 c$$$          trr = sum( eqirm*matmul(x0mat,dconjg(eqirm)) ) !mmnorm
00835 c$$$          write(6,
00836 c$$$      &         '"ttt",3f12.8,2x,f10.5,2x,2e23.15,2x,2e23.15)') q, 2*schi*frr, trr,ld0/trr
00837 c$$$          enddo
00838 c$$$          ifx=iclose(ifchipm2)
00839 c$$$      1101 continue
00840 c$$$      stop 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'
00841 cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00842
00843 !! nov2016 moved from tetwt5 --> here
00844 call getkeyvalue("gwinput","nband_chi0",nbmx, default=nband )
00845 call getkeyvalue("gwinput","emax_chi0", ebmx, default=ld10 )
00846 mtet=(/1,1,1/)
00847 call getkeyvalue("gwinput","multitet",mtet,3,default=(/1,1,1/))
00848 ! multitet=T ==> micro tetrahedron method (divided-tetrahedron). Not used so much now...
00849 allocate(ekxx1(nband,nqbz),ekxx2(nband,nqbz))
00850
00851
00852 !! -- EIBZ mode for nolfco -----
00853 eibzmode=eibz4x0()
00854 !! If eibzmode=T, it is efficient but can slightly break crystal symmetry.
00855
00856 !! This is because band connectivity is judged by just from band ordering in tetrahedron weitht tetwt5.
00857 c    if(nolfco) then
00858 c        eibzmode = .false.
00859 c    endif
00860 c!! -----
00861
00862 !! === Use of symmetry. EIBZ procedure PRB81,125102 ===
00863 !! For rotation of zcousg. See readeigen.F rotwv.F ppbafp.fal.F(for index of product basis).
00864 if(eibzmode) then
00865 !! commentout block inversion Use iqxendx=iqxend because of full inversion
00866 call cputid(0)
00867 write(6,*)' ---goto eibzmode block ---'
00868 iqxendx=iqxend
00869 if(epsmode) iqxendx=iqxend
00870 allocate( nwgt(nqbz,iqxini:iqxendx), !qeibz(3,nqbz,iqxini:nqbz),neibz(iqxini:nqbz),
00871 & igx(ngrp*2,nqbz,iqxini:iqxendx),igxt(ngrp*2,nqbz,iqxini:iqxendx),
00872 & eibzsym(ngrp,-1:1,iqxini:iqxendx))
00873 !! Check timereversal is required for symmetrization operation or not. If tiii=timereversal=F is enforced,
00874 !! the symmetrization procedure in x0kf_v4h becomes a little time-consuming.
00875 write(6,*)
00876 write(6,('=== Goto eibzgen === TimeReveal switch =',l1))timereversal()
00877 if(MPI__root) iprintx=.true.
00878 call eibzgen(nqibz,symgg,ngrp,qibze(:,iqxini:iqxend),iqxini,iqxendx,qbz,nqbz,
00879 i    timereversal(),qinv,iprintx,
```

```

00880      o  nwgt,igx,igxt,eibzsym,tiii)
00881      write(6, "('Used timeReveal for EIBZ = ',11) ") tiii
00882      call cputid(0)
00883 ! PBindex: index for product basis. We will unify this system; still similar is used in ppbafp_v2.
00884      call PBindex(natom,lx,l2nl,nx) !all input. Returns required index stored in arrays in m_pbindex.
00885      call cputid(0)
00886      call readqgcou() !no input. Read QGcou and store data into variables.
00887 c  call Spacegroupprot(symgg,ngrr,plat,natom,pos) ! all inputs.
00888      else
00889          !dummy allocation to overlaid -check bound !sep2014
00889          ixendx=ixend
00890          allocate( nwgt(1,ixini:ixendx),igx(1,1,ixini:ixendx)
00891          & ,igxt(1,1,ixini:ixendx), eibzsym(1,1,ixini:ixendx)) !dummy
00892          nwgt=1
00893      endif
00894
00895      allocate( llw(nw_i:nw,nq0i), llwI(niw,nq0i) )
00896 !! == Calculate x0(q,iw) and W == main loop 1001 for iq.
00897 !! NOTE: iq=1 (q=0,0,0) write 'EPS0inv', which is used for iq>nqibz for ixc=11 mode
00898 !! Thus it is necessary to do iq=1 in advance to performom iq >nqibz.
00899 !! (or need to modify do 1001 loop).
00900 !! iq>nqibz for ixc=11 is not time-consuming.
00901      call MPI__hx0fp0_rankdivider2(ixini,ixend)
00902
00903 !! -----
00904 !! == loop over iq =====
00905 !! -----
00906      do 1001 iq = ixini,ixend ! NOTE: q=(0,0,0) is omitted when ixini=2
00907 .not.      if( MPI__task(iq) ) cycle
00908 .or.      if(ixc==10lnormalm) then
00909          ifrcwi = iopen('WVI.'//charnum5(iq),0,-1,mrecl)
00910      endif
00911      if (normalm) then
00912          ifrcw = iopen('WVR.'//charnum5(iq),0,-1,mrecl)
00913      endif
00914      call cputid (0)
00915
00916      q = qibze(:,iq)
00917      call readqg('QGcou', q, ginv, quu,ngc,ngveccB)
00918
00919 !! Caution : confusing point
00920 !! ngc by QGcou is shown at the bottom of lqg4gw.
00921 !! ngc read from PPOVL are given by rdata4gw---> ngc(iq>nqibz )=ngc for q=0
00922 !!
00923 .and.c      if( newaniso2iq==1 ) then ! *sanity check
00924      if( iq==1 ) then ! *sanity check
00925          if(sum(q**2)>1d-10) then
00926              call rx( ' hx0fp0: sanity check. |q(iqx)| /= 0' )
00927          endif
00928      endif
00929
00930 !! ==== readin Coulomb matrix ====
00931      ngb = nbloch + ngc !ngb is readin from vcoul 25jan2006
00932      write(6,*)
00933      write(6, "('==== do 1001: iq q=',i7,3f9.4, ' =====') )iq,q !qq
00934      write(6, "(' nbloch ngb ngc=',3i10) ") nbloch,ngb,ngc
00935
00936 !! == readin diagonalized Coulomb interaction ==
00937 !! zcousq: E(\nu,I), given in PRB81,125102; vcousq: sqrt(v), as well.
00938 .and..not.c      if(newaniso2(chipm)) then
00939 .not.      if((chipm)) then
00940          vcoudfile='Vcoud.'//charnum5(iq) !this is closed at the end of do 1001. iq was iqgv
00941          ifvcoud = iopen(trim(vcoudfile),0,-1,0)
00942          read(ifvcoud) ngb0
00943          if( ngb0/=ngb ) call rx( 'hx0fp0.m.f:ngb0/=ngb' )
00944          read(ifvcoud) qvv
00945          if(sum(abs(qvv-q))>1d-10) then
00946              write(6,*) 'qvv =',qvv
00947              call rx( 'hx0fp0: qvv/=0 hvcc is not consistent' )
00948          endif
00949          if(allocated(zcousq)) deallocate( zcousq,vcousq )
00950          allocate( zcousq(ngb0,ngb0),vcousq(ngb0))
00951          read(ifvcoud) vcousq
00952          read(ifvcoud) zcousq
00953          idummy=iclose(trim(vcoudfile))
00954          vcousq=sqrt(vcousq)
00955          if(allocated(zzr)) deallocate(zzr)
00956          allocate(zzr(1,1)) !dummy
00957          zzr=0d0
00958      endif
00959
00960 .and..and.      if(chipm nolfc0) then ! ix/=10222) then
00961          nmbas_in = nmbas
00962      elseif(nolfc0) then
00963          nmbas_in = 1
00964      else
00965          nmbas_in = ngb
00966      endif

```

```

00967      nmbas1 = nmbas_in
00968      nmbas2 = nmbas1
00969
00970 !! ==== set up for epsilon mode ====
00971      if(epsmode) then
00972          iqxc2 = iq- (ngibz+ng0ix)
00973 .not..and.      if((chipm)nolfco) then
00974          allocate( x0mean(nw_i:nw,1,1) )
00975          x0mean=0d0
00976      endif
00977 .and.!! zzzr is only for chipmnolfco mode
00978 .and.      if(chipm nolfco) then
00979          allocate(zzr(ngb,nmbas),x0mean(nw_i:nw,nmbas,nmbas))
00980          x0mean=0d0
00981          zzzr =0d0
00982          zzzr(1:nbloch,1:nmbas) = svec(1:nbloch,1:nmbas)
00983      endif
00984 !! ... Open ChiPM*.nolfc_mat
00985 .and.      if( wqt(iq-ngibz)==0d0chipm ) then
00986          ifchipmn_mat=iopen('ChiPM'//charnum4(iqxc2)//'.nlfco.mat',1,3,0)
00987          write(ifchipmn_mat,"(255i5)") nmbas
00988          write(ifchipmn_mat,"(255i5)") imbas(1:nmbas)
00989          write(ifchipmn_mat,"(255e23.15)") moms(1:nmbas)
00990          write(ifchipmn_mat,"(255e23.15)") mmnorm(1:nmbas)
00991 c          write(ifchipmn_mat,"(255e23.15)") eiqr(1:nmbas)!if necessary, fix code to give eiqr.
00992      takaoAug2012 write(ifchipmn_mat,"( ' Here was eiqr: If needed, need to fix hx0fp0')")
00993 .not.      if(nolfco) then
00994          ifchipm_fmat=iopen('ChiPM'//charnum4(iqxc2)//'.fmat',0,3,0)
00995          write(ifchipm_fmat) nbloch, natom,nmbas, iqxini,iqxcend, nw_i,nw
00996          write(ifchipm_fmat) imbas(1:nmbas),moms(1:nmbas),mmnorm(1:nmbas)
00997          write(ifchipm_fmat) nblocha(1:natom),svec(1:nbloch,1:nmbas)
00998          write(ifchipm_fmat) zzzr(1:nbloch) !zzr(1:nbloch,1)
00999      endif
01000 .and..not.      elseif(wqt(iq-ngibz)==0d0(chipm)) then
01001 !! ... Open EPS* file
01002      filepsnolfc = 'EPS'//charnum4(iqxc2)//'.nlfco.dat'
01003      ifepsdatnolfc = iopen ( filepsnolfc,1,3,0)
01004      write(ifepsdatnolfc,"(a)") ' q(1:3) w(Ry) eps epsi --- NO LFC'
01005 .not.      if(nolfco) then
01006          fileps = 'EPS'//charnum4(iqxc2)//'.dat'
01007          ifepsdat = iopen ( fileps,1,3,0)
01008          write(ifepsdat,"(a)") ' q(1:3) w(Ry) eps epsi --- LFC included. '
01009      endif
01010      endif
01011      endif
01012
01013 .and.      if(epsmodenolfco) then !iepsmode==202) then
01014 c          rcxqmean=0d0
01015      else
01016          write(6,*) "rcxq alloc ngb nwhis nrm ---",ngb,nwhis,nrm
01017          allocate( rcxq(ngb,ngb,nwhis,nrm) )
01018      endif
01019
01020 !! === zzzr conversion on different basis. ppovlz is used in get_zzzr2 in m_zzzr (called in x0kf_v4h).
01021 .and.      if(chipmnolfco) then
01022          if(allocated(ppovlz)) deallocate(ppovlz)
01023          allocate(ppovlz(ngb,nmbas1))
01024          ppovlz= zzzr
01025 .and.      elseif(nolfco nmbas1==1) then !for <e^iqr|x0|e^iqr>
01026 c          if(allocated(ppovlzinv)) deallocate(ppovlzinv)
01027          if(allocated(ppovlz)) deallocate(ppovlz)
01028          if(allocated(ppovl)) deallocate(ppovl)
01029          allocate(ppovl(ngc,ngc),ppovlz(ngb,ngb)) !, ppovlzinv(ngb,ngb))
01030          call readppovl0(q,ngc,ppovl)
01031          ppovlz(1:nbloch,:) = zcousq(1:nbloch,:)
01032          ppovlz(nbloch+1:nbloch+ngc,:) = matmul(ppovl,zcousq(nbloch+1:nbloch+ngc,:))
01033          write(6,*) 'nnnnn',nbloch+ngc,ngb
01034      else
01035          !may2013 this removes 0^-1 factor from zzzr
01036 c          if(allocated(ppovlzinv)) deallocate(ppovlzinv)
01037          if(allocated(ppovlz)) deallocate(ppovlz)
01038          if(allocated(ppovl)) deallocate(ppovl)
01039          allocate(ppovl(ngc,ngc),ppovlz(ngb,ngb)) !, ppovlzinv(ngb,ngb))
01040          call readppovl0(q,ngc,ppovl)
01041          allocate(ppovl_(ngb,ngb))
01042          ppovl_=0d0
01043          do i=1,nbloch
01044              ppovl_(i,i)=1d0
01045          enddo
01046          ppovl_(nbloch+1:nbloch+ngc,nbloch+1:nbloch+ngc)=ppovl
01047 .not.      if(eibz4x0()) then !sep2014 added for eibz4x0=F
01048          ppovl_ = matmul(ppovl_,zcousq)
01049      endif
01050          ppovlz = ppovl_
01051          deallocate(ppovl_,ppovl)
01052      endif

```



```

01139      i          q,
01140      i          nspin,is,isf, !symmetrize, !
01141      i          qbas,ginv, qbz,wbz,
01142      d          nlmto,nqbz,nctot, !noccx,noccxv,
01143      d          nbloch, nwhis, !nlmnm,mdimx,
01144      i          iq,ngb,ngc,ngpmx,ngcmx, !ngb/=ngc+nbloch for smbasis()=T oct2005
01145      i          nqbze,nband,nqibz,
01146      o          rcxq, ! rcxq is the accumulating variable for spins
01147      i          nolfco,zzr,nmbas_in, zcousq, !ppovl,nmbas1,nmbas2, is removed ppovlz,
01148      i          chipm,eibzmode, !zloffd,!for nolfco Add nmbas Sep2006
01149      i          nwgt(:,iq),igx(:,iq),igxt(:,iq),ngrp, eibzsym(:,iq),crpa)
01150 !! -----Question, Apr2015takao. -----
01151 !! ??? we may need
01152 !! ??? "if(is==nspinmx.or.chipm) then" for chipm mode.
01153 !! ??? really OK ??? Need check more... Compare with old code...
01154 !! -----
01155 !kino 2014-08-19 add
01156
01157 !! == Symmetrizer for crystal symmetry (and also for spin)
01158 !! Symmetrize and convert to Enu basis by dconjg(tranpsoc(zcousq)*rcxq8zcousq if eibzmode
01159 .or.          if (is==nspinmxchipm) then ! Apr2015. Takao think ".or.chipm" is required for chipm mode
01160                  ! Because rcxq is calculated for each is, symmetrized and its
contribution
01161                  ! is added to zxq in dpsion5.
01162      call x0kf_v4hz_symmetrize(npm, !ncc,
01163      c      i          ihw,nhw,jhw,whw,nhwtot, ! tetwt5
01164      c      i          nlb,n2b,nbnbx,nbnb, ! use whw by tetwt5 ,
01165      i          q,
01166      i          nspin,is,isf, !symmetrize, !
01167      i          qbas,ginv, !qbz,wbz,
01168      c      i          nblocha, !nlm,nlnmv,nlnmc,iclass,
01169      c      i          ppb(1,is),
01170      c      i          icore,ncore,
01171      c      d          nlmto,nqbz,nctot, !noccx,noccxv,
01172      c      d          natom, !nl,nclass,natom,nnc,
01173      d          nbloch, nwhis, ! nlnmx,mdimx,
01174      i          iq,ngb,ngc,ngpmx,ngcmx, !ngb/=ngc+nbloch for smbasis()=T oct2005
01175      i          nqbze,nband,nqibz,
01176      o          rcxq, ! rcxq is the accumulating variable for spins
01177      i          nolfco,zzr,nmbas_in, zcousq, !ppovl,nmbas1,nmbas2, is removed ppovlz,
01178      i          chipm,eibzmode, !zloffd,!for nolfco Add nmbas Sep2006
01179      i          ngrp, eibzsym(:,iq))
01180      endif
01181      call tetdeallocate() !deallocate(ihw,nhw,jhw, whw,ibjb,nlb,n2b)
01182      iecut=1
01183      if(debug) write(6,"(a)") ' --- goto dpsion5 --- '
01184 .or.          if(is==nspinmxchipm) then
01185                  write(6,"(' nmbas1,nmbas2=' ,2i10)") nmbas1,nmbas2
01186                  call dpsion5(frhis,nwhis, freq_r, nw, freq_i,niw, realomega, imagomega,
01187                  i          rcxq, npm,nw_i, nmbas1,nmbas2, ! rcxq is alterd---used as work
01188                  o          zxq, zxqi,
01189                  i          chipm, schi,is, ecut(iecut),ecuts(iecut))
01190 .and.          if(nolfcoepsmode) then
01191                  do iw=nw_i,nw
01192                  x0mean(iw,:)=zxq(:,iw)
01193                  enddo
01194                  endif
01195                  write(6,*) ' --- end of dpsion5 ----',sum(abs(zxq)),sum(abs(zxqi))
01196      endif
01197 1003 continue !end of spin loop ====
01198      if(allocated(rcxq) ) deallocate(rcxq)
01199
01200 !! === RealOmega =====
01201      if (realomega) then
01202 .or.          if(chipm) then !ixc==22ixc==23) then
01203                  if (nspin==1) call rx( 'chipm modes are for nspin==2')
01204 .and..and..not..or.          elseif(epsmodenolfco(chipm)) then !ixc==2iepsmode==202) then
01205                  if (nspin==1) x0mean= 2d0*x0mean !if paramagnetic, multiply x0 by 2
01206                  if (nspin==1) zxq = 2d0*zxq !if paramagnetic, multiply x0 by 2
01207                  else
01208                  if (nspin == 1) zxq = 2d0*zxq !if paramagnetic, multiply x0 by 2
01209                  endif
01210
01211 c          write (ifxd,"(1x,3f10.4)") q(1),q(2),q(3)
01212 c          write (ifrx) rxq,cxq
01213      if(epsmode) then
01214          if(nolfco) then
01215              ttt='without LFC'
01216          else
01217              ttt='with LFC'
01218          endif
01219          if(chipm) then
01220              write(6,*) '--- chi0_{+-}}^{ -1} --- '//ttt
01221          else
01222              write(6,*) '--- dielectric constant --- '//ttt
01223              write(6,*) " trace check for w-v"
01224          endif

```

Generated on Fri Nov 11 2016 17:43:19 for ecalj/fpgw/ code document by Doxygen



Generated on Fri Nov 11 2016 17:43:19 for ecalj/fpgw/ code document by Doxygen



```

01390         endif
01391
01392 .not.c$$$          if(newaniso2) then          ! Original mode
01393 c$$$              call rx( 'not checked here')
01394 c$$$c            call wcf( ngb, vcoul, zxq(1,1,iw), imode, zw0)
01395 c$$$              endif
01396
01397 c$$$!!... a debug mode
01398 c$$$              write(6, "('hhh --- EigenValues for Im( W) -----')")
01399 c$$$              allocate(ebb(ngb))
01400 c$$$              call diagcvh2( (zw0-transpose(dconjg(zw0)))/2d0/img, ngb, ebb)
01401 c$$$              do ii=1,ngb
01402 .and.c$$$          if( abs(ebb(ii))>1d-8 .and. ebb(ii)>0) then
01403 c$$$              write(6, "('hhhIWq : iw ii eb=' ,2i4,d13.5)") iw, ii, ebb(ii)
01404 c$$$              else
01405 c$$$              write(6, "('hhhIWqxxx : iw ii eb=' ,2i4,d13.5)") iw, ii, ebb(ii)
01406 c$$$              endif
01407 c$$$              enddo
01408 c$$$              deallocate(ebb)
01409
01410 .and.c            if(newaniso2iq>ngibz) then
01411 if(iq>ngibz) then
01412 c                zw(1:ngb,1:ngb) = 0d0
01413 c                write(ifrcw, rec=((iq-ixqini)*(nw-nw_i+1)+ iw-nw_i+1 ) ) zw      ! WP = vsc-v
01414 else
01415 zw(1:ngb,1:ngb) = zw0
01416 c                write(ifrcw, rec=((iq-ixqini)*(nw-nw_i+1)+ iw-nw_i+1 ) ) zw      ! WP = vsc-v
01417 write(ifrcw, rec= iw-nw_i+1) zw      ! WP = vsc-v
01418 call tr_chkwrite("freq_r iq iw realomg trwv=", zw, iw, frr,nblochpmx, nbloch,ngb,iq)
01419 endif
01420 !! epsmode
01421 .and..not..and.   elseif(epsmode(chipm)) then !ixc/=23) then ! No LFC (local field correction).
! It's better to use echo 4| hbasfp0.
01422 if(debug)write(6,*) 'xxx2 epsmode iq,iw=',iq,iw
01423 c                write(6,*) 'ppppp sumcheck zxq=',sum(abs(zxq)),sum(abs(zzr)),sum(abs(vcoul)),sum(abs(gbvec))
01424 c                if(newaniso2) then
01425 !! there is difference of two vcmean below since we use (sligthy) screened Coulomb (screenfac() in
switch.F)
01426 !! NOTE that we use vcoul with screening (screenfac() is used in hvccfp0.F
01427 c                vcmean = fourpi/sum(q**2*tpioa**2) !aug2012
01428 vcmean=vcousq(1)**2
01429 epsi(iw,iqixc2)= 1d0/(1d0 - vcmean*zxq(1,1,iw))
01430 write(6,(' " iq iw omega eps epsi nolfc=" ,2i6,f8.3,2e23.15,3x, 2e23.15,
& " vcmean x0mean=" , 2e23.15,3x, 2e23.15') ) iqixc2,iw,2*frr,
01432 & 1d0/epsi(iw,iqixc2),epsi(iw,iqixc2),vcmean, zxq(1,1,iw)!x0mean(iw,1,1)
01433 write(ifeqsdatsnolfc,(' (3f12.8,2x,d12.4,2e23.15,2x,2e23.15)') )
01434 & q, 2*frr, 1d0/epsi(iw,iqixc2),epsi(iw,iqixc2)
01435 .not.            if(nolfc) then
01436 ix=0
01437 do igb1=ix+1,ngb
01438 do igb2=ix+1,ngb
01439 .and.            if(igb1==igb2==1) then
01440 epstilde(igb1,igb2)= -vcmean*zxq(igb1,igb2,iw) !aug2012
01441 else
01442 epstilde(igb1,igb2)= -vcousq(igb1)*zxq(igb1,igb2,iw)+vcousq(igb2)
01443 endif
01444 if(igb1==igb2) epstilde(igb1,igb2)=1+epstilde(igb1,igb2)
01445 enddo
01446 enddo
01447 epstinvg(ix+1:ngb,ix+1:ngb)=epstilde(ix+1:ngb,ix+1:ngb)
01448 call matcinv(ngb-ix,epstinvg(ix+1:ngb,ix+1:ngb))
01449 epsi(iw,iqixc2)= epstinvg(1,1)
01450 write(6,(' " iq iw omega eps epsi wlf="
& ,2i6,f8.3,2e23.15,3x, 2e23.15)')
01452 & iqixc2,iw,2*frr,1d0/epsi(iw,iqixc2),epsi(iw,iqixc2)
01453 write(6,*)
01454 write(ifeqsdatsnolfc,(' (3f12.8,2x,d12.4,2e23.15,2x,2e23.15)') )
01455 & q, 2*frr, 1d0/epsi(iw,iqixc2),epsi(iw,iqixc2)
01456 endif
01457 c$$$              else
01458 c$$$              write(6,*)'not support epsmode and newaniso=F mode now.'
01459 c$$$c$$$          vcmean= sum( dconjg(gbvec) * matmul(vcoul,gbvec) )
01460 .not.c$$$c$$$c$$$ if(nolfc) then
01461 c$$$c$$$c$$$          x0mean(iw,1,1) = sum( dconjg(zzr(:,1))* matmul(zxq(:,1,iw),zzr(:,1)))
01462 c$$$c$$$c$$$          endif
01463 c$$$c$$$c$$$          epsi(iw,iqixc2) = 1d0/(1- vcmean * x0mean(iw,1,1))
01464 c$$$c$$$c$$$          write(6,(' " iq iw omega eps epsi nolfc=" ,2i6,f8.3,2e23.15,3x, 2e23.15,
& " vcmean x0mean=" , 2e23.15,3x, 2e23.15)') iqixc2,iw,2*frr,
01466 c$$$c$$$c$$$          & 1d0/epsi(iw,iqixc2),epsi(iw,iqixc2),vcmean,x0mean(iw,1,1)
01467 c$$$c$$$c$$$          write(ifeqsdatsnolfc,(' (3f12.8,2x,d12.4,2e23.15,2x,2e23.15)') )
01468 c$$$c$$$c$$$          & q, 2*frr, 1d0/epsi(iw,iqixc2),epsi(iw,iqixc2)
01469 .not.c$$$c$$$c$$$ if(nolfc) then ! With LFC
01470 c$$$c$$$c$$$          imode=2
01471 c$$$c$$$c$$$          call wcf( ngb, vcoul, zxq(1,1,iw), imode,
& zw0) ! write(6,(' 'ssschk1=' ,3d13.5)') sum(abs(zw0)) sum(abs(gbvec))
01473 c$$$c$$$c$$$          epsi(iw,iqixc2)= sum( dconjg(gbvec) * matmul(zw0,zzr(:,1)) )
01474 c$$$c$$$c$$$          write(6,(' " iq iw omega eps epsi wlf="

```

```

01475 c$$$c$$$ & ,2i6,f8.3,2e23.15,3x, 2e23.15')
01476 c$$$c$$$ & iqixc2,iw,2*frr,ld0/epsi(iw,iqixc2),epsi(iw,iqixc2)
01477 c$$$c$$$ write(6,*)
01478 c$$$c$$$ write(ifepsdat,'(3f12.8,2x,d12.4,2e23.15,2x,2e23.15)')
01479 c$$$c$$$ & q, 2*frr,ld0/epsi(iw,iqixc2),epsi(iw,iqixc2)
01480 c$$$c$$$ endif
01481 c$$$ endif
01482 C --- ChipM mode
01483 .and. elseif(epsmodechipm) then
01484 allocate( x0meanx(nmbas,nmbas) )
01485 if(nolfco) then ! ChipM mode without LFC
01486 c$$$ if(legas) then
01487 c$$$ call rx( ' hx0fp0.m.F need to implement thigs here if required')
01488 c$$$! --- three lines below may work for test purpose for legas. But not sure.
01489 c$$$c vcmean= sum( dconjg(gbvec) * matmul(vcou,gbvec) )
01490 c$$$c write(ifchipmn,'(3f12.8,2x,f8.5,2x,2e23.15)')
01491 c$$$c & q, 2*schi*frr, ld0-vcmean*2*x0mean(iw,1,1)
!4*pi*alat**2/sum(q**2)/4d0/pi**2*x0mean(iw)
01492 c$$$ else
01493 c$$$ x0meanx = x0mean(iw,,:)/2d0 !in Ry unit.
01494 c$$$ endif
01495 x0meanx = x0mean(iw,,:)/2d0 !in Ry unit.
01496 else
01497 C ... ChipM mode with LFC... NoLFC part
01498 zxq(1:nbg,1:nbg,iw) = zxq(1:nbg,1:nbg,iw)/2d0 ! in Ry.
01499 do imbl=1,nmbas
01500 do imb2=1,nmbas
01501 x0meanx(imbl,imb2)=
01502 & sum( svec(1:nbloch,imb1)*
01503 & matmul(zxq(1:nbloch,1:nbloch,iw),svec(1:nbloch,imb2))) !/ mmnorm**2 I removed mmnorm
may2007
01504 enddo
01505 enddo
01506 ! x0meanx= <m|chi^+-(\omega)|m>/<m|m>**2
01507 endif
01508 do imbl=1,nmbas
01509 do imb2=1,nmbas
01510 x0meanx(imbl,imb2) =
01511 & x0meanx(imbl,imb2)/mmnorm(imbl)/mmnorm(imb2)
01512 enddo
01513 enddo
01514 write(ifchipmn_mat,'(3f12.8,2x,f20.15,2x,255e23.15)')q, 2*schi*frr, x0meanx(:, :)
01515 .not. if(nolfco) write(ifchipm_fmat) q, 2*schi*frr, zxq(1:nbloch,1:nbloch,iw)
01516
01517 c! These lines commented by "c! ' are histories ---> For Takao's memo. Maybe not so useful for others.
01518 c! ! for NoLFC, Get I from q=0, and calculate Tr(Chipm)
01519 c! allocate( x0mat(nmbas,nmbas),x0matinv(nmbas,nmbas) )
01520 c! ifx = iopen ('StonerNLFC.dat',1,3,0)
01521 c! if(iw==0 .and. sum(q**2) <ld-13) then
01522 c! x0mat = x0meanx
01523 c! x0mat(:, :) = x0mat + transpose(dconjg(x0mat))
01524 c! x0matinv= 0.5d0*x0mat
01525 c! call matcinv(nmbas,x0matinv)
01526 c! write(6,*) ' q=',q
01527 c! write(6,*) ' nmbas ifx=',nmbas,ifx
01528 c! write(6,*) ' x0matinv=',x0matinv
01529 c! allocate( evall(nmbas) )
01530 c! call diagno00(nmbas,x0matinv, evall)
01531 c! ! Note that x0matinv at omega=0 is negative definite matrix (by definition).
01532 c! do il=1,nmbas
01533 c! write(6,(' eval(iw=0)=",i5,f15.5)') i1, -evall(i1)
01534 c! enddo
01535 c! jzero2 = minval(-evall)
01536 c! deallocate(evall)
01537 c! write(ifx,"(e23.15)") jzero2
01538 c! do imb=1,nmbas !temporary
01539 c! write(ifx,"(e23.15,' ! tttt temporary... U_mm in eV')")
01540 c! & rydberg()*jzero2*mmnorm(imb)**2/momsite(imb)**2
01541 c! enddo
01542 c! elseif(iw==0) then
01543 c! read(ifx,*,end=1013,err=1013) jzero2
01544 c! goto 1014
01545 c! 1013 continue
01546 c! stop " i/o error StonerNLFC.dat"
01547 c! 1014 continue
01548 c! endif
01549 c! ifx= iclose('StonerNLFC.dat')
01550 c! if(onceww(6)) write(6,*)' i/o end: StonerNLFC.dat'
01551 c! !
01552 c! x0matinv = x0meanx
01553 c! call matcinv(nmbas,x0matinv)
01554 c! do i=1,nmbas
01555 c! x0matinv(i,i)= x0matinv(i,i) + jzero2 ! (chipm_0^+-)^-1 + I
01556 c! enddo
01557 c! x0mat= x0matinv
01558 c! do i=1,nmbas
01559 c! x0mat(i,i) = x0mat(i,i)+ img*ld-30 ! to avoid inversion error.

```

```

01560 c!          enddo
01561 c!          call matcinv(nmbas,x0mat) !this is full x0_+-
01562 c!          trr = sum( eigrm*matmul(x0mat,dconjg(eigrm)) )
01563 c!          write(ifchipmn,
01564 c!      &          '(3f12.8,2x,f20.15,2x,2e23.15,2x,2e23.15)') q, 2*schi*frr, trr,ld0/trr
01565 c!          deallocate( x0mat,x0matinv)
01566 c!
01567 c! C--- With LFC ! save or read Istoner
01568 c!          if(.not.nolfco) then
01569 c!              zzz = zxq(1:nbloch,1:nbloch,iw)
01570 c!              ifstoner = iopen ('Stoner.dat',1,3,0)
01571 c!              if( sum(q**2) < 1d-10 .and. iw==0 ) then
01572 c!                  call diagno00(nbloch,zzz,ss0)
01573 c!                  ! zzz is negative definite at omegw=0 if the ground state is stable.
01574 c!                  ! minval(ss0) is for the largest negative value (softest mode).
01575 c!                  Istoner = -ld0/minval(ss0)
01576 c!                  do ii= 1,nbloch
01577 c!                      if(verbose())>50.or.iw<=2) then
01578 c!                          write(6,"(' eig chi^0_+- =' ,
01579 c!      &                          i4,d13.5,256d13.5 )" ) ii, ss0(ii)
01580 c!                      endif
01581 c!                  enddo
01582 c!
01583 c! cxxxx thisa SVD procedure is not used now.
01584 c! c!          SVD of chi^-1: !now only look for lowest eigenvalue problem... So rather eigenvalue problem
01585 c!          instead of SVD          write(6,"(a,i5)")' ----SVD: chiinv --- iw=',iw
01586 c!          c          zxq(1:nbloch,1:nbloch,iw)=zzz
01587 c!          c          call zgesvdnn(
01588 c!      i          nbloch, zxq(1:nbloch,1:nbloch,iw),
01589 c!      o          SS0,UU0,VT0)
01590 c!          Istoner = -sum(UU0(:,1)*VT0(1,:))/ss0(1)
01591 c!          do ii= 1,nbloch
01592 c!              write(ifstoner,'(4e23.15)') UU0(ii,1),VT0(1,ii)
01593 c!          enddo
01594 c!          write(ifstoner,"(e23.15)")
01595 c!      &          Istoner
01596 c!          do imb=1,nmbas
01597 c!              write(ifstoner,"(e23.15,'!tttt temporary U_mm in eV')")
01598 c!      &          Istoner*rydberg()*mmnorm(imb)**2/momsite(imb)**2
01599 c!          enddo
01600 c!          elseif(iw==0) then
01601 c!              read(ifstoner,*) Istoner
01602 c!          endif
01603 c!          ifstoner = iclose('Stoner.dat')
01604 c! C... <eqir| 1/(1 + I chi^0_+-) | eqir>
01605 c!          mmat = + Istoner * zzz
01606 c!          do i = 1, nbloch
01607 c!              mmat(i,i) = mmat(i,i) + 1d0
01608 c!          enddo
01609 c!          trr0 = sum( dconjg(zzr(1:nbloch,1))*
01610 c!      &              matmul( mmat,zzr(1:nbloch,1) ) )
01611 c!          c          write(6,"(' <eqir| 1 + I chi0^+-|eqir> =' ,255e23.15)") trr0
01612 c!          do i=1,nbloch
01613 c!              mmat(i,i) = mmat(i,i)+ img*ld-30 ! to avoid inversion error.
01614 c!          enddo
01615 c!
01616 c! c$$$c prtest for NiO with 4 bloch basis
01617 c! c$$$          zzzx = mmat !matmul(sproj,matmul(mmat,sproj))
01618 c! c$$$          call zgesvdnn(
01619 c!      i          nbloch, zzzx,
01620 c!      o          eex,UU0,VT0)
01621 c! c$$$! projected denominator
01622 c! c$$$          denom = matmul( sproj,matmul(Istoner * zzz,sproj))
01623 c! c$$$          do i = 1, nbloch
01624 c! c$$$              denom(i,i) = denom(i,i) + 1d0
01625 c! c$$$          enddo
01626 c! c$$$!
01627 c! c$$$          zzzx=denom
01628 c! c$$$          call zgesvdnn(
01629 c!      i          nbloch, zzzx,
01630 c!      o          eey,UU0,VT0)
01631 c! c$$$          write(ifchipm2,
01632 c!      &          '(3f12.8,2x,f20.15,2x,4f11.5,3x,4f11.5)') q, 2*schi*frr, eex,eey
01633 c! c$$$!
01634 c! c$$$          zzz=denom
01635 c! c$$$          do i=1,nbloch
01636 c! c$$$              zzz(i,i) = zzz(i,i)+ img*ld-30 ! to avoid inversion error.
01637 c! c$$$          enddo
01638 c! c$$$          call matcinv(nbloch, zzz)
01639 c! c$$$Ctest --- another inversion procedure ! zzz is the inverse of denom
01640 c! c$$$          VT = dconjg(transpose(UU0))
01641 c! c$$$          UU = dconjg(transpose(VT0))
01642 c! c$$$          zzz=0d0
01643 c! c$$$          do i=1,nbloch
01644 c! c$$$              do ix=1,nbloch
01645 c! c$$$                  do iy=1,nbloch

```

```

01646 c! c$$$$ zzzz(ix,iy) = zzzz(ix,iy) + UU(ix,i)*VT(i,iy)/eey(i)
01647 c! c$$$$ enddo
01648 c! c$$$$ enddo
01649 c! c$$$$ enddo
01650 c! c$$$$ zzzx = matmul(denom,zzzy)
01651 c! c$$$$ do i=1,nbloch
01652 c! c$$$$ do j=1,nbloch
01653 c! c$$$$ write(6, "('zzzx=',2i5,2d13.6)")i,j,zzzx(i,j)
01654 c! c$$$$ enddo
01655 c! c$$$$ enddo
01656 c! c$$$$ zzzx = matmul( sproj,matmul(zzz,sproj) )
01657 c! c$$$$ mmatx = matmul(zzzx, zzzzy)
01658 c! c$$$$ trrx = sum( dconjg(zzr(1:nbloch,1)) *
01659 c! c$$$$ & matmul(mmatx,zzr(1:nbloch,1)) )
01660 c! c$$$$cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01661 c! call matcinv(nbloch, mmat)
01662 c! mmat = matmul(zzz,mmat)
01663 c! c--- prtest I found that This makes the difference at high energy part!!! Nov-9-2006
01664 c! c--- mmat = matmul( sproj,matmul(mmat,sproj) )
01665 c! trr = sum( dconjg(zzr(1:nbloch,1)) *
01666 c! & matmul(mmat,zzr(1:nbloch,1)) )
01667 c! write(ifchipm,
01668 c! & '(3f12.8,2x,f20.15,2x,2e23.15,2x,4e23.15)')
01669 c! & q, 2*schi*frr, trr, ld0/trr
01670 c! endif
01671 c! deallocate(x0meanx)
01672 c! endif
01673 c write(6,*)'tttt aaa iw=',iw
01674 1015 continue
01675 c write(6,*)'tttt end of do 1015 loop'
01676 c if(newaniso2) then
01677 c if(allocated(sk)) deallocate(sk),sks,w_k,w_ks)
01678 c endif
01679
01680
01681 c
01682 c if(chipm.and.(.not.nolfco))
01683 c & deallocate(sqovlp,sqovlpi,uu0,vt0,ss0,mmat,zzz)
01684 c$$$$ if( ixc==5.or.ixc==6 ) then
01685 c$$$$ jpm=1
01686 c$$$$ nwmax = nw
01687 c$$$$ if(ixc==5) nwmax =nw
01688 c$$$$ allocate(trwv(nw_i:nw),trwv2(nw_i:nw))
01689 c$$$$ do iw = nw_i,nw !max ! trace check
01690 c$$$$ trwv(iw) = zxq(6,7,iw)
01691 c$$$$ trwv2(iw) = 0d0
01692 c$$$$ do i = 1,ngb
01693 c$$$$ trwv2(iw) = trwv2(iw) + zxq(i,i,iw)
01694 c$$$$ enddo
01695 c$$$$ enddo
01696 c$$$$ do iw= nw_i,nw-1
01697 c$$$$ if(ixc==5)
01698 c$$$$ & write(6, "('iq iw[min_max]=' ,2i5,2f7.4,' trwv by wwk*h= ',
01699 c$$$$ & 12d13.5)") iq, iw, freq_r(iw), freq_r(iw+1),
01700 c$$$$ & (trwv2(iw)+trwv2(iw+1))/2d0*(freq_r(iw)-freq_r(iw+1)),
01701 c$$$$ & (trwv(iw)+trwv(iw+1)) /2d0*(freq_r(iw)-freq_r(iw+1))
01702 c$$$$ !weight for the histogram range. by tetwt5
01703 c$$$$ if(ixc==6)
01704 c$$$$ & write(6, "('iq iw[min_max]=' ,2i5,2f7.4,' trwv by whw = ',
01705 c$$$$ & 12d13.5)") iq,iw, freq_r(iw), freq_r(iw+1),
01706 c$$$$ & trwv2(iw),trwv(iw) !weight for the histogram range. by tetwt5
01707 c$$$$ enddo
01708 c$$$$ deallocate(trwv,trwv2)
01709 c$$$$ endif
01710
01711 c if( allocated(zzr) ) deallocate(zzr)
01712 c if( allocated(x0mean) ) deallocate(x0mean)
01713 c if( allocated(gbvec) ) deallocate(gbvec)
01714
01715
01716 c ... Close files for epsmode
01717 c if(epsmode) then !iepsmode/=0) then ! only calculate iq>nqibz
01718 c if(chipm) then
01719 c ifchipmn_mat=iclose('ChiPM'//charnum4(iqixc2)///'.nlfcmat')
01720 c if(.not.nolfco) then
01721 c ifchipm_fmat=iclose( 'ChiPM'//charnum4(iqixc2)///'.fmat')
01722 c endif
01723 c else
01724 c filepsnolfc = 'EPS'//charnum4(iqixc2)///'.nolfc.dat'
01725 c ifepsdatnolfc = iclose( filepsnolfc)
01726 c if(.not.nolfco) then
01727 c fileps = 'EPS'//charnum4(iqixc2)///'.dat'
01728 c ifepsdat = iclose(fileps)
01729 c endif
01730 c endif
01731 c endif
01732 c --- realomega end =====

```

```

01733
01734
01735
01736 c --- imagomega =====
01737     if (imagomega) then
01738         write(6,*) ' goto imag omega'
01739         if (nspin == 1) zxqi = 2d0*zxqi ! if paramagnetic, multiply x0 by 2
01740         if (ecorr_on>0) then !ixc==101.or.(sergeyv.and.imagonly) then
01741             imode=0
01742         else
01743             imode=1
01744         endif
01745
01746 !! === iw loop for imag axiw ===
01747     do 1016 iw = 1,niw
01748 c         if(newaniso2.and. iq<=ngibz ) then
01749             if( iq<=ngibz ) then
01750 !! Eqs.(37),(38) in PRB81 125102
01751                 if(iq==1) then
01752                     ix=1
01753                     zw0(:,1)=0d0
01754                     zw0(1,:)=0d0
01755                 else
01756                     ix=0
01757                 endif
01758 !! Eqs.(37),(38) in PRB81 125102
01759                 do igb1=ix+1,ngb
01760                     do igb2=ix+1,ngb
01761                         epstilde(igb1,igb2)= -vcousq(igb1)*zxqi(igb1,igb2,iw)*vcousq(igb2)
01762                         if(igb1==igb2) epstilde(igb1,igb2)=1+epstilde(igb1,igb2)
01763                     enddo
01764                 enddo
01765                 epstinvs=epstilde
01766                 call matcinv(ngb-ix,epstinvs(ix+1:ngb,ix+1:ngb))
01767                 do igb1=ix+1,ngb
01768                     do igb2=ix+1,ngb
01769                         zw0(igb1,igb2)= vcousq(igb1)*epstinvs(igb1,igb2)*vcousq(igb2)
01770                         if(igb1==igb2) zw0(igb1,igb2)= zw0(igb1,igb2)-vcousq(igb1)*vcousq(igb2)
01771                     enddo
01772                 enddo
01773 c                 if(iq==1) write(ifepstinvs) epstinvs(ix+1:ngb,ix+1:ngb),iq,iw
01774             endif
01775 c         if(newaniso2.and.iq>ngibz) then
01776             if(iq>ngibz) then
01777 !! Full inversion to calculalte eps with LFC.
01778                 ix=0
01779                 vcoul = fourpi/sum(q**2*tpioa**2) ! --> vcousq(1)**2 !fourpi/sum(q**2*tpioa**2-eee)
01780                 do igb1=ix+1,ngb
01781                     do igb2=ix+1,ngb
01782                         if(igb1==1.and.igb2==1) then
01783                             epstilde(igb1,igb2)= 1d0 - vcoul*zxqi(1,1,iw)
01784                             cycle
01785                         endif
01786                         epstilde(igb1,igb2)= -vcousq(igb1)*zxqi(igb1,igb2,iw)*vcousq(igb2)
01787                         if(igb1==igb2) then
01788                             epstilde(igb1,igb2)=1d0 + epstilde(igb1,igb2)
01789                         endif
01790                     enddo
01791                 enddo
01792                 epstinvs(ix+1:ngb,ix+1:ngb)=epstilde(ix+1:ngb,ix+1:ngb)
01793                 call matcinv(ngb-ix,epstinvs(ix+1:ngb,ix+1:ngb))
01794                 llwi(iw,iq0)= 1d0/epstinvs(1,1)
01795                 write(6,*) 'iq iw_img eps(withLFC) eps(woLFC)',iq,iw,llwi(iw,iq0),1d0-vcoul*zxqi(1,1,iw)
01796
01797 c$$$c         read(ifepstinvs) epstinvs(2:ngb,2:ngb),iqx,iwx
01798 c$$$c         if(iw/=iwx) then
01799 c$$$c             write(6,*) 'iw iwx=',iw,iwx
01800 c$$$c             stop 'hx0fp0: iw=iwx' !sanity check
01801 c$$$c         endif
01802 c$$$cmmmm3
01803 c$$$c         ski(1:ngb)= zxqi(1,1:ngb,iw)
01804 c$$$c         sksi(1:ngb)= zxqi(1:ngb,1,iw)
01805 c$$$c         ski(1:ngb)= zxqi(1,1:ngb,iw)
01806 c$$$c         sksi(1:ngb)= zxqi(2,1:ngb,iw)
01807 c$$$c         vcoul = fourpi/sum(q**2*tpioa**2) ! test-->vcousq(1)**2 !fourpi/sum(q**2*tpioa**2-eee)
01808 c$$$c         vcoulsq= sqrt(vcoul)
01809 c$$$c         write(ifiss) iw,iq0,ngb,q
01810 c$$$c         write(ifiss) vcoul,vcoulsq,vcousq(2:ngb),ski(1:ngb),sksi(1:ngb)
01811 c$$$c         w_ki(1) = 0d0
01812 c$$$c         w_ksi(1)= 0d0
01813 c$$$c         w_ki( 2:ngb)= vcoulsq*matmul( ski(2:ngb)*vcousq(2:ngb), epstinvs(2:ngb,2:ngb) )
01814 c$$$c         w_ksi(2:ngb)= vcoulsq*matmul( epstinvs(2:ngb,2:ngb), sksi(2:ngb)*vcousq(2:ngb))
01815 c$$$cmmmm epspp mode ---> no - vcoulsq*sum( ski(2:ngb) * w_ksi(2:ngb)*vcousq(2:ngb) )
01816 c$$$c         llwi(iw,iq0)= 1d0 -vcoul*ski(1) !- vcoulsq*sum( ski(2:ngb) * w_ksi(2:ngb)*vcousq(2:ngb) )
01817 c$$$c
01818 c$$$c         llwi(iw,iq0)= 1d0 - vcoul*zxqi(1,1,iw) !- vcoulsq*sum( ski(2:ngb) *

```

```

w_ksI(2:ngb)*vcousq(2:ngb) )
01819 c$$$ write(6,*) 'iq iw llwI',iq,iw,llwi(iw,iq0)
01820 c$$$c write(ifisk) iw,iq0,q
01821 c$$$c write(ifisk) vcousq(2:ngb)*w_ki(2:ngb),vcousq(2:ngb)*w_ksi( 2:ngb)
01822 endif
01823
01824 c$$$ if(.not.newaniso2) then ! original mode
01825 c$$$ call rx( 'not checked here')
01826 c$$$c call wcf( ngb, vcoul,zxqi(1,1,iw),imode, zw0)
01827 c$$$ endif
01828
01829 c if(newaniso2.and.iq>nqibz) then
01830 if(iq>nqibz) then
01831 c zw(1:ngb,1:ngb) = 0d0 ! zw(nblochpmx,nblochpmx)
01832 c write(ifrcwi, rec=(iq-ixxini)*niw + iw) zw ! WP = vsc-v
01833 else
01834 c zw(1:ngb,1:ngb) = zw0 ! zw(nblochpmx,nblochpmx)
01835 c write(ifrcwi, rec=(iq-ixxini)*niw + iw) zw ! WP = vsc-v
01836 c write(ifrcwi, rec=iw) zw ! WP = vsc-v
01837 c call tr_chkwite("freq_i iq iw imgomg trwv=",zw,iw,freq_i(iw),nblochpmx,nbloch,ngb
,iq)
01838 endif
01839
01840
01841 !! --- Miyake's total energy branch !Nov2004. not maintained now... need to fix this maybe(2012takao)
-----
01842 if(.false.) then
01843 c if(ecorr_on>0 .and. (.not.newaniso2)) then !I did not modified this for newaniso2 2012takao
01844 if (debug) write(6,*) 'ip,ix=',iq,iw,' niw=',niw
01845 call getwk(iq, wibz, wgt,ngbz,nqibz,nstibz,nq0i, wk4ec)
01846 call ecorq2(vcoul, zw0, ngb, iq,iw,ieceig,
01847 o erpaqw, trpvqw, trlogqw)
01848 c --- integration along imaginary axis.
01849 ! omit k and basis index for simplicity
01850 ! wint = -(i/4pi) < [w'=-inf,inf] Q(w') >
01851 !
01852 ! When w' ==> iw', w' is now real,
01853 ! wint = (1/2pi) < [w'=0,inf] Q(iw') >
01854 !
01855 ! transform: x = 1/(1+w')
01856 ! this leads to a denser mesh in w' around 0 for equal mesh x
01857 ! which is desirable since Q is peaked around w'=0
01858 ! wint = (1/2pi) < [x=0,1] Q(iw') / x^2 >
01859 faca = wk4ec* wiw(iw)
01860 trpv(iecut) = trpv(iecut) + faca* trpvqw
01861 trlog(iecut) = trlog(iecut) + faca* trlogqw
01862 totexc(iecut) = totexc(iecut)+ faca* erpaqw ! = trpv+ trlog
01863 c ecqw(iq,iw) = erpaqw
01864 if(iw==1) then
01865 write(ieclg,*)
01866 endif
01867 if(iw==1.and.iq==ixxini) then
01868 write(ieclg,
01869 & "(' iq iw omega/i(Ry) IntWgt',
01870 & ' trpvqw(eV) ecqw(eV) ecqw*IntWgt',
01871 & ' : ecut ecuts')")
01872 endif
01873 write(ieclg,"( 2i5,3f14.6,3f14.6,2f8.3)")
01874 & iq,iw, 2d0*freq_i(iw), faca, trpvqw*hartree, erpaqw*hartree,
01875 & faca*erpaqw*hartree, ecut(iecut),ecuts(iecut)
01876 close(ieclg)
01877 open(ieclg,file="ecorr.chk",access='append')
01878 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01879 c allocate( ovlpc(ngb,ngb),evall(ngb),
01880 c & evecc(ngb,ngb))
01881 c evall=0d0
01882 c ovlpc=0d0
01883 c do i=1,ngb
01884 c ovlpc(i,i)=1d0
01885 c enddo
01886 c nmix=ngb
01887 c call diagcv(ovlpc,zw0/2d0+transpose(dconjg(zw0))/2d0,evecc,ngb, evall,nmix,1d99, nev)
01888 c call diagcv(ovlpc,zw0,evecc,ngb, evall,nmix,1d99, nev)
01889 c write(6,"('ngb nev=' ,2i5)") ngb,nev
01890 c write(6,"('chk eigen of zw0 Max Min=' ,2d13.6)")maxval(evall),minval(evall)
01891 c do i=1,3
01892 c write(6,*) i, evall(i)
01893 c enddo
01894 c do i=ngb-3,ngb
01895 c write(6,*) i, evall(i)
01896 c enddo
01897 c deallocate( ovlpc,evall,evecc)
01898 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
01899 c endif
01900 1016 continue
01901 c if(newaniso2) then
01902 deallocate(epstin)

```

```

01903         if(allocated(epstilde)) deallocate(epstilde)
01904         if(allocated(epstilde)) deallocate(epstilde)
01905 c         endif
01906 c$$$      enddo
01907      endif
01908 c... imagomega end =====
01909         if(allocated(vcou1)) deallocate(vcou1)
01910         if(allocated(zw0)) deallocate(zw0)
01911         if(allocated(zxq )) deallocate(zxq)
01912         if(allocated(zxqi)) deallocate(zxqi)
01913         if (ixc==101.or.normalm) then
01914             ifrcwi = iclose('WVI.'//charnum5(iq))
01915         endif
01916         if (normalm) then
01917             ifrcw = iclose('WVR.'//charnum5(iq))
01918         endif
01919 1001 continue
01920 !! == end of loop 1001 for q point ==
01921     call mpi_barrier()
01922
01923
01924 !! === Recieve llw and llwI at node 0, where q=0(iq=1) is calculated. ===
01925     if(mpi_size/=1) then
01926         do iq=nqibz+1,iqzend
01927             iq0 = iq - nqibz
01928 c         write(6,*)' iq iq0 mpi_rank mpi_ranctab(iq)=',iq, iq0,mpi_rank,mpi_ranctab(iq),mpi_root,nw,nw_i,
niw
01929             if(mpi_ranctab(iq)/=0) then !jan2012
01930                 if(mpi_ranctab(iq) == mpi_rank) then
01931                     dest=0
01932                     call mpi_dblecomplexsend(llw(nw_i,iq0),(nw-nw_i+1),dest)
01933                     call mpi_dblecomplexsend(llwi(1,iq0),niw,dest)
01934                 elseif(mpi_root) then
01935 c         write(6,*)' mpi_recv iq from',iq,mpi_ranctab(iq),nw,nw_i,niw
01936                     src=mpi_ranctab(iq)
01937                     call mpi_dblecomplexrecv(llw(nw_i,iq0),(nw-nw_i+1),src)
01938                     call mpi_dblecomplexrecv(llwi(1,iq0),niw,src)
01939 c         do i=nw_i,nw
01940 c         write(6,*)'recivxxx',i,llw(i,iq0)
01941 c         enddo
01942 c         write(6,*)' recv llw sum=',sum(abs(llw(:,iq0))),nw,nw_i
01943 c         write(6,*)' recv llwI sum=',sum(abs(llwi(:,iq0))),niw
01944             endif
01945         endif
01946     enddo
01947 endif
01948
01949 c$$$      deallocate( llw, llwi )
01950 c$$$!! == generate llw and llwI ==
01951 c$$$      ifiss=iopen('SkSks',0,-1,0)
01952 c$$$      ifepstinv = iopen('EPS0inv',0,0,0)
01953 c$$$      allocate( llw(nw_i:nw,nq0i), llwi(niw,nq0i) )
01954 c$$$      read(ifepstinv) ngb
01955 c$$$      do 1501 iq0=1,nq0i
01956 c$$$          rewind ifepstinv
01957 c$$$          read(ifepstinv) ngb
01958 c$$$          allocate(vcoulsq(2:ngb),sk(ngb),sks(ngb),w_k(ngb),w_ks(ngb) )
01959 c$$$          do iw=nw_i,nw
01960 c$$$              read(ifepstinv) epstinv(2:ngb,2:ngb),iqx,iwx
01961 c$$$              read(ifiss) iw,iq0x,ngb,q
01962 c$$$              if(iw/=iwx) stop 'hx0fp0:1501 iw/=iwx'
01963 c$$$              read(ifiss)vcoul,vcoulsq,vcousq(2:ngb),sk(1:ngb),sks(1:ngb)
01964 c$$$              w_k(1) =0d0
01965 c$$$              w_ks(1)=0d0
01966 c$$$              w_k( 2:ngb)= vcoulsq*matmul( sk(2:ngb)*vcousq(2:ngb), epstinv(2:ngb,2:ngb) )
01967 c$$$              w_ks(2:ngb)= vcoulsq*matmul( epstinv(2:ngb,2:ngb), sks(2:ngb)*vcousq(2:ngb))
01968 c$$$              !! epsPP mode - vcoulsq*sum( sk(2:ngb) * w_ks(2:ngb)*vcousq(2:ngb) )
01969 c$$$              llw(iw,iq0)= 1d0 -vcoul*sks(1) !- vcoulsq*sum( sk(2:ngb) * w_ks(2:ngb)*vcousq(2:ngb) )
01970 c$$$              write(6,*) 'epsPP iq iw',iq,iw, 1d0 - fourpi* sk(1)/sum(q**2*tpioa**2)
01971 c$$$          enddo
01972 c$$$          deallocate(vcoulsq,sk,sks,w_k,w_ks)
01973 c$$$          allocate(vcoulsq(2:ngb),ski(ngb),sksi(ngb),w_ki(ngb),w_ksi(ngb) )
01974 c$$$          do iw=1,niw
01975 c$$$              read(ifepstinv) epstinv(2:ngb,2:ngb),iqx,iwx
01976 c$$$              read(ifiss) iw,iq0x,ngb,q
01977 c$$$              if(iw/=iwx) stop 'hx0fp0:1501 iw/=iwx'
01978 c$$$              read(ifiss) vcoul,vcoulsq,vcousq(2:ngb),ski(1:ngb),sksi(1:ngb)
01979 c$$$              w_ki(1) = 0d0
01980 c$$$              w_ksi(1)= 0d0
01981 c$$$              w_ki( 2:ngb)= vcoulsq*matmul( ski(2:ngb)*vcousq(2:ngb), epstinv(2:ngb,2:ngb) )
01982 c$$$              w_ksi(2:ngb)= vcoulsq*matmul( epstinv(2:ngb,2:ngb), sksi(2:ngb)*vcousq(2:ngb))
01983 c$$$              !! epsPP mode ---> no - vcoulsq*sum( ski(2:ngb) * w_ksi(2:ngb)*vcousq(2:ngb) )
01984 c$$$              llwi(iw,iq0)= 1d0 -vcoul*ski(1) !- vcoulsq*sum( ski(2:ngb) * w_ksi(2:ngb)*vcousq(2:ngb) )
01985 c$$$              write(6,*) 'iq iw llwI',iq,iw,llwi(iw,iq0)
01986 c$$$          enddo
01987 c$$$          deallocate(vcoulsq,ski,sksi,w_ki,w_ksi)
01988 c$$$ 1501 continue

```



```

01989
01990 !! == W(0) divergent part and W(0) non-analytic constant part.==
01991 !! Note that this is only for q=0 -->iq=1
01992 c      if(newaniso2.and.ixc==11.and.mpi__rank==0) then
01993      if((ixc==11.or.ixc==10011.or.ixc==111).and.mpi__rank==0) then
01994 !! get w0 and w0i (diagonal element at Gamma point
01995 !! This return w0, and w0i
01996      call w0w0i(11w,11wi,nw_i,nw,nq0i,niw,q0i)
01997 !! === w0,w0i are stored to zw for q=0 ===
01998 !! === w_ks*wk are stored to zw for iq >nqibz ===
01999      do iq = 1,1      !iq=1 only 4pi/k**2 /eps part only ! iq = iqxini,iqxend
02000      q = qibze(:,iq)
02001 c      if(iq>nqibz) then
02002 c      iq0 = iq - nqibz
02003 c      read(ifisk) ngb,nw_ixxx,nwxxx,niwxxx
02004 c      allocate(vw_k(ngb),vw_ks(ngb))
02005 c      endif
02006      do ircw=1,2
02007      if (ircw==1) then; nini=nw_i; nend=nw;
02008      ifrcwx = iopen('WVR.'//charnum5(iq),0,-1,mrecl)
02009      elseif(ircw==2) then; nini=1; nend=niw;
02010      ifrcwx = iopen('WVI.'//charnum5(iq),0,-1,mrecl)
02011      endif
02012      do iw=nini,nend
02013 c      if(iq<=nqibz) read(ifrcwx, rec=((iq-iqxini)*(nend-nini+1)+ iw-nini+1 ) ) zw !(1:ngb,1:ngb)
02014      read(ifrcwx, rec= iw-nini+1 ) zw !(1:ngb,1:ngb)
02015 c      if( iq==1 ) then
02016      if(ircw==1) zw(1,1) = w0(iw)
02017      if(ircw==2) zw(1,1) = w0i(iw)
02018
02019 ccccccccccccccccccccccccccccccccccccccccccccccccccc
02020 c      if(ircw==1) zw(1,1) = 0d0
02021 c      if(ircw==2) zw(1,1) = 0d0
02022      if(ircw==1) then
02023      write(6, "('ffffrrr:', f13.6,2x,f13.6,x,f13.6)") hartree*freq_r(iw),w0(iw)
02024      endif
02025 ccccccccccccccccccccccccccccccccccccccccccccccccccc
02026
02027
02028 c$$$cmmm3 ccccccccccccccccccccccccccccccccccccccccccc
02029 c$$$      elseif( iq>nqibz ) then !-->In future, we store sperical average of zw below to zw(at q=0)===
02030 c$$$      write(6,*)'ddd skip readin ifisk dddddd'
02031 c$$$      read(ifisk) iwxx,iq0xx,qxx
02032 c$$$      if(iwxx /=iw) stop 'iwxx/=iw'
02033 c$$$      if(iq0xx /=iq-nqibz) stop 'iq0xx /=iq'
02034 c$$$      if(sum(abs(qibze(:,iq)-qxx))>1d-8) stop 'sum(abs(qq-qxx))>1d-8'
02035 c$$$      read(ifisk) vw_k(2:ngb),vw_ks(2:ngb)
02036 c$$$      zw=0d0
02037 c$$$      do igb1=1+1,ngb
02038 c$$$      do igb2=1+1,ngb
02039 c$$$      vclvc2 = vw_ks(igb1)*vw_k(igb2)
02040 c$$$      if(ircw==1) zw(igb1,igb2)=vclvc2/11w(iw,iq0)
02041 c$$$      if(ircw==2) zw(igb1,igb2)=vclvc2/11wi(iw,iq0)
02042 c$$$      enddo
02043 c$$$      enddo
02044 c$$$cccccccccccccccccccccccccccccccccccccccccccccccccc
02045 c      endif
02046 c      if(iq==1.or.iq>nqibz) write(ifrcwx,rec=((iq-iqxini)*(nend-nini+1)+ iw-nini+1 ) ) zw
!(1:ngb,1:ngb)
02047 c      write(ifrcwx,rec=((iq-iqxini)*(nend-nini+1)+ iw-nini+1 ) ) zw !(1:ngb,1:ngb)
02048      write(ifrcwx,rec= iw-nini+1 ) zw !(1:ngb,1:ngb)
02049 ccccccccccccccccccccccccccccccccccccccccccccccccccc
02050 cmmmm3
02051 c      if(iq<=nqibz) then
02052 c      zw=0d0
02053 c      write(ifrcwx, rec=((iq-iqxini)*(nend-nini+1)+ iw-nini+1 ) ) zw !(1:ngb,1:ngb)
02054 c      endif
02055 ccccccccccccccccccccccccccccccccccccccccccccccccccc
02056 cooo check write
02057 c      if(mod(iw,3)==1) then
02058 c      do igb1=1,ngb,23
02059 c      do igb2=1,ngb,23
02060 c      if(ircw==1) write(*, "('zzzwr:',4i4,2d13.5)")iq,iw,igb1,igb2,zw(igb1,igb2)
02061 c      if(ircw==2) write(*, "('zzzwi:',4i4,2d13.5)")iq,iw,igb1,igb2,zw(igb1,igb2)
02062 c      enddo
02063 c      enddo
02064 c      endif
02065 ccccccccccccccccccccccccccccccccccccccccccccccccccc
02066      enddo
02067      if (ircw==1) then
02068      ifrcwx = iclose('WVR.'//charnum5(iq))
02069      elseif(ircw==2) then
02070      ifrcwx = iclose('WVI.'//charnum5(iq))
02071      endif
02072      enddo
02073      enddo
02074      endif

```



```

02075
02076
02077 c$$$!! --- legas mode is not working now. Need fixing... voltot ntot are not given.
02078 c$$$      if(epsmode.and.legas) then
02079 c$$$          call rx( ' LEGAS mode is not maintained well. Need some fixing.')
02080 c$$$          voltot=0d0
02081 c$$$          ntot=0d0
02082 c$$$          write(6,*)' Find LEGAS. legas =',legas
02083 c$$$          iflegas = 2101
02084 c$$$          open (iflegas,file='LEGAS')
02085 c$$$          read(iflegas,*)rs
02086 c$$$          close(iflegas)
02087 c$$$          alpha = (9*pi/4d0)**(1d0/3d0)
02088 c$$$          qfermi = alpha/rs
02089 c$$$          efx = qfermi**2
02090 c$$$          valn = efx*1.5d0*voltot/3d0/pi**2
02091 c$$$          write (6,*)' ##### egas test mode legas=T ##### given rs =',rs
02092 c$$$          write (6,*)'      Exact Fermi momentum qf =', qfermi
02093 c$$$          write (6,*)'      Exact Fermi energy Ef =', efx
02094 c$$$          do iq = iqxini,iqxend ! q=(0,0,0) is omitted!
02095 c$$$              if(iq<=nqibz) cycle
02096 c$$$              write(6,*)' iq=',iq
02097 c$$$              iqxc2 = iq- (nqibz+nq0ix)
02098 c$$$              filele = 'EPSEG'//charnum4(iqxc2)//'.dat'
02099 c$$$              ife = iopen( filele,1,3,0)
02100 c$$$              write(ife,"(a)")
02101 c$$$              &      ' q(1:3) w(Ry) eps epsi --- NO LFC'
02102 c$$$              q = qibze(:,iq)
02103 c$$$              qt= sqrt(sum(qibze(1:,iq)**2))*2d0*pi/alat
02104 c$$$              qs= qt/qfermi
02105 c$$$              write(6,"(' qs qfermi=',2d13.5)" ) qs,qfermi
02106 c$$$              write(6,"(' q-q^2/2 q+q^2=',2d13.5)" ) qs-qs**2/2d0,qs+qs**2/2d0
02107 c$$$              do iw = nw_i,nw
02108 c$$$                  ww = freq_r(iw)
02109 c$$$                  muu = ww/qfermi**2
02110 c$$$                  if( qs<2d0 .and. muu < qs-qs**2/2d0) then
02111 c$$$                      x0mx= -img*qfermi/(4*pi*qs)*2*muu
02112 c$$$                  elseif( qs<2d0 .and. muu < qs+qs**2/2d0) then
02113 c$$$                      x0mx= -img*qfermi/(4*pi*qs)*( 1d0-(muu/qs-.5d0*qs)**2 )
02114 c$$$                  else
02115 c$$$                      x0mx=0d0
02116 c$$$                  endif
02117 c$$$                  vcmmmm= 4*pi/qt**2
02118 c$$$                  epsi(iw,iqxc2) = 1d0/(1- vcmmmm * x0mx)
02119 c$$$                  epsi(iw,iqxc2) = 1d0/(1- vcmmmm(iq) * x0meanx)
02120 c$$$                  write(ife,'(3f12.8,2x,d12.4,2e23.15,2x,2e23.15)')
02121 c$$$              &      q, 2*ww,1d0/epsi(iw,iqxc2),epsi(iw,iqxc2)
02122 c$$$              enddo
02123 c$$$          enddo
02124 c$$$          write(6,*)' -----legas end-----'
02125 c$$$      endif
02126
02127 !! Write TEECOR ecorr_on mode
02128      if(imagomega.and.ecorr_on>0) then
02129          hartree=2d0*rydberg()
02130          ifcor = iopen('TEECORR2',1,-1,0) ! output files
02131          do iecut=1,necut
02132              write(6,"( ' RPA Ec = ' 3f23.15,' ecut ecuts (Ry)=' ,2d12.4)")
02133              &      totexc(iecut)*hartree,trpv(iecut)*hartree, trlog(iecut)*hartree
02134              &      ,ecut(iecut),ecuts(iecut)
02135              write(ifcor,*) '=====
02136              write(ifcor,*) 'Correlation energy Erpa (eV)'
02137              write(ifcor,*) '=====
02138              write(ifcor,*)' ### '
02139              write(ifcor,"(5e23.15)")
02140              &      totexc(iecut)*hartree,trpv(iecut)*hartree, trlog(iecut)*hartree
02141              &      ,ecut(iecut),ecuts(iecut)
02142          enddo
02143          c... output ecqw !      write(ifcor,*)'### ecqw(q,w) ###'
02144          write(ifcor,*)' nqibz =',nqibz
02145          write(ifcor,*)' nq0i =',nq0i
02146          write(ifcor,*)' niw =',niw
02147          write(ifcor,*)' --- See details of Ec in ecor.chk ---'
02148          c      nqitot = nqibz + nq0i
02149          c      call wecqwf(ifcor,
02150          c          d      nqibz,nqibz,nq0i,nqitot,niw,
02151          c          o      wibz,wqt,wx,freqx,ecqw)
02152          c... Write electron gas correlation energy
02153          c$$$      legas = .false.
02154          c$$$      INQUIRE (file = 'LEGAS', exist = legas)
02155          c$$$      if(legas) then !!! test for electron gas case.
02156          c$$$          call rx( ' LEGAS mode is not maintained well. Need some fixing.')
02157          c$$$          voltot=0d0
02158          c$$$          ntot=0d0
02159          c$$$          write(6,*)' find LEGAS. legas =',legas
02160          c$$$          iflegas = 2101
02161          c$$$          open (iflegas,file='LEGAS')

```

```

02162 c$$$      read(iflegas,*)rs
02163 c$$$      close(iflegas)
02164 c$$$      alpha = (9*pi/4d0)**(1d0/3d0)
02165 c$$$      qfermi = alpha/rs
02166 c$$$      efx = qfermi**2
02167 c$$$      valn = efx**1.5d0*voltot/3d0/pi**2
02168 c$$$      write (6,*)' ##### egas test mode  legas=T ##### given rs =',rs
02169 c$$$      write (6,*)' egas  Exact Fermi momentum  qf  =', qfermi
02170 c$$$      write (6,*)' egas  Exact Fermi energy    Ef  =', efx
02171 c$$$      if(tetra) call rx( 'legas You have to give ef of tetrahedron')
02172 c$$$      efz=(ntot*3*pi**2/voltot)**(2d0/3d0) ! ef is calculated from ntot.
02173 c$$$      qfermi= dsqrt(efz)
02174 c$$$      alpha = (9*pi/4d0)**(1d0/3d0)
02175 c$$$      rs = alpha/qfermi
02176 c$$$      write (ifcor,*)' --- electron gas ---'
02177 c$$$      write (ifcor,*)' density parameter rs= ', rs
02178 c$$$      write (ifcor,*)' kf= ',qfermi
02179 c$$$      write (ifcor,*)' ### Barth-Hedin formula'
02180 c$$$      ecelgas = eclda_bh(rs) * hartree * ntot
02181 c$$$      write (ifcor,*)ecelgas
02182 c$$$      write (ifcor,*)' ### Perdew-Zunger formula'
02183 c$$$      ecelgas = eclda_pz(rs) * hartree * ntot
02184 c$$$      write (ifcor,*)ecelgas
02185 c$$$      write (ifcor,*)' ### Gell-Mann and Brueckner formula'
02186 c$$$      ecelgas = (-0.0311d0 * dlog(rs) -0.048d0) * hartree * ntot
02187 c$$$      write (ifcor,*)ecelgas
02188 c$$$      endif
02189
02190      call cputid(0)
02191      call mpi_finalize
02192      if(ixc==11) call rx0( ' OK! hx0fp0 mode=11      read <Q0P> normal sergeyv')
02193      if(ixc==111) call rx0( ' OK! hx0fp0 mode=111      normal sergeyv')
02194      if(ixc==10011)call rx0( ' OK! hx0fp0 mode=10011      crpa normal sergeyv')
02195      if(ixc==12) call rx0( ' OK! hx0fp0 mode=12      Ecor sergeyv mode')
02196      if(ixc==101) call rx0( ' OK! hx0fp0 mode=101      Ecor ')
02197      if(ixc==202) call rx0( ' OK! hx0fp0 mode=202 sergeyv epsPP NoLFC')
02198      if(ixc==203) call rx0( ' OK! hx0fp0 mode=203 sergeyv eps LFC ')
02199      if(ixc==222) call rx0( ' OK! hx0fp0 mode=222 chi+- NoLFC sergeyv')
02200      if(ixc==223) call rx0( ' OK! hx0fp0 mode=223 chi+- LFC sergeyv')
02201      end
02202
02203 c-----
02204      real*8 function eclda_bh(rs)
02205      real(8) :: rs,cp,rp,z
02206      cp = 0.0504d0*0.5d0 ! 0.5 changes unit from Ry to Hartree
02207      rp = 30.d0
02208      z = rs / rp
02209      eclda_bh = -cp * ( (1.d0+z**3)*dlog(1.d0+1.d0/z)
02210      + 0.5d0*z - z**2 - 0.33333333d0 )
02211      end
02212 c-----
02213      real*8 function eclda_pz(rs)
02214      real(8) :: rs
02215      if (rs.ge.1.d0) then
02216          eclda_pz = -0.1423d0 / (1.d0 + 1.0529d0*dsqrt(rs) + 0.334d0*rs)
02217      else
02218          eclda_pz = -0.0480d0 + 0.0311d0*dlog(rs) - 0.0116d0 * rs
02219          + 0.0020d0*rs*dlog(rs)
02220      endif
02221      end
02222 c-----
02223      subroutine wecqw(ifcor,
02224      d      nqibz,nqibz,nq0i,nqitot,niw,
02225      o      wibz,wqt,wx,freqx,ecqw)
02226
02227      implicit double precision (a-h,o-z)
02228      dimension wibz(nqibz),wqt(nq0i),wx(niw),
02229      freqx(niw),ecqw(nqitot,niw)
02230      real(8):: rydberg
02231      write(ifcor,*)'### ecqw(q,w) ###'
02232      write(ifcor,*)'nqibz =',nqibz
02233      write(ifcor,*)'nq0i =',nq0i
02234      write(ifcor,*)'niw =',niw
02235      do ip = 2,nqitot
02236          if (ip <= nqibz) then
02237              wk = wibz(ip)*0.5d0 ! 0.5 for the normalization of wibz
02238          else
02239              wk = wqt(ip-nqibz)*wibz(1)*0.5d0 ! 0.5 for the normalization of wibz
02240              wk = wqt(ip-nqibz)* 1d0/dble(nqibz)
02241          endif
02242          write(ifcor,*)'### iq,wq = ',ip,wk
02243          sume=0d0
02244          do ix = 1,niw
02245              write(ifcor,*)freqx(ix),ecqw(ip,ix),wx(ix)
02246              sume=sume+ wx(ix)/(freqx(ix)*freqx(ix)) * ecqw(ip,ix)
02247          enddo
02248          write(ifcor,*) ' sum ecqw*wx=', wk*sume*2d0*rydberg()

```

```

02249 ! end of ip-loop
02250     enddo
02251     return
02252     end
02253 c-----
02254 subroutine getsqovlp(q,ngc,ngb,sqovlp)
02255 !! == Get sqrt of ppovl ==
02256     implicit none
02257     real(8)::q(3)
02258     integer:: ngc,ngb,nbloch,i,nmxx,ix,iy,nev
02259     complex(8):: sqovlp(ngb,ngb)
02260     complex(8),allocatable:: ooo(:,:),ppo(:,:),sqovlpi(:,:),ppovl(:,:)
02261     complex(8),allocatable:: ovlp(:,:),evec(:,:)
02262     real(8),allocatable:: eval(:)
02263     nbloch = ngb-ngc
02264     if(ngc==0) goto 888
02265
02266     allocate(ppovl(1:ngc,1:ngc))
02267     call readppovl0(q,ngc,ppovl)
02268     allocate(ooo(ngc,ngc),ppo(ngc,ngc),evec(ngc,ngc),eval(ngc))
02269     ooo= 0d0
02270     do ix=1,ngc
02271         ooo(ix,ix)=1d0
02272     enddo
02273     ppo = ppovl
02274     deallocate(ppovl)
02275     nmxx = ngc
02276     evec = 0d0
02277     eval = 0d0
02278     call diagcv(ooo, ppo,
02279 &         evec, ngc, eval, nmxx, 1d99, nev)
02280     write(6,*)' diagcv overlap ngc nev=',ngc,nev
02281     deallocate(ooo,ppo)
02282 c
02283 888 continue
02284     sqovlp=0d0
02285     do i=1,nbloch
02286         sqovlp(i,i)=1d0
02287     enddo
02288     do i=1,ngc
02289         if(eval(i)<0d0) then
02290             call rx('getsqovlp: eval(i) <0d0')
02291         endif
02292         do ix=1,ngc; do iy=1,ngc
02293             sqovlp(ix+nbloch,iy+nbloch)=
02294 &         sqovlp(ix+nbloch,iy+nbloch)
02295 &         + evec(ix,i)* sqrt(eval(i))* dconjg(evec(iy,i))
02296         enddo ; enddo
02297     enddo
02298     if(allocated(evec)) deallocate(evec)
02299     if(allocated(eval)) deallocate(eval)
02300     write(6,*)' end of getsqovlp'
02301 c     sqovlpi = sqovlp
02302 c     call matcinv(ngb,sqovlp) ! inverse
02303 c     ovlp=ovlp
02304 c     deallocate(ppovl,ovlp)
02305     end
02306
02307 c-----
02308 subroutine tr_chkwrite(tagname,zw,iw,freqq,nblochpmx,nbloch,ngb,iq)
02309 !! == check write for zw, no output == !!
02310     implicit none
02311     integer:: nblochpmx,nbloch,ngb,iw,i,iq
02312     complex(8):: zw(nblochpmx,nblochpmx),trwv,trwv2
02313     real(8):: freqq
02314 c     logical:: smbasis
02315     character*(*)::tagname
02316     trwv=0d0
02317 c     if(.not.smbasis()) then
02318 c         do i = 1,nbloch
02319 c             trwv = trwv + zw(i,i)
02320 c         enddo
02321 c     endif
02322     trwv2 = 0d0
02323     do i = 1,ngb
02324         trwv2 = trwv2 + zw(i,i)
02325     enddo ! write(6,(' realomg trwv=" ,2i6,4d22.14')' ) iq,iw,trwv(iw),trwv2(iw)
02326     write(6,(' a,f10.6,2i5,4d20.12')' )trim(adjustl(tagname)),freqq,iq,iw,trwv,trwv2
02327 c     do i = 1,ngb
02328 c         write(6,(' "iii i=" ,i4,a,f10.4,2i5,4d22.14')' )i,tagname,freqq,iq,iw,zw(i,i)
02329 c     enddo
02330     end
02331
02332 c-----
02333 c subroutine test_xxx(tagname,zw,iw,freqq,nblochpmx,nbloch,ngb,iq)
02334 c     implicit none
02335 c     integer:: nblochpmx,nbloch,ngb,iw,i,iq

```

```

02336 c      complex(8):: zw(nblochpmx,nblochpmx),trwv,trwv2
02337 c      real(8):: freqq
02338 c      logical :: smbasis
02339 c      character*(*)::tagname
02340 c      trwv2 = 0d0
02341 c      forall( i = 1:ngb)
02342 c          trwv2 = trwv2 + zw(i,i)
02343 c      end forall
02344 c      end
02345 c-----
02346
02347 function matcinvf(a) result(b)
02348 !!== Test routine for Inversion ==
02349 implicit none
02350 integer :: info,n,n2(2)
02351 integer,allocatable :: ipiv(:)
02352 complex(8):: a(:,,:), b(1)
02353 complex(8),allocatable:: work(:)
02354 n2= shape(a)
02355 n=n2(1)
02356 call zcopy(n,b,1,a,1)
02357 call zgetrf(n,n,a,n,ipiv,info)
02358 if(info/=0) then
02359     write(6,*)' matcinv: zegtrf info=',info
02360     call rx( ' matcinv: zegtrf ' )
02361 endif
02362 allocate(work(n*n))
02363 call zgetri(n,a,n,ipiv,work,n*n,info)
02364 deallocate(work)
02365 if(info/=0) then
02366     write(6,*)' matcinv: zegtri info=',info
02367     call rx( ' matcinv: zegtri ' )
02368 endif
02369 end
02370
02371 c-----
02372 subroutine diagno00(nbloch,wpvc,eval)
02373 !! == obtain eigenvalue only for input complex matrix wpvc(nbloch,nbloch)
02374 implicit none
02375 integer:: nbloch,nmx,nev,i
02376 complex(8),allocatable:: ovlpc(:,,:),evecc(:,,:),wpvcc(:,,:)
02377 real(8)::emx,eval(nbloch)
02378 complex(8):: wpvc(nbloch,nbloch)
02379 allocate( ovlpc(nbloch,nbloch),evecc(nbloch,nbloch),wpvcc(nbloch,nbloch))
02380 wpvcc= wpvc
02381 ovlpc= 0d0
02382 do i=1,nbloch
02383     ovlpc(i,i)=1d0
02384 enddo
02385 eval=0d0
02386 nev = nbloch
02387 nmx = nbloch
02388 call diagcv(ovlpc,wpvcc, evecc, nbloch, eval, nmx, 1d99, nev)
02389 deallocate(ovlpc,evecc,wpvcc)
02390 end
02391

```

## 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 1343 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 following 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
00008 c      use m_readeps,only: read_eps, epsinv, w_mu, llmat2=>llmat,deallocate_eps
00009 !!
00010
00011      use m_readfermi,only: readfermi,ef
00012      use m_readqg,only: readngmx,readqg
00013      use m_readeigen,only: init_readeigen,init_readeigen2,readeval
00014      use m_read_bzdata,only: read_bzdata, !<--- 'call read_bzdata' sets up following data.
00015      & ngrp2=>ngrp,nqbz,nqibz,n1,n2,n3,qbas,ginv,
00016      & dq_,qbz,wbz,qibz,wibz,
00017      & ntetf,idtetf,iblbz, qbwz,nqbwz !for tetrahedron
00018 c      & idteti, nstar,irk,nstbz
00019      use m_genallcf_v3,only: genallcf_v3,
00020      & nclass,natom,nspin,nl,nn,ngrp,
00021      & nlmtol,nlnmx, nctot,niw, !nw_input=>nw,
00022      & alat, delta,deltaw,esmr,symgrp,clabl,iclass, !diw,dw,
00023      & invg, il, in, im, nlnm,
00024      & plat, pos, ecore, symgg
00025
00026      use m_keyvalue,only: getkeyvalue
00027      use m_pbindx,only: pbindx !,norbt,l_tbl,k_tbl,ibas_tbl,offset_tbl,offset_rev_tbl
00028      use m_readqgcou,only: readqgcou
00029
00030 !! Base data to generate matrix elements zmel*. Used in "call get_zmelt".
00031      use m_rdpp,only: rdpp, !"call rdpp" generate following data.
00032      & nblocha,lx,nx,ppbrd,mdimx,nbloch,cgr
00033 !! Generate matrix element for "call get_zmelt".
00034      use m_zmel,only: !these data set are stored in this module, and used when
00035      & nband,itq,ngcmx,ngpmx, ppovlz,
00036      & ppbir,shtvg, miat,tiat , ntq
00037 !! frequency
00038      use m_freq,only: getfreq,
00039      & frhis,freq_r,freq_i, nwhis,nw_i,nw,npm !output of getfreq
00040 !! antiferro
00041 c      use m_anf,only: anfcond,
00042 c      & laf,ibasf !,ldima,pos,natom
00043 !! tetwt
00044      use m_tetwt,only: tetdeallocate,gettetwt, !followings are output of
00045      'L871:call gettetwt')
00046      & whw,ihw,nhw,jhw,ibjb,nbnbx,nhwtot,n1b,n2b,nbnb
00047 !! w0 and w0i (head part at Gamma point)
00048      use m_w0w0i,only: w0w0i,
00049      & w0,w0i,llmat
00050
00051 !! MPI
00052      use m_mpi,only: mpi_hx0fp0_rankdivider2q,mpi_hx0fp0_rankdivider2s,
00053      & mpi_qtask,mpi_initializegspbm,mpi_finalize,mpi_root,
00054      & mpi_broadcast,mpi_dblecomplexsendq,mpi_dblecomplexrecvq,mpi_rank,mpi_size,
00055      & mpi_grantab,mpi_consoleout,mpi_ss,mpi_se, mpi_allreducesums,
00056      & mpi_barrier, mpi_rankq,mpi_rootq,mpi_roots
00057 !! q0p
00058      use m_readq0p,only: readq0p,
00059      & wqt,q0i,nq0i ,nq0iadd,ixyz
00060
00061      implicit none
00062      integer,allocatable:: nwgt(:, :)
00063      integer:: iopen,maxocc2,iclose, ixc,ixxini,ixxend,
00064      & ifhbe, nprecb,mrecb,mrece,nlmtot,nqbtz,!nband,
00065      & i,nq0ix,ngrpmx,mxx,nqbze,nqibze,ini,ix,ngrpx !ngcmx,
00066      & nblochpmx,ndummy1,ndummy2,ifcphi,is,nwp, !ifvcfpout,,mdimx,nbloch
00067      & ifepscond,nxx,ifvxcput,ifgb0vec
00068      & ,nw0,iw,ifinin,iw0,noccxv,noccx
00069      & ,nprecx,mrecl,ifwd,ifrcwi,ifrcw,nspinmx,ifianf,ibas
```

```

00069      &      ,ibas1,irot,iq,ngb,ixq2,ifepsdatnolfc,ifepsdat,ngbin,igc0
00070      &      ,kx,isf,kqxx,kp,job,nwmax !,ifev1,ifev2 !,nhwtot
00071      &      ,ihis,ik,ibib,ib1,ib2,ichkhis,ihww,j,imode
00072      &      , ifchipmlog , nw_w,nwmin !,ngpmx
00073      real(8):: dum1,dum2,dum3,wqtsum,epsrng,dnorm, dwry,dwh,omg2, q(3), qgbin(3),qx(3)
00074      real(8):: ua=ld0 ! this is a dummy.
00075      integer:: ifrb(2),ifcb(2),ifrbh(2),ifchb(2), ndble=8, nword
00076      integer,allocatable :: ngveccb(:,:), iqib(:),ifppb(:) !,lx(:) ngvecc(:,:),
00077      complex(8),allocatable:: geigb(:,:,:), geig(:,:),vcoul(:,:),
00078      &      zw(:,:),zw0(:,:), zxq(:,:),zxqi(:,:),
00079      real(8),allocatable :: eqt(:), !ppbrd (:,:,:),cgr(:,:,:)
00080      &      ppbrdx(:,:,:),aaa(:,:),symope(:,:),
00081      &      ppb(:,:),pdb(:,:),dpb(:,:),ddb(:,:), qbze(:,:),qibze(:,:) !,ecore(:,:)
00082 c      &      freqr(:),freqi(:) !rw(:,:),cw(:,:) --->zw
00083      complex(8),allocatable :: rcxq(:,:,:),
00084      complex(8) :: fff,img=(0d0,ld0)
00085      complex(8),allocatable :: wwkw(:,:),
00086      real(8) :: qbxz(3)
00087      logical :: debug
00088 c      integer,allocatable:: ibasf(:)
00089      logical :: realomega, imagomega
00090      complex(8),allocatable:: zxr(:,:),ppovl(:,:),ppovlzinv(:,:) !,ppovlz(:,:)
00091      complex(8) :: epxxx,vcmean
00092      character*9 fileps
00093      character*15 filepsnolfc
00094 c      logical :: paralellx0=.true. !, hist
00095      character(5) :: charnum5
00096      character(20):: xxt
00097      real(8) :: emin, emax ,emax2,emin2
00098 c      integer :: iSigma_en !sf..21May02 !iSigma_en is integer
00099      !parameter stored in GWIN_V2
00100      !which determines approximation for self-energy.
00101      !Self-energy should be made hermitian for energies to be real
00102 cxxx !iSigma_en==0 SE_nn'(ef)+img integral:delta_nn'([SE_nn(e_n)+c.c.]/2-SE_nn(ef))
00103 cxxx !iSigma_en==1 SE_nn'(ef)+delta_nn'([SE_nn(e_n)+c.c.]/2-SE_nn(ef))
00104      !iSigma_en==2 [SE_nn'((e_n+e_n')/2)+h.c.]/2
00105      !iSigma_en==3 [(SE_nn'(e_n)+SE_nn'(e_n'))/2+h.c.]/2
00106      real(8) :: omg2max,omg1max,wemax
00107      logical::imagonly=.false. , noq0p !,readgwinput
00108      integer::nwin, incwfin, verbose,nbcut,nbcut2,ifpomat,nmmx,ikpo,nn_,noo,igxxx,nomx
00109 c      real(8)::efin
00110      logical :: noloco=.false.
00111      integer:: ispl,isp2, ngc,mrecg ! bzcuse,
00112      real(8):: quu(3),deltaq(3),qqq(3)=0d0 !
00113      complex(8),allocatable:: wgt(:,:),
00114      real(8),allocatable:: qbz2(:,:)
00115      logical :: qbzreg
00116 !      logical ::smbasis !smbasis will be implemented in m_zmel.f which generates <phi|phi M>
00117      real(8):: q_r(3)
00118      complex(8),allocatable:: pomat(:,:)
00119      logical :: timereversal,onceww
00120      integer :: jpm,ncc
00121      real(8) :: frr !, sciss
00122      integer :: ngb0,ifvcoud,idummy,igb1,igb2,ngb_in,nmbas1,nmbas2,iq0,ifisk,ixq,ig,nmbaslx !ifepstin,
00123      complex(8),allocatable:: zcousq(:,:),epstin(:,:),epstilde(:,:),zcousqsum(:,:),zcousq(:,:),eemat(
:,:),zcousq0(:,:)
00124      real(8),allocatable:: vcousq(:),vcousq0(:),vcoudummy(:)
00125      real(8):: fourpi,sqfourpi,tpioa,absq,vcoul,vcoulsq
00126 !! Eq.(40) in PRB81 125102
00127 c      complex(8),allocatable::sk(:,:),sks(:,:),skI(:,:),skI(:,:),
00128      &      w_k(:,:),w_ks(:,:),w_kI(:,:),w_ksI(:,:), llw(:,:), llwI(:,:),
00129      complex(8),allocatable::sk(:,:),sks(:,:),ski(:,:),sksi(:,:),
00130      &      w_k(:),w_ks(:),w_ki(:), w_ksi(:)
00131      complex(8),allocatable:: llw(:,:), llwi(:,:),aaamat(:,:)
00132      integer:: lxxklm,nlxxklm,ifidmlx,ifrcwx,iq0xx,ircw,nini,nend,iwxx,nw_ixxx,nwxxx,niwxxx,iwx,iccl,icc2
00133      complex(8):: vclvc2
00134      integer,allocatable:: neibz(:),ngrpt(:),igx(:,:),igxt(:,:),eibzsym(:,:),
00135      real(8),allocatable:: aik(:,:),
00136      integer,allocatable:: aiktimer(:,:)
00137      integer:: l2nl, nmbas_in , ixqendx,imb2 !iqqv,
00138      logical:: eibz4x0,tiii,iprintx,chipm=.false.,iqinit,localfieldcorrectionllw
00139      real(8)::qv(3),ecut,ecuts,hartree,rydberg,pi
00140      character(128):: vcoudfile
00141      integer :: iqeibz
00142      complex(8):: epslfc, axxx(10)
00143      integer:: src,dest
00144      integer:: ifw0w0i
00145      logical :: symmetrize,eibzmode
00146      real(8):: schi=-9999 !dummy
00147      integer:: i_reduction_npm, i_reduction_nwhis, i_reduction_nmbas2
00148      logical:: crpa
00149      integer,allocatable :: iclasst(:), invgx(:)
00150      integer:: ibasx,ificlass,ifile_handle,ifiq0p
00151      complex(8),allocatable:: ppovl(:,:)
00152      logical:: tetra ,readw0w0itest=.false.
00153      integer::nw_ixx,nwxx
00154

```

```

00155      logical:: w4pmode
00156      complex(8),allocatable:: wmu(:, :),wmuk(:, :)
00157      integer:: ifw4p,ngbq0,igb
00158      real(8):: qv(3,3)
00159
00160      real(8)::ebmx
00161      integer:: nbmx,mtet(3)
00162      real(8),allocatable:: ekxx1(:, :),ekxx2(:, :)
00163
00164 c      integer:: ifief
00165 c      real(8):: ef
00166 !-----
00167 !TIME0_1001 ProgAll
00168 !TIME0_11001 readbzddata
00169      call mpi__initializeqspbm()
00170      call mpi__consoleout('hx0fp0_sc')
00171      call cputid(0)
00172      allocate( zxr(1,1)) !dummy
00173      hartree= 2d0*rydberg()
00174      pi      = 4d0*atan(1d0)
00175      fourpi = 4d0*pi
00176      sqfourpi=sqrt(fourpi)
00177      write(6,*) ' --- hx0fp0_sc Choose omodes below -----'
00178      write(6,*) '   ixc= 11,10011,or 1011 '
00179      write(6,*) ' --- Put number above ! -----'
00180      if( mpi__root ) then
00181          read(5,*) ixc !c      call readin5(ixc,igxini,igxend)
00182      end if
00183      call mpi__broadcast(ixc)
00184      crpa=.false.
00185      if(ixc==0) call rx( ' --- ixc=0 --- Choose computational mode!' )
00186      call headver('hx0fp0_sc',ixc)
00187 c      call getkeyvalue("GWinput","ScaledGapX0",sciss,default=1d0)
00188 c      write(6,")(' ScaledGapX0=',f8.3)") sciss
00189      if(ixc==11) then
00190          write(6,*) " OK ixc=11 normal mode "
00191      elseif(ixc==10011) then
00192          write(6,*) " OK ixc=10011 crpa mode "
00193          crpa=.true.
00194      elseif(ixc==1011) then
00195          write(6,*) 'OK ixc=1011 Add W0W0I part at q=0'
00196      else
00197          write(6,*) 'we only allow ixc==11. given ixc=',ixc
00198          call rx( 'error:we only allow ixc==11.')
00199      endif
00200 !! newanis2 is now fixed to be .true.
00201      call getkeyvalue("GWinput","ecut_p",ecut, default=1d10 )
00202      call getkeyvalue("GWinput","ecuts_p",ecuts,default=1d10 )
00203 c      Prof.Nagara says this cause a stop in ifort --->why???
00204 c      write(6,*) 'Timereversal=',Timereversal()
00205
00206 !! Readin BZDATA. See m_read_bzdata in gwsrc/rwbzdata.f
00207      call read_bzdata()
00208
00209 !TIME1_11001 "readbzddata"
00210 !TIME0_12001 QOP
00211 !! Use regular mesh even for bzcase==2 and qbzreg()==T
00212 !!   off-regular mesh for bzcase==1 and qbzreg()==F
00213 c      if( ( bzcase()==2.and.qbzreg() ) .or.
00214 c      &      ( bzcase()==1.and.(.not.qbzreg()) ) ) then
00215 !! this mechanism for qbzreg=F is too complicated. We may need to modify defination of qbz for qbzreg=F.
00216      if(.not.qbzreg()) then ! set off-gamma mesh
00217          deltaq= qbas(:,1)/n1 + qbas(:,2)/n2 +qbas(:,3)/n3
00218          do i=1,nqbz
00219              qbz(:,i) = qbz(:,i) - deltaq/2d0
00220              write(6,")(' i qbz=',i3,f8.4)") i,qbz(:,i)
00221          enddo
00222      endif
00223      write(6,")(' nqbz nqibz ngrp=',3i5)") nqbz,nqibz,ngrp
00224 !! == Readin by genallcf ==
00225 !! See "use m_genallcf_v3" at the begining of this routine
00226 !! We set basic data.
00227 c      nwin = 0      !Readin nw from NW file
00228      incwfin= 0      !use ForX0 for core in GWIN
00229 c      efin = 0d0      !readin EFERMI
00230 c--- EFERMI
00231      call readefermi()
00232      call genallcf_v3(incwfin) !in module m_genallcf_v3
00233      if(ngrp/= ngrp2) call rx( 'ngrp inconsistent: BZDATA and LMTO GWIN_V2')
00234 c      nw_input = nw ;
00235 c      write(6,*) 'nw delta=',nw_input,delta
00236      debug=.false.
00237      if(verbose()>=100) debug=.true.
00238      if(debug) write(6,*) ' end of genallc'
00239      tpioa=2d0*pi/alat
00240 !!!! WE ASSUME iclass(iatom)= iatom, nclass = natom. !!!!!!!!!!!!!!!!!!!!!!!
00241      if(nclass /= natom) call rx( ' hx0fp0_sc: nclass /= natom ')

```



```

00242 !! --- tetra or not
00243 c      if(delta <= 0d0) then
00244     tetra = .true.
00245     delta = -delta
00246     write(6,*)' hx0fp0.sc: tetrahedron mode delta=',delta
00247 c      else
00248     tetra = .false. ! switch for tetrahedron method for dielectric functions
00249 c      endif
00250 !! --- read dimensions of h,hb
00251     ifhbe = iopen('hbe.d',1,0,0)
00252     read (ifhbe,*) nprecb,mrecb,mrece,nlmtot,nqbzt,nband,mrecg
00253     if(nlmt0/=nlmtot) call rx( ' hx0fp0: nlmt0/=nlmtot in hbe.d')
00254     if(nqbz /=nqbzt ) call rx( ' hx0fp0: nqbz /=nqbzt in hbe.d')
00255
00256 !! --- Readin Offset Gamma -----
00257     call readq0p()
00258     write(6,('(### nqibz nq0i nq0iadd=', 3i5))nqibz,nq0i,nq0iadd
00259
00260 c$$$      if(.not.newaniso2) then
00261 c$$$          wqtsum = sum(abs(wqt(1:nq0i)))
00262 c$$$          call getkeyvalue("GWinput","TestNoQ0P",noq0p,default=.false.)
00263 c$$$      endif
00264
00265 !TIME1_12001 "Q0P"
00266 !TIME0_13001 mptauof
00267     call getsrdpp2(nclass,nl,nxx)
00268     call readngmx('QGpsi',ngpmx)
00269     call readngmx('QGcou',ngcmx)
00270     write(6,*)' ngcmx ngpmx=',ngcmx,ngpmx
00271     nqbze = nqbz *(1 + nq0i+nq0iadd)
00272     nqibze = nqibz + nq0i+nq0iadd
00273     allocate( qbze(3, nqbze), qibze(3, nqibze))
00274     call dcopy(3*nqbz, qbze, 1, qbze,1)
00275     call dcopy(3*nqibz,qibz, 1, qibze,1)
00276     do i = 1,nq0i+nq0iadd
00277         qbze(:,nqibz+i) = q0i(:,i)
00278         ini = nqbz*(1 + i -1)
00279         do ix=1,nqbz
00280             qbze(:,ini+ix) = q0i(:,i) + qbze(:,ix)
00281         enddo
00282     enddo
00283 !! ----- dummy ngrpx=1 -----
00284     ngrpx = 1
00285     l2nl=2*(nl-1)
00286     allocate(symope(3,3))
00287     symope(1:3,1) = (/1d0,0d0,0d0/)
00288     symope(1:3,2) = (/0d0,1d0,0d0/)
00289     symope(1:3,3) = (/0d0,0d0,1d0/)
00290 !! dummy. Get space-group transformation information. See header of mptauof.
00291     ificlass=ifile_handle()
00292     open (ificlass,file='CLASS')
00293     allocate(iclassst(natom),invvx(ngrp))
00294     & ,miat(natom,ngrp),tiat(3,natom,ngrp),shvtg(3,ngrp))
00295     write(6,*)' --- Readingin CLASS info ---'
00296     do ibas = 1,natom
00297         read(ificlass,*) ibasx, iclasst(ibas)
00298         write(6, "(2i10)") ibasx, iclasst(ibas)
00299     enddo
00300     close(ificlass)
00301     call mptauof(symope,ngrp,plat,natom,pos,iclassst
00302     o ,miat,tiat,invvx,shvtg ) !note: miat,tiat,shvtg are defined in m_zmel.
00303     if(verbose())>=40) write (*,*)' hsf0.sc.m.F: end of mptauof'
00304 !! call rdpp gives pbrd = radial integrals and cgr = rotated cg coeffecients.
00305     call rdpp(nxx, nl, ngrpx, nn, nclass, nspin, symope,qbas)
00306     ntq=nband
00307     allocate(itq(ntq))
00308     do i=1,ntq
00309         itq(i)=i
00310     enddo
00311 !! Pointer to optimal product basis
00312 c      allocate(imdim(natom))
00313 c      call indxmdm (nblocha,nclass,iclass,natom,
00314 c      o imdim ) !in m_zmel
00315     nblochpmx = nbloch + ngcmx
00316     allocate(ngvecb(3,ngcmx))
00317     iqpend = nqibz + nq0i + nq0iadd
00318     write(6,*)' nqibz nqibze=',nqibz,nqibze
00319 !TIME1_13001 "mptauof"
00320 !TIME0_14001 init_readeigen
00321 !!... initialization of readEigen !readin m_hamindex
00322     call init_readeigen(ginv,nspin,nband,mrece)!EVU EVD are read in init_readeigen
00323     call init_readeigen2(mrecb,nlmt0,mrecg)
00324 c      --- ecore ---
00325 c      allocate(ecore(nctot,nspin)) !core energies
00326 c      do is = 1,nspin
00327 c      if (nctot.gt. 0) then
00328 c      call catch1 (w(iecore),is,nctot,2,ecore(:,is)) !core energies

```



```

00329 c      write(6,*)' ecore is=',is,ecore(:,is)
00330 c      endif
00331 c      enddo
00332
00333 c      --- set realomega, imagomega tetra nw niw nwp ifgb0vec -----
00334 !      nwp, freq_r, frhis(1:nwhis+1)
00335 c      if ( ixc==1 ) then !old imagw = 2 case
00336 c      realomega =.true.
00337 c      imagomega =.true.
00338 c      stop 'hsfp0sc: ixc==1 is not implimented'
00339 ccccccccccccccccccccfaleev 21May02, use only ixc=1,11 modes ccccccccccc
00340 c      elseif( ixc==2.or.ixc==3 ) then
00341 c      realomega =.true.
00342 c      imagomega =.false.
00343 c      niw = 0
00344 c      ifepscond = 2102
00345 c      open (ifepscond,file='EPScond')
00346 c      read (ifepscond,*) epsrng, dwry !epsrng dw in Ry
00347 c      dw = dwry/2d0
00348 c      close(ifepscond)
00349 c      if(dw==0d0) then
00350 c      nw = 1
00351 c      else
00352 c      nw = (epsrng/2d0 - 1d-10)/(dw/2d0) + 2 !epsrng/2d0 corresponds to in a.u.
00353 c      endif
00354 c      allocate(epsin(nw,neps))
00355 c      elseif( ixc==4.or.ixc==5.or.ixc==6) then
00356 c      ! ... These are test modes.
00357 c      ! ixc=4 tetrahedron weight test. tetwt5.vs.tetwt5. Write tethis.chk
00358 c      ! ixc=5 Spectrum function (Img part) along the Real axis with tetwt4
00359 c      ! ixc=6 Spectrum function (Img part) along the Real axis with tetwt5. Histogram method.
00360 c      realomega = .true.
00361 c      imagomega = .false.
00362 c      tetra = .true.
00363 c      niw = 0
00364 c      ! --- For tetwt5 --- the tetrahedron weight for spectrum function (imaginary part)
00365 c      ! Histogram bins are specified by freq_r(1:nwp)
00366 c      ! nwp=nw+1; frhis(1)=0
00367 c      ! The 1st bin is [frhis(1), frhis(2)] ...
00368 c      ! The last bin is [frhis(nw), frhis(nwp)].
00369 c
00370 c      ! ... These parameters specifies a test histogram bins;Sergey's mesh just for test modes.
00371 c      nw0 = 200 !100 800
00372 c      dwh = 0.01d0 !0.02d0 0.0025d0 !in hartree
00373 c      ! ...
00374 c      call findemaxmin(ifev,nband,nqbz,nspin,emax,emin)
00375 c      if (nctot .gt. 0) Emin = minval(ecore)
00376 c      omg2max = (Emax-Emin)*.5+.2d0 !(in Hartree) covers all relevant omega, +.2 for margin
00377 c      omg1max = dwh*(nw0-1)
00378 c      nwp = int(sqrt(omg2max*(2*nw0-1d0)/dwh-(nw0**2-3*nw0+1d0)))+1 ! + 1 for margin
00379 c      nw = nwp-1
00380 c      write(6,*) Emax,Emin,nw0,nw ! nwp is new max number in frequency array
00381 c      write(6, '(a32,2i7,2d15.3)') 'hx0fp1: nw0,nw,omg1max,omg2max='
00382 c      & , nw0,nw, omg1max,omg2max
00383 c      if (nw <= nw0) stop 'hx0fp0:ixc==[456] nw2 <= nw'
00384 c      allocate(freq_r(nwp))
00385 c      do iw=1,nwp !This is a test mesh by Sergey.Faleev
00386 c      if(iw<nw0) then; freq_r(iw)=dwh*(iw-1)
00387 c      else; freq_r(iw)=dwh*(iw**2+nw0**2-3*nw0+1)/(2*nw0-1d0)
00388 c      endif
00389 c      enddo !freq_r(iw) is linear for iw<nw and quadratic for nw<=nw2
00390 c      !freq_r(iw) chosen in such a way that it is continues with
00391 c!!! nw nwp=nw+1 freq_r(1:nwp) are used after here.
00392 c      allocate(frhis(nwp))
00393 c      frhis=freq_r(1:nwp)
00394 c      nwhis=nw
00395
00396 !! We get frhis,freq_r,freq_i, nwhis,nw,npm by getfreq
00397 c      realomega = .true.
00398 c      imagomega = .true.
00399 c      tetra = .true.
00400 c      call findemaxmin(nband,qbze,nqbz,nspin, emax,emin)
00401 c      if(.not.qbzreg()) then
00402 c      allocate(qbz2(3,nqbz))
00403 c      do iq=1,nqbz
00404 c      qb2(:,iq)=qbz(:,iq)+dq_
00405 c      enddo
00406 c      call findemaxmin(nband,qbz2,nqbz,nspin ,emax2,emin2)
00407 c      emax=max(emax,emax2)
00408 c      emin=min(emin,emin2)
00409 c      deallocate(qbz2)
00410 c      endif
00411 c      if (nctot > 0) emin=minval(ecore(:,1:nspin))
00412 c      omg2max = (emax-emin)*.5d0+.2d0
00413 c      ! (in Hartree) covers all relevant omega, +.2 for margin
00414 c      if(mpi_root) write(6, '(' emin emax omg2max=',3f13.5)') emin, emax, omg2max
00415 c      call getwemax(.true.,wemax) !wemax is to determine nw !real axis divisions

```

```

00416      if(mpi__root) write(6,"(' wemax= ',f13.4)") wemax
00417      call getfreq(.false.,realomega,imagomega,tetra,omg2max,wemax,niw,ua,mpi__root)
00418      nwp = nw+1
00419
00420      !! We first accumulate Imaginary parts. Then do K-K transformation to get real part.
00421      noccxv = maxocc2(nspin,ef, nband, qbz,ngbze)
00422      !max no. of occupied valence states
00423      if(noccxv>nband) call rx( 'hx0fp0_sc: all the bands filled! too large Ef')
00424      noccx = noccxv + nctot
00425      nprecx = ndble      !We use double precision arrays only.
00426      mrecl = nprecx*2*nblochpmx*nblochpmx/nword()
00427      if(mpi__root)then
00428          ifwd = iopen('WV.d',1,-1,0)
00429          write (ifwd,"(1x,10i14)") !(1x,i3,i8,i5,5i4")
00430      & nprecx,mrecl,nblochpmx,nwp,niw,nqibz + nq0i-1,nw_i
00431          ifwd = iclose('WV.d'); ifwd=0
00432      endif
00433      allocate( zw(nblochpmx,nblochpmx) )
00434      nspinmx = nspin
00435      !TIME1_14001 "init_readeigen"
00436      !TIME0_15001 ppbafp_v2
00437
00438      !!... these are used x0k
00439      call getkeyvalue("GWinput","nbcutlow",nbcut, default=0 )
00440      call getkeyvalue("GWinput","nbcutlowto",nbcut2, default=0 )
00441      write(6,"(' nbcut nbcutlowto=',2i5)") nbcut,nbcut2
00442      !! -- ppb= <Phi(SLn,r) Phi(SL'n',r) B(S,i,Rr)>
00443      !! This is general for rotated CG coefficient; but hx0fp0 mode is only for ngrpx=1 (not rotated).
00444      !! Compare usage in hsfp0 modes.
00445      irot=1
00446      allocate( ppbir(nlnmx*nlnmx*mdimx*nclass,irot,nspin))
00447      do is = 1,nspin
00448          call ppbafp_v2(irot,ngrp, is,nspin,
00449              i il,in,im,nlnm,      !w(i_mnl),
00450              i nl,n,nclass,nlnmx,
00451              i mdimx,lx,nx,nxx,      !Bloch wave
00452              i cgr, nl-1,      !rotated CG
00453              i ppbrd,      !radial integrals
00454              o ppbir(:,irot,is))      !this is in m_zmel
00455      enddo
00456      if(debug) write(6,*) ' end of ppbafp_v2'
00457      !TIME1_15001 "ppbafp_v2"
00458      !TIME0_16001 readqgcou
00459      call getkeyvalue("GWinput","nbcutlow",nbcut, default=0 )
00460      call getkeyvalue("GWinput","nbcutlowto",nbcut2, default=0 )
00461      write(6,"(' nbcut nbcutlowto=',2i5)") nbcut,nbcut2
00462      ixini=1 !for newaniso
00463      eibzmode = eibz4x0()
00464
00465      !! nov2016 moved from tetwt5 --> here
00466      call getkeyvalue("GWinput","nband_chi0",nbmx, default=nband )
00467      call getkeyvalue("GWinput","emax_chi0", ebm, default=ld10 )
00468      mtet=(/1,1,1/)
00469      call getkeyvalue("GWinput","multitet",mtet,3,default=(/1,1,1/))
00470      ! multitet=T ==> micro tetrahedron method (divided-tetrahedron). Not used so much now...
00471      allocate(ekxx1(nband,nqbz),ekxx2(nband,nqbz))
00472
00473      !! === Use of symmetry. EIBZ procedure PRB81,125102 ===
00474      !! For rotation of zcousq. See readeigen.F rotwv.F ppbafp.fal.F(for index of product basis).
00475      if(eibzmode) then
00476          !! commentout block inversion Use ixqendx=ixqend because of full inversion
00477          ixqendx=ixqend
00478          allocate( nwgt(nqbz,ixqini:ixqendx), !qeibz(3,nqbz,ixqini:nqibz),neibz(ixqini:nqibz),
00479              & igx(ngrp*2,nqbz,ixqini:ixqendx),igxt(ngrp*2,nqbz,ixqini:ixqendx),
00480              & eibzsym(ngrp,-1:1,ixqini:ixqendx))
00481          iprintx=.false.
00482
00483          write(6,*)
00484          write(6,"('=== Goto eibzgen === TimeRevesal switch =',11)")timereversal()
00485          if(mpi__root) iprintx=.true.
00486          call eibzgen(nqibz,symgg,ngrp,qibze(:,ixqini:ixqend),ixqini,ixqendx,qbz,nqbz,
00487              i timereversal(),ginv,iprintx,
00488              o nwgt,igx,igxt,eibzsym,tiii)
00489          write(6,"('Used timeRevesal for EIBZ = ',11)") tiii
00490          call cputid(0)
00491      c$$$
00492      c$$$      write(6,"('TimeRevesal switch = ',11)") timereversal()
00493      c$$$      call
00494          eibzgen(nqibz,symgg,ngrp,qibze(:,ixqini:ixqend),ixqini,ixqendx,qbz,nqbz,timereversal(),ginv,iprintx,
00495              c$$$      o nwgt,igx,igxt,eibzsym)
00496      c$$$!! Check timereversal is required for symmetrization operation or not. If not tiii=timereversal=F is
00497          used.
00498      c$$$!! this is because the symmetrization is a little time-consuming.
00499      c$$$      tiii=timereversal()
00500      c$$$      if(minval(igxt)==1) tiii=.false.
00501      c$$$      iprintx=.true.
00502      c$$$cccccccccccccccccccccccc

```

```

00501 c$$$c      tiii=.true.
00502 c$$$$cccccccccccccccccc
00503 c$$$      write(6, "('=== goto eibzgen === used timereversal=',l1)")tiii
00504 c$$$      call
           eibzgen(nqibz,symgg,nggrp,qibze(:,iqxini:ixqend),iqxini,ixqendx,qbz,nqbz,tiii,ginv,iprintx,
00505 c$$$      o      nwgt,igx,igxt,eibzsym)
00506
00507 !All input. this returns required index stored in arrays in m_pbindex.
00508      call pbindex(natom,lx,l2nl,nx)
00509      ! PBindex: index for product basis. We will unify this system; still similar is
           used in ppbafp_v2.
00510      call readqgcou() ! no input. Read QGcou and store data into variables.
00511 !! call Spacegroupprot(symgg,nggrp,plat,natom,pos) ! all inputs.
00512      else !dummy allocation to overlaid -check bound !sep2014
00513          ixqendx=ixqend
00514          allocate( nwgt(1,ixqini:ixqendx),igx(1,1,ixqini:ixqendx)
00515      &      ,igxt(1,1,ixqini:ixqendx), eibzsym(1,1,ixqini:ixqendx)) !dummy
00516      endif
00517
00518      allocate( llw(nw_i:nw,nq0i), llwi(niw,nq0i) )
00519      llw=1d99
00520      llwi=1d99
00521      if(ixc==1011)then !ixc==11 is a debug mode to test contrib. at \Gamma point.
00522          goto 1191
00523      endif
00524 !! for w4phonon. all nodes have wmu array.
00525      w4pmode=.false.
00526      if(sum(ixyz)/=0) w4pmode=.true.
00527      if(w4pmode) then
00528          allocate( wmu(2:nblochpmx,3))
00529          wmu=1d99
00530      endif
00531
00532 !! rank divider
00533      call mpi_hx0fp0_rankdivider2q(iqxini,ixqend)
00534      call mpi_hx0fp0_rankdivider2s(nspinmx)
00535
00536 !! == Calculate x0(q,iw) and W == main loop 1001 for iq.
00537 !! NOTE: iq=1 (q=0,0,0) write 'EPS0inv', which is used for iq>nqibz for ixc=11 mode
00538 !! Thus it is necessary to do iq=1 in advance to performom iq >nqibz.
00539 !! (or need to modify do 1001 loop).
00540 !! -----
00541 !! === do 1001 loop over iq =====
00542 !! -----
           iqinit=.true.
00543      write(6,('irank=',i5," allocated(MPI__qtask)=",L5'))mpi__rank,allocated(mpi__qtask)
00544      do iq = ixqini,ixqend
00545          if (mpi__qtask(iq)) write(6,('irank iq=',i5,i5')) mpi__rank,iq
00546      enddo
00547
00548
00549 !! Get ngbq0 (for q=0) and broadcast for w4p
00550      if( mpi__root.and. w4pmode ) then
00551          q = (/0d0,0d0,0d0/)
00552          call readqg('QGcou', q, ginv, quu,ngc,ngvecb)
00553          ngbq0 = nbloch+ngc
00554      endif
00555      call mpi__broadcast(ngbq0)
00556
00557 !TIME1_16001 "readqgcou"
00558 !TIME0_170001 do1001
00559      do 1001 iq = ixqini,ixqend
00560          if( .not. mpi__qtask(iq) ) cycle
00561          if (mpi__roots) then
00562              ifrcwi = iopen('WVI.'//charnum5(iq),0,-1,mrecl)
00563              ifrcw = iopen('WVR.'//charnum5(iq),0,-1,mrecl)
00564          endif
00565 !!
00566          call cputid(0)
00567          q = qibze(:,iq)
00568          call readqg('QGcou', q, ginv, quu,ngc,ngvecb) ! q was qq
00569
00570 !! Caution : confusing point
00571 !! ngc by QGcou is shown at the bottom of lqg4gw.
00572 !! ngc read from PPOVL are given by rdata4gw.
00573 !! Note that ngc(iq>nqibz)=ngc (q=0), because when it is generated in mkqg.F
00574 !!
00575 c      if( newaniso2.and.iq==1 ) then ! *sanity check
00576          if( iq==1 ) then ! *sanity check
00577              if(sum(q**2)>1d-10) call rx( ' hx0fp0.sc: sanity check. |q(iqx)| /= 0' )
00578          endif
00579
00580 !! ==== readin Coulomb matrix ====
00581          ngb = nbloch + ngc
00582          write(6,('do 1001: iq q=',i5,3f9.4))iq,q
00583          write(6,*)'nbloch ngb ngc=',nbloch,ngb,ngc
00584
00585 !! === readin diagonalized Coulomb interaction ===

```

```

00586 !! zcousq: E(\nu,I), given in PRB81,125102; vcousq: sqrt(v), as well.
00587 c      if(newaniso2) then
00588         vcoudfile='Vcoud.'//charnum5(iq) ! iq was iqqv this is closed at the end of do 1001
00589         ifvcoud = iopen(trim(vcoudfile),0,-1,0)
00590         read(ifvcoud) ngb0
00591         read(ifvcoud) qvv
00592         if(sum(abs(qvv-q))>1d-10) then
00593             write(6,*)'qvv =',qvv
00594             call rx( 'hx0fp0: qvv/=0 hvcc is not consistent')
00595         endif
00596         if(allocated(zcousq)) deallocate( zcousq,vcousq )
00597         allocate( zcousq(ngb0,ngb0),vcousq(ngb0))
00598         read(ifvcoud) vcousq
00599         read(ifvcoud) zcousq
00600         idummy=iclose(trim(vcoudfile))
00601         vcousq=sqrt(vcoussq)
00602
00603 c      if(newaniso2.and. iq>ngibz.and.(.not.localfieldcorrectionllw()) ) then
00604         if(iq>ngibz.and.(.not.localfieldcorrectionllw()) ) then
00605             if( ngb0/=ngb ) then
00606                 call rx( 'hx0fp0.m.f:ngb0/=ngb')
00607             endif
00608             nolfco =.true.
00609             nmbas_in = 1
00610 c         elseif(newaniso2) then !.and.iq==1) then
00611             else
00612                 nolfco = .false.
00613                 nmbas_in = ngb
00614             endif
00615             nmbas1 = nmbas_in
00616             nmbas2 = nmbas1
00617
00618 !! newaniso=T case. Used in get_zmelt in m_zmel called in x0kf_v4hz
00619         if(allocated(ppovlz)) deallocate(ppovlz)
00620         if(allocated(ppovlzinv)) deallocate(ppovlzinv)
00621         if(allocated(ppovl)) deallocate(ppovl)
00622         allocate(ppovl(ngc,ngc),ppovlz(ngb,ngb), ppovlzinv(ngb,ngb))
00623         call readppovl0(q,ngc,ppovl) !q was qq
00624 c         ppovlz(1:nbloch,:) = zcousq(1:nbloch,:)
00625 c         ppovlz(nbloch+1:nbloch+ngc,:)
00626 c         & = matmul(ppovl,zcousq(nbloch+1:nbloch+ngc,:))
00627         allocate(ppovl_(ngb,ngb))
00628         ppovl_=0d0
00629         do i=1,nbloch
00630             ppovl_(i,i)=1d0
00631         enddo
00632         ppovl_(nbloch+1:nbloch+ngc,nbloch+1:nbloch+ngc)=ppovl
00633         if(.not.eibz4x0()) then !sep2014 added for eibz4x0=F
00634             ppovl_ = matmul(ppovl_,zcousq)
00635         endif
00636         ppovlz = ppovl_
00637         deallocate(ppovl_,ppovl)
00638
00639 c$$$ if(ixc==11) then
00640 c$$$ write(6,*)" xxx2: memsize 8*ngb*ngb*nwhis=", 8*ngb*ngb*nwhis,' ngb nwhis=',ngb,nwhis
00641 c$$$ allocate( rcxq(ngb,ngb,nwhis,npm) )
00642 c$$$ rcxq=(0d0,0d0)
00643 c$$$ else
00644 c$$$ if(oncew2(2)) write(6,*)" xxx2:allocate zxq zxqi memsize 16*ngb*ngb*(nwp+niw)=",
00645 c$$$ & 16*ngb*ngb*(1+nwp+niw),' ngb nwp niw=',ngb,nwp,niw
00646 c$$$ allocate(
00647 c$$$ & zxq (ngb,ngb,nw_i:nw), !,nwp) feb2006
00648 c$$$ & zxqi(ngb,ngb,niw))
00649 c$$$ zxq=0d0; zxqi=0d0
00650 c$$$ endif
00651
00652         allocate( rcxq(nmbas1,nmbas2,nwhis,npm) )
00653         allocate( zw0(ngb,ngb) !, zxq (ngb,ngb,nw_i:nw), zxqi(ngb,ngb,niw) )
00654         rcxq = 0d0
00655
00656 !! -----
00657 !! === loop over spin=== -----
00658 !! -----
00659 !TIME0_180001 Do1003
00660 !      do 1003 is = 1,nspinx
00661         do 1003 is = mpi__ss,mpi__se
00662             write(6,*)"('### ',2i4,' out of ngibz+n0qi+nq0iadd nsp=' ,2i4,' ### ')"
00663             & iq, is, ngibz + nq0i+nq0iadd, nspin
00664             if(debug) write(6,*)' niw nw=',niw,nw
00665             isf = is
00666
00667 !! Tetrahedron weight.
00668 !! output
00669 !!      nbnbx
00670 !!      ihw(ibjb,kx): omega index, to specify the section of the histogram.
00671 !!      nhw(ibjb,kx): the number of histogram sections
00672 !!      jhw(ibjb,kx): pointer to whw

```

```

00673 !!      whw( jhw(ibjb,kx) ) \to whw( jhw(ibjb,kx) + nhw(ibjb),kx)-1 ), where ibjb=ibjb(ib,jb,kx)
00674 !!      : histogram weights for given ib,jb,kx for histogram sections
00675 !!      from ihw(ibjb,kx) to ihw(ibjb,kx)+nhw(ibjb,kx)-1.
00676 c      write(6,*) ' --- goto x0kf_v4hz ---- newaniso= ',newaniso2
00677 !! input
00678 !!      ekxx1 for rk,is
00679 !!      ekxx2 for q+rk,isf
00680      do kx = 1, nqbz
00681          call readeval(qbz(:,kx), is, ekxx1(1:nband, kx) )
00682          call readeval(q+qbz(:,kx), isf, ekxx2(1:nband, kx) )
00683      enddo
00684      call gettetwt(q,iq,is,isf,nwgt(:,iq),frhis,nwhis,npm,
00685      i      qbas,ginv, ef, nqibz, nband,ekxx1,ekxx2, nctot,ecore,
00686      i      nqbz,qbz,nqbwz,qbwz, ntetf,idtetf,iblbz,
00687      i      nbmx,ebmx,mtet,eibzmode) !nov2016
00688
00689 !! == x0kf_v4hz is the main routine to accumalte imaginary part of x0 ==
00690      iqeibz=iq
00691      if(npm==1) then
00692          ncc=0
00693      else
00694          ncc=nctot
00695      endif
00696      call x0kf_v4hz(npm,ncc,
00697      i      ihw,nhw,jhw,whw,nhwtot, ! tetwt5
00698      i      nlb,n2b,nbnbx,nbnb, ! use whw by tetwt5 ,
00699      i      q,
00700      i      nspin,is,isf, !symmetrize, !
00701      i      qbas,ginv, qbz,wbz,
00702      d      nlmto,nqbz,nctot, !noccx,noccxv,
00703      d      nbloch, nwhis, !nlmnm,mdimx,
00704      i      iq,ngb,ngc,ngpmx,ngcmx, !ngb/=ngc+nbloch for smbasis()=T oct2005
00705      i      nqbze,nband,nqibz,
00706      o      rcxq, ! rcxq is the accumulating variable for spins
00707      i      nolfco,zzr,nmbas_in, zcousq, !ppovl,nmbas2 is removed.,nmbas1 pprov1z,
00708      i      chipm,eibzmode, !zloffd,
00709      i      nwgt(:,iqeibz),igx(:,iqeibz),igxt(:,iqeibz),ngrp, eibzsym(:,iqeibz),crpa)
00710      write(6,*) ' end of x0kf_v4h sum rcxq=',sum(abs(rcxq))
00711      call tetdeallocate() !deallocate(ihw,nhw,jhw, whw,ibjb )
00712 c      if(tetra) deallocate( nlb,n2b)
00713      1003 continue;write(6,*) 'end of spin-loop nwp=',nwp !end of spin-loop
00714 !TIME1_180001 "Do1003"
00715 c=====end of spin loop=====
00716
00717 !! symmetrize and convert to Enu basis by dconjg(tranpsoc(zcousq)*rcxq8zcousq if eibzmode
00718 !TIME0_190001 x0kf_sym
00719      if(eibzmode) then
00720          is=1 ! dummy
00721          call x0kf_v4hz_symmetrize(npm,!ncc,
00722      c      i      ihw,nhw,jhw,whw,nhwtot, ! tetwt5
00723      c      i      nlb,n2b,nbnbx,nbnb, ! use whw by tetwt5 ,
00724      i      q,
00725      i      nspin,is,isf, !symmetrize, !
00726      i      qbas,ginv, !qbz,wbz,
00727      c      i      nblocha,!nlm,nlnmv,nlnmc,iclass,
00728      c      i      ppb(1,is),
00729      c      i      icore,ncore,
00730      c      d      nlmto,nqbz,nctot, !noccx,noccxv,
00731      c      d      natom, !nl,nclass,natom,nnc,
00732      d      nbloch, nwhis, !nlmnm,mdimx,
00733      i      iq,ngb,ngc,ngpmx,ngcmx, !ngb/=ngc+nbloch for smbasis()=T oct2005
00734      i      nqbze,nband,nqibz,
00735      o      rcxq, ! rcxq is the accumulating variable for spins
00736      i      nolfco,zzr,nmbas_in, zcousq, !ppovl,nmbas2 is removed.,nmbas1 pprov1z,
00737      i      chipm,eibzmode, !zloffd,
00738      i      ngrp, eibzsym(:,iqeibz))
00739      endif
00740
00741 !! reduction rcxq in the S-axis
00742      write(6,*) 'MPI__AllreduceSumS start'
00743      do i_reduction_npm=1,npm
00744          do i_reduction_nwhis=1,nwhis
00745              do i_reduction_nmbas2=1,nmbas2
00746                  call mpi_allreducesums(
00747      .      rcxq(1,i_reduction_nmbas2,i_reduction_nwhis,i_reduction_npm), nmbas1)
00748                  enddo
00749              enddo
00750          enddo
00751          write(6,*) 'MPI__AllreduceSumS end'
00752 !TIME1_190001 "x0kf_sym"
00753 !TIME0_200001 "HilbertTransformation"
00754 !! --- Hilbert transform. Genrerate Real part from Imaginary part. =====
00755      if(allocated(zxq) ) deallocate(zxq,zxqi)
00756      allocate(zxq(nmbas1,nmbas2,nw_i:nw), zxqi(nmbas1,nmbas2,niw))
00757      write(6,('goto dpsion5: nwhis nw_i niw nw_w nmbas1 nmbas2=',6i5)) nwhis,nw_i,nw,niw,nmbas1,nmbas2
00758      write(6,*) '----- nmbas1,nmbas2=', nmbas1,nmbas2
00759      call dpsion5(frhis,nwhis, freq_r, nw, freq_i,niw, realomega, imagomega,

```

```

00760 i rcxq, nwmw_1, nmbsas1, nmbsas2, ! rcxq is alterd---used as work
00761 o zxq, zxi,
00762 i chipm, schi, is, ecut, ecuts)
00763 write(6,*)' --- end of dpsion5 ----',sum(abs(zxq)),sum(abs(zxi))
00764 if(allocated(rcxq) ) deallocate(rcxq)
00765 !TIME1_200001 "HilbertTransformation"
00766
00767 !! === RealOmega ===
00768 if (realomega) then
00769 !TIME0_210001 ralloc
00770 if (nspin == 1) zxq = 2d0*zxq !if paramagnetic, multiply x0 by 2
00771 nwmax = nw
00772 nwmin = nw_i
00773 !! prepare for iq0.
00774 iq0 = iq - ngibz
00775 c if(newaniso2) then
00776 c$$$ if( iq==1 ) then
00777 c$$$ write(6,*)'open EPS0inv mpi=',MPI__rank
00778 c$$$ ifepstin = iopen('EPS0inv',0,-1,0)
00779 c$$$ write(ifepstin) ngb
00780 c$$$ endif
00781 c$$$
00782 c$$$ if(iqinit) then
00783 c$$$ allocate( sk(ngb,nwmin:nwmax,nq0i), sks(ngb,nwmin:nwmax,nq0i) )
00784 c$$$ allocate( skI(ngb,niw,nq0i), sksI(ngb,niw,nq0i))
00785 c$$$ iqinit=.false.
00786 c$$$ endif
00787 allocate(epstilde(ngb,ngb))
00788 allocate(epstinvs(ngb,ngb))
00789 c endif
00790 !KINO write(6,*)'kino: nwmin,nwmax,ngb=',nwmin,nwmax,ngb
00791 write(6, *)" === trace check for W-V === nwmin nwmax=",nwmin,nwmax
00792 !TIME1_210001 "ralloc"
00793 !TIME0_220001 do1015
00794 do 1015 iw = nwmin,nwmax
00795 frr= dsign(freq_r(abs(iw)),dble(iw))
00796 imode = 1
00797 c if(newaniso2.and.iq<=ngibz) then !for mmmw
00798 if(iq<=ngibz) then !for mmmw
00799 if(iq==1) then
00800 ix=1
00801 zw0(:,1)=0d0
00802 zw0(1,:)=0d0
00803 else
00804 ix=0
00805 endif
00806
00807 !! Eqs.(37),(38) in PRB81 125102 (Friedlich)
00808 do igb1=ix+1,ngb
00809 do igb2=ix+1,ngb
00810 epstilde(igb1,igb2)= -vcousq(igb1)*zxq(igb1,igb2,iw)*vcousq(igb2)
00811 if(igb1==igb2) epstilde(igb1,igb2)=1+epstilde(igb1,igb2)
00812 enddo
00813 enddo
00814 epstinvs(ix+1:ngb,ix+1:ngb)=epstilde(ix+1:ngb,ix+1:ngb)
00815 call matcinv(ngb-ix,epstinvs(ix+1:ngb,ix+1:ngb))
00816
00817 !! w4p writing eps
00818 if(iw==0.and.w4pmode) then
00819 !static epstinvs is saved. For q=0 epstilde (mu=1 skipped). For q/=0 full matrix inversion.
00820 !(ix=1 is set for q=0)
00821 ifw4p = ifile_handle()
00822 open(ifw4p,file='W4PHONON.'//charnum5(iq),form='unformatted')
00823 write(ifw4p) iq,q,ngb,ix !ix=0, or ix=1 for q=0 (iq=1)
00824 write(ifw4p) epstinvs(ix+1:ngb,ix+1:ngb)
00825 close(ifw4p)
00826 endif
00827 !TIME0_3000011 zweqzw0
00828
00829 c$$$ cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00830 c$$$ cmmm direct inversion vs. block inversion
00831 c$$$ if(iq>ngibz) then
00832 c$$$ c direct inversion
00833 c$$$ ix=0
00834 c$$$ do igb1=ix+1,ngb
00835 c$$$ do igb2=ix+1,ngb
00836 c$$$ epstilde(igb1,igb2)= -vcousq(igb1)*zxq(igb1,igb2,iw)*vcousq(igb2)
00837 c$$$ if(igb1==igb2) epstilde(igb1,igb2)=1+epstilde(igb1,igb2)
00838 c$$$ enddo
00839 c$$$ enddo
00840 c$$$ epstinvs(ix+1:ngb,ix+1:ngb)=epstilde(ix+1:ngb,ix+1:ngb)
00841 c$$$ call matcinv(ngb-ix,epstinvs(ix+1:ngb,ix+1:ngb))
00842 c$$$ do igb1=1+ix,ngb
00843 c$$$ do igb2=1+ix,ngb
00844 c$$$ zw0(igb1,igb2)= vcousq(igb1)*epstinvs(igb1,igb2)*vcousq(igb2)
00845 c$$$ if(igb1==igb2) zw0(igb1,igb2)= zw0(igb1,igb2)-vcousq(igb1)*vcousq(igb2)
00846 c$$$ enddo

```

Generated on Fri Nov 11 2016 17:43:19 for ecalj/fpgw/ code document by Doxygen







```

01016         if(iq==1) then
01017             ix=1
01018             zw0(:,1)=0d0
01019             zw0(1,:)=0d0
01020         else
01021             ix=0
01022         endif
01023         do igb1=ix+1,ngb
01024             do igb2=ix+1,ngb
01025                 epstilde(igb1,igb2)= -vcousq(igb1)*zxqi(igb1,igb2,iw)*vcousq(igb2)
01026                 if(igb1==igb2) epstilde(igb1,igb2)=1+epstilde(igb1,igb2)
01027             enddo
01028         enddo
01029         epstinv=epstilde
01030         call matcinv(ngb-ix,epstinv(ix+1:ngb,ix+1:ngb))
01031         do igb1=ix+1,ngb
01032             do igb2=ix+1,ngb
01033                 zw0(igb1,igb2)= vcousq(igb1)*epstinv(igb1,igb2)*vcousq(igb2)
01034                 if(igb1==igb2) zw0(igb1,igb2)= zw0(igb1,igb2)-vcousq(igb1)*vcousq(igb2)
01035             enddo
01036         enddo
01037 c$$$                if(iq==1) write(ifepstinv) epstinv(ix+1:ngb,ix+1:ngb),iq,iw
01038
01039         zw(1:ngb,1:ngb) = zw0 ! zw(nblochpmx,nblochpmx)
01040         if (mpi__roots) then
01041             write(ifrcwi, rec= iw) zw ! WP = vsc-v
01042         endif
01043         call tr_chkwrite("freq_i iq iw imgomg trwv=",zw,iw,freq_i(iw),nblochpmx,nbloch,ngb
, iq)
01044     endif
01045
01046 c             if( newaniso2.and.iq>nqibz) then
01047         if(iq>nqibz) then
01048             !! Full inversion to calculalte eps with LFC.
01049             vcoul = fourpi/sum(q**2*tpioa**2) ! --> vcousq(1)**2! !fourpi/sum(q**2*tpioa**2-eee)
01050             if(localfieldcorrectionllw()) then
01051                 ix=0
01052                 do igb1=ix+1,ngb
01053                     do igb2=ix+1,ngb
01054                         if(igb1==1.and.igb2==1) then
01055                             epstilde(igb1,igb2)= 1d0 - vcoul*zxqi(1,1,iw)
01056                             cycle
01057                         endif
01058                         epstilde(igb1,igb2)= -vcousq(igb1)*zxqi(igb1,igb2,iw)*vcousq(igb2)
01059                         if(igb1==igb2) then
01060                             epstilde(igb1,igb2)=1d0 + epstilde(igb1,igb2)
01061                         endif
01062                     enddo
01063                 enddo
01064                 epstinv(ix+1:ngb,ix+1:ngb)=epstilde(ix+1:ngb,ix+1:ngb)
01065                 call matcinv(ngb-ix,epstinv(ix+1:ngb,ix+1:ngb))
01066                 if(iq0<=ng0i) llwi(iw,iq0)= 1d0/epstinv(1,1)
01067             else
01068 c commentout block inversion
01069 c$$$                skI (1:ngb,iw,iq0)= zxqi(1,1:ngb,iw)
01070 c$$$                sksI (1:ngb,iw,iq0)= zxqi(2,1:ngb,iw) !nmbas1=2 see z1stcol in x0kf_v4h.
01071 c$$$                sksI (1:ngb,iw,iq0)= zxqi(1:ngb,1,iw) !nmbas1=2 see z1stcol in x0kf_v4h.
01072 c$$$                vcoul = fourpi/sum(q**2*tpioa**2) ! test-->vcousq(1)**2
!fourpi/sum(q**2*tpioa**2-eee)
01073 c$$$                vcoulsq= sqrt(vcoul)
01074 c$$$!! llwI without LFC. LFC contribution is added in
01075             if(iq0<=ng0i) llwi(iw,iq0)= 1d0 -vcoul*zxqi(1,1,iw) !- vcoulsq*sum( skI(2:ngb) *
w_ksI(2:ngb)*vcousq(2:ngb) )
01076             endif
01077             if(iq0<=ng0i) write(6, "('iq iw_img eps(wLFC) eps(noLFC)',i4,i4,2f10.4,2x,2f10.4)")
01078             & iq,iw,llwi(iw,iq0),1d0-vcoul*zxqi(1,1,iw)
01079         endif
01080
01081 1016         continue
01082 c             if(newaniso2) then
01083 c$$$                if(iq==1) ifepstinv = iclose('EPS0inv') !iq==1 close write mode.
01084                 deallocate(epstinv)
01085                 if(allocated(epstilde)) deallocate(epstilde)
01086 c             endif
01087         endif
01088 !! === ImagOmega end ===
01089 !TIME1_230001 "imagomega"
01090
01091 c         1002 continue ! end of frequency block-loop
01092         if(allocated(vcoul)) deallocate(vcoul)
01093         if(allocated(zw0)) deallocate(zw0)
01094         if(allocated(zxq )) deallocate(zxq)
01095         if(allocated(zxqi)) deallocate(zxqi)
01096
01097         if (mpi__roots) then
01098             ifrcwi = iclose('WVI.'//charnum5(iq))
01099             ifrcw = iclose('WVR.'//charnum5(iq))

```

```

01100         endif
01101 !!
01102 1001 continue
01103 !TIME1_170001 "do1001"
01104 c=====end of loop over q point =====
01105 c=====
01106     call mpi_barrier()
01107
01108 !TIME0_24001 w0mpi
01109 !! == Recieve llw and llwI at node 0, where q=0(iq=1) is calculated. ==
01110     if(mpi_size/=1) then
01111         do iq=nqibz+1,iqzend
01112             iq0 = iq - nqibz
01113 c     write(6,*)' iq iq0 mpi_rank mpi_ranctab(iq)=',iq,
01114         iq0,MPI_rank,MPI_ranctab(iq),MPI_root,nw,nw_i,niw
01115         if(mpi_qranctab(iq)/=0) then !jan2012
01116             if(mpi_qranctab(iq) == mpi_rankq) then
01117 c     write(6,*)' mpi_send iq from',iq,MPI_ranctab(iq)
01118 c     write(6,*)' send llw sum=',sum(abs(llw(:,iq0))),nw,nw_i
01119 c     do i=nw_i,nw
01120 c     write(6,*)'sendxxx',i,llw(i,iq0)
01121 c     enddo
01122 c     write(6,*)' send llwI sum=',sum(abs(llwI(:,iq0))),niw
01123         dest=0
01124         if(iq0<=nq0i) then
01125             call mpi_dblecomplexsendq(llw(nw_i,iq0),(nw-nw_i+1),dest)
01126             call mpi_dblecomplexsendq(llwi(1,iq0),niw,dest)
01127         endif
01128         if(ixyz(iq0)/=0) then
01129             call mpi_dblecomplexsendq(wmuk(2:ngbq0,ixyz(iq0)),ngbq0-1,dest)
01130         endif
01131 c     write(6,*)' mpi_recv iq from',iq,MPI_ranctab(iq),nw,nw_i,niw
01132         src=mpi_qranctab(iq)
01133         if(iq0<=nq0i) then
01134             call mpi_dblecomplexrecvq(llw(nw_i,iq0),(nw-nw_i+1),src)
01135             call mpi_dblecomplexrecvq(llwi(1,iq0),niw,src)
01136         endif
01137         if(ixyz(iq0)/=0) then
01138             call mpi_dblecomplexrecvq(wmuk(2:ngbq0,ixyz(iq0)),ngbq0-1,src)
01139         endif
01140 c     do i=nw_i,nw
01141 c     write(6,*)'recvxxx',i,llw(i,iq0)
01142 c     enddo
01143 c     write(6,*)' recv llw sum=',sum(abs(llw(:,iq0))),nw,nw_i
01144 c     write(6,*)' recv llwI sum=',sum(abs(llwI(:,iq0))),niw
01145         endif
01146     endif
01147     enddo
01148     endif
01149 !TIME1_24001 "w0mpi"
01150
01151 c commentout block inversion
01152 c$$$!! Add LFC (local field correction) to llw and llwI
01153 c$$$     if(newaniso2 .and. MPI_rank == 0 ) then ! only on root node
01154 c$$$         iq=1 !for q=0
01155 c$$$         vcoudfile='Vcoud.'//charnum5(iq)
01156 c$$$         ifvcoud = iopen(trim(vcoudfile),0,-1,0)
01157 c$$$         read(ifvcoud) ngb0
01158 c$$$         read(ifvcoud) qvv
01159 c$$$         if(sum(abs(qvv))>1d-10) then
01160 c$$$             write(6,*)'qvv =',qvv
01161 c$$$             stop 'hx0fp0: qvv/=0 hvcc is not consistent'
01162 c$$$         endif
01163 c$$$         if(allocated(zcousq0)) deallocate( zcousq0,vcousq0 )
01164 c$$$         allocate( zcousq0(ngb0,ngb0),vcousq0(ngb0))
01165 c$$$         read(ifvcoud) vcousq0
01166 c$$$         read(ifvcoud) zcousq0
01167 c$$$         idummy=iclose(trim(vcoudfile))
01168 c$$$         vcousq=sqrt(vcousq)
01169 c$$$         allocate(epstinvc(ngb0,ngb0),w_k(ngb0),w_ks(ngb0),w_kI(ngb0),w_ksI(ngb0),eemat(ngb0,ngb0))
01170 c$$$
01171 c$$$         do iq0=1,nq0i
01172 c$$$             iq = iq0 + nqibz
01173 c$$$             q = qibze(:,iq)
01174 c$$$
01175 c$$$             vcoudfile='Vcoud.'//charnum5(iq)
01176 c$$$             ifvcoud = iopen(trim(vcoudfile),0,-1,0)
01177 c$$$             read(ifvcoud) ngb
01178 c$$$             read(ifvcoud) qvv
01179 c$$$             if(sum(abs(qvv-q))>1d-10) then
01180 c$$$                 write(6,*)'qvv =',qvv
01181 c$$$                 stop 'hx0fp0: qvv/=0 hvcc is not consistent'
01182 c$$$             endif
01183 c$$$             if(allocated(zcousq)) deallocate(zcousq)
01184 c$$$             if(allocated(vcousq)) deallocate(vcousq)
01185 c$$$             allocate( zcousq(ngb0,ngb0),vcousq(ngb0))

```

```

01186 c$$$      read(ifvcoud) vcousq
01187 c$$$      read(ifvcoud) zcousq
01188 c$$$      idummy=iclose(trim(vcoudfile))
01189 c$$$      vcousq=sqrt(vcousq)
01190 c$$$
01191 c$$$      ifepstin = iopen('EPS0inv',0,0,0)
01192 c$$$      read(ifepstin) ngb
01193 c$$$
01194 c$$$      ngc=ngb-nbloch
01195 c$$$      if(allocated(ppovlz)) deallocate(ppovlz)
01196 c$$$      if(allocated(ppovl)) deallocate(ppovl)
01197 c$$$      allocate(ppovl(ngc,ngc),ppovlz(ngb,ngb))
01198 c$$$      call readppovl0(q,ngc,ppovl) !q was qq
01199 c$$$      ppovlz(1:nbloch,:) = zcousq(1:nbloch,:)
01200 c$$$      ppovlz(nbloch+1:nbloch+ngc,:) = matmul(ppovl,zcousq(nbloch+1:nbloch+ngc,:))
01201 c$$$
01202 c$$$! eemat: Z\mu_i(\b fk=0)^* <i|j> Z\nu_j(\b fk)
01203 c$$$      eemat =matmul(transpose(dconjg(zcousq0)),matmul(ppovlz,zcousq))
01204 c$$$      vcoul = fourpi/sum(q**2*tpioa**2) ! test-->vcousq(1)**2 !fourpi/sum(q**2*tpioa**2-eee)
01205 c$$$      vcoulsq = vcoul*.5
01206 c$$$      write(6,*)
01207 c$$$
01208 c$$$      do iw=nwmin,nwmax
01209 c$$$      read(ifepstin) epstin(2:ngb,2:ngb),iqx,iwx
01210 c$$$      epstin(2:ngb,2:ngb) = matmul( transpose(dconjg(eemat(2:ngb,2:ngb))),
01211 c$$$      & matmul(epstin(2:ngb,2:ngb),eemat(2:ngb,2:ngb)) )
01212 c$$$      if(iw/=iwx) then
01213 c$$$      write(6,*)'iw iwx=',iw,iwx
01214 c$$$      stop 'hx0fp0_sc: iw/=iwx'
01215 c$$$      endif
01216 c$$$      w_k(2:ngb) = vcoulsq*matmul( epstin(2:ngb,2:ngb), sk(2:ngb,iw,iq0)*vcousq(2:ngb))
01217 c$$$      epslfc = -vcoulsq*sum( sks(2:ngb,iw,iq0) * w_k(2:ngb) *vcousq(2:ngb) )
01218 c$$$      llw(iw,iq0) = llw(iw,iq0) + epslfc
01219 c$$$      write(6, "('eps(on real) iq iw',2i4,2f9.3,2x,2f9.3)") iq0,iw,
01220 c$$$      llw(iw,iq0)-epslfc,llw(iw,iq0)
01220 c$$$      enddo
01221 c$$$      do iw=1,niw
01222 c$$$      read(ifepstin) epstin(2:ngb,2:ngb),iqx,iwx
01223 c$$$      if(iw/=iwx) then
01224 c$$$      write(6,*)'iw iwx=',iw,iwx
01225 c$$$      stop 'hx0fp0_sc: iw/=iwx'
01226 c$$$      endif
01227 c$$$      w_kI(2:ngb)= vcoulsq*matmul( epstin(2:ngb,2:ngb), skI(2:ngb,iw,iq0)*vcousq(2:ngb))
01228 c$$$      epslfc=- vcoulsq*sum( sksI(2:ngb,iw,iq0)* w_kI(2:ngb)*vcousq(2:ngb) )
01229 c$$$      llwI(iw,iq0)= llwI(iw,iq0)+epslfc
01230 c$$$      write(6, "('eps(on img ) iq iw',2i4,2f9.3,2x,2f9.3)")iq0,iw,
01231 c$$$      llwI(iw,iq0)-epslfc,llwI(iw,iq0)
01231 c$$$      enddo
01232 c$$$      ifepstin = iclose('EPS0inv')
01233 c$$$      enddo
01234 c$$$      endif
01235
01236
01237 !! == W(0) divergent part and W(0) non-analytic constant part.==
01238 1191 continue
01239 !TIME0_40001 WVRI
01240 c      if(newaniso2 .and. MPI__rank == 0 ) then ! MIZUHO-IR only on root node
01241 if(mpi__rank == 0 ) then ! MIZUHO-IR only on root node
01242
01243
01244 !! ix=1011 is a special mode to overwrite llw and llwI for test purpose
01245 !! A file W0W0I is generated by call w0w0i, but usually unused at anywhere.
01246 if(ixc==1011) then
01247 ifw0w0i = ifile_handle('W0W0I')
01248 open(ifw0w0i,form='unformatted')
01249 read(ifw0w0i) nw_ixx,nwxx,niw,nq0ix
01250 write(6,*)'w0w0i: n=',nw_ixx,nwxx,niw,nq0ix
01251 if(nq0i/=nq0ix) call rx('nq0i/=nq0ix')
01252 if(nw_i/=nw_ixx) call rx(nw_i/=nw_ixx)
01253 if(nw/=nwxx) call rx(nw/=nwxx)
01254 read(ifw0w0i) llw(nw_i:nw,1:nq0i)
01255 read(ifw0w0i) llwi(1:niw,1:nq0i)
01256 c      read(ifw0w0i) w0(nw_i:nw)
01257 c      read(ifw0w0i) w0i(1:niw)
01258 close(ifw0w0i)
01259 endif
01260
01261 !! get w0 and w0i (diagonal element at Gamma point)
01262 !! This return w0 and w0i. (llw and llwi are input)
01263 !! Outputs w0,w0i,llmat. See use m_w0w0i at the beginning of this routine.
01264 call w0w0i(llw,llwi,nw_i,nw,nq0i,niw,q0i) !all inputs. get effective W0,W0i, and L(omega=0)
01265 matrix.
01266
01267 !! Finalize w4phonon
01267 !! wmuk(ix)= matmul(wmu,qv) ==> wmu= matmul(wmuk,qvinv)
01268 if(w4pmode) then
01269 do i=1,3

```

```

01270         qv(:,i)= tpioa*q0i(:,ixyz(i))
01271         qv(:,i)= qv(:,i)/sqrt(sum(qv(:,i)**2))
01272     enddo
01273     call matinv(3,qv)
01274     allocate( wmu(2:ngbq0,3) )
01275     do igb=2,ngbq0
01276         wmu(igb,:) =matmul(wmuk(igb,:),qv)
01277     enddo
01278     ifw4p = ifile_handle()
01279     open(ifw4p,file='W4PHONON.HeadWing',form='unformatted')
01280     write(ifw4p) llmat(1:3,1:3),ngbq0 !for q=0
01281     write(ifw4p) wmu(2:ngbq0,1:3) !for q~0
01282     close(ifw4p)
01283     cccccccccccccccccccccc
01284     c         write(6,*)'ngbq0=',ngbq0
01285     c         write(6,*)'llmat=',llmat
01286     c         write(6,*)'wmu sum=',sum(abs(wmu(2:ngbq0,1:3)))
01287     c         write(6,*)'wmuksum=',sum(abs(wmuk(2:ngbq0,1:3)))
01288     c         call rx(' test end xxxxxxxxxxxxxx')
01289     cccccccccccccccccccccc
01290     deallocate(wmu,wmuk)
01291     endif
01292
01293     !! Read WVR and WVI at Gamma point, and give correct W(0) (averaged in the Gamma cell, where
01294     !! Gamma cell) is the micro cell of BZ including Gamma point).
01295
01296     c         write(6,*)'sumcheck w0,w0i=',sum(abs(w0)),sum(abs(w0i))
01297     !! === w0,w0i are stored to zw for q=0 ===
01298     !! === w_ks+wk are stored to zw for iq >ngibz ===
01299     ! We assume iq=1 is for rank=0
01300     do iq = 1,1 !iq=1 only 4pi/k**2 /eps part only ! iq = iqxini,iqend
01301     c         if( .not. MPI__task(iq) ) cycle
01302     q = qibze(:,iq)
01303     do ircw=1,2
01304         if (ircw==1) then
01305             nini=nw_i
01306             nend=nw
01307             ifrcwx = iopen('WVR.'//charnum5(iq),0,-1,mrecl)
01308         elseif(ircw==2) then; nini=1; nend=niw;
01309             ifrcwx = iopen('WVI.'//charnum5(iq),0,-1,mrecl)
01310         endif
01311     do iw=nini,nend
01312     c         if(iq<=ngibz) read(ifrcwx, rec=((iq-iqxini)*(nend-nini+1)+ iw-nini+1 ) ) zw !(1:ngb,1:ngb)
01313         read(ifrcwx, rec= iw-nini+1 ) zw !(1:ngb,1:ngb)
01314         if( iq==1 ) then
01315             if(ircw==1) zw(1,1) = w0(iw)
01316             if(ircw==2) zw(1,1) = w0i(iw)
01317         endif
01318     c         write(ifrcwx,rec=((iq-iqxini)*(nend-nini+1)+ iw-nini+1 ) ) zw !(1:ngb,1:ngb)
01319         write(ifrcwx,rec=iw-nini+1) zw !(1:ngb,1:ngb)
01320     enddo
01321     if (ircw==1) then
01322         ifrcwx = iclose('WVR.'//charnum5(iq))
01323     elseif(ircw==2) then
01324         ifrcwx = iclose('WVI.'//charnum5(iq))
01325     endif
01326     enddo
01327     end do
01328     endif
01329     is = iclose('hbe.d')
01330     !TIME1_40001 "WVRI"
01331     !TIME1_1001 "ProgAll"
01332     !TIMESHOW
01333     call cputid(0)
01334     write(6,*) '--- end of hx0fp0_sc --- irank=',mpi__rank
01335     call flush(6)
01336     call mpi_finalize
01337     if(ixc==11) call rx0( ' OK! hx0fp0_sc ixc=11 Sergey F. mode')
01338     if(ixc==1011) call rx0( ' OK! hx0fp0_sc ixc=1011 W0W0Ionly')
01339     end program hx0fp0_sc
01340
01341
01342     C=====
01343     subroutine tr_chkwrite(tagname,zw,iw,freqq,nblochpmx,nbloch,ngb,iq)
01344     implicit none
01345     integer:: nblochpmx,nbloch,ngb,iw,i,iq
01346     complex(8):: zw(nblochpmx,nblochpmx),trwv,trwv2
01347     real(8):: freqq
01348     character*(*)::tagname
01349     trwv=0d0
01350     do i = 1,nbloch
01351         trwv = trwv + zw(i,i)
01352     enddo
01353     trwv2 = 0d0
01354     do i = 1,ngb
01355         trwv2 = trwv2 + zw(i,i)
01356     enddo ! write(6,(' realomg trwv=",2i6,4d22.14')' ) iq,iw,trwv(iw),trwv2(iw)

```

```

01357      write(6,'(a,f10.4,2i5,4d22.14)')tagname,freqq,ig,iw,trwv,trwv2
01358 c    do i = 1,ngb
01359 c      write(6,'("iii i=",i4,a,f10.4,2i5,4d22.14)')i,tagname,freqq,ig,iw,zw(i,i)
01360 c    enddo
01361 c  end
01362
01363

```

## 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 !! output 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(ifiqq, file='QGpsi',)
00023 !!   write(ifiqq ) 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 (ifiqq) 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 (ifiqq) 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(ifiqq)
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 m_keyvalue,only: getkeyvalue
00048     implicit none
00049     integer(4) :: n1q,n2q,n3q,ifiqg,ifiqgc,ifigw0,nggrp,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,al,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, lnq0iadd=.false. !keepppovl,
00059     integer(4):: iq0pinxxx ,ifile_handle,n1,n2,n3
00060     integer:: gammacellctrl=0
00061     pi= 4d0* atan(1d0)
00062     call cputid(0)
00063     write(6,*) ' qg4gw: Generate Q0P->1; Readin Q0P->2; band mode->3; SW(chipm)->4'
00064     write(6,*) '          Generate Q0P->101(old offset Gamma)'
00065     write(6,*) '          Generate Q0P and Q0P for xyz ->201 '
00066     read (5,*) iq0pin
00067     call headver('qg4gw',iq0pin)
00068     write(6,*) ' mode iq0pin = ',iq0pin
00069     if(iq0pin==100.or.iq0pin==1.or.iq0pin==2.or.iq0pin==3.or.iq0pin==101) then
00070         iq0pinxxx=iq0pin
00071     elseif(iq0pin==10002) then
00072         iq0pinxxx=2
00073         gammacellctrl=1 !Gammacell skip mode
00074     elseif(iq0pin==20002) then
00075         iq0pinxxx=2
00076         gammacellctrl=2 !Gammacell only mode
00077     elseif(iq0pin==4) then
00078         iq0pinxxx=2
00079     elseif(iq0pin==201) then
00080         iq0pinxxx=1
00081         lnq0iadd=.true.
00082     else
00083         call rx( 'Not allowed iq0pin')
00084     endif
00085     c this is moved to gwinit.m.F march2016
00086     c!! Generate templeta of GWinput for iq0pin=-100
00087     c     if(iq0pin==100) then
00088     c         call conv2gwinput()
00089     c         call rx0( ' OK! qg4gw mode=-100 to generate GWinput')
00090     c     endif
00091     idummy=q0pchoice()
00092     write(6, "(' q0pchoice() = ',i4)") q0pchoice()
00093
00094     ifi=ifile_handle()
00095     open (ifi, file='LATTC')
00096     read(ifi,*) alat
00097     read(ifi,*) plat(1:3,1)
00098     read(ifi,*) plat(1:3,2)
00099     read(ifi,*) plat(1:3,3)
00100     read(ifi,*) !dummy
00101     close(ifi)
00102     !! --- readin SYMOPS. point group operations. r'=matmul(symops(:,,:),r) for any ig.
00103     ifi=ifile_handle()
00104     open (ifi, file='SYMOPS')
00105     read(ifi,*) ngrp
00106     write(6,*) ' SYMOPS ngrp=',ngrp
00107     allocate(symops(3,3,ngrp))
00108     do ig = 1,ngrp
00109         read(ifi,*)
00110         do i=1,3
00111             read(ifi,*) symops(i,1:3,ig)
00112         enddo
00113     enddo
00114     close(ifi)
00115     !! --- check write
00116     write(6,*) ' --- primitive vectors ---'
00117     write(6, "(' unit(a.u.) alat =',f13.6 )") alat
00118     write(6, "(' primitive_1 =',3f13.6)") plat(1:3,1)
00119     write(6, "(' primitive_2 =',3f13.6)") plat(1:3,2)
00120     write(6, "(' primitive_3 =',3f13.6)") plat(1:3,3)
00121     write(6,*) ' --- point group operations --- '
00122     do ig = 1,ngrp
00123         print *, ' ig=',ig
00124         do i=1,3
00125             write(6, "(3f14.6)") symops(i,1:3,ig)
00126         enddo
00127     enddo
00128     !! --- Readin GWinput
00129     call getkeyvalue("GWinput", "n1n2n3", nnn,3)
00130     n1q=nnn(1); n2q=nnn(2); n3q = nnn(3)
00131     call getkeyvalue("GWinput", "QpGcut_psi",qpgx2)
00132     call getkeyvalue("GWinput", "QpGcut_cou",qpgcut_cou)
00133     call getkeyvalue("GWinput", "unit_2pioa",unit2)

```

```

00134         if(unit2) then
00135             unit = 2d0*pi/alat
00136             qpgx2 = qpgx2 *unit
00137             qpgcut_cou= qpgcut_cou *unit
00138         endif
00139         qpgcut_psi = qpgx2
00140         write(6, "(' --- k points for GW from GWinput =',3i3)") nnn(1:3)
00141         write(6, "(' --- |k+G| < QpG(psi) QpG(Cou)=' ,2d13.6)") qpgcut_psi, qpgcut_cou
00142         ifiqg = ifile_handle()
00143         open(ifiqg ,file='QGpsi',form='unformatted')
00144         ifiqgc = ifile_handle()
00145         open(ifiqgc,file='QGcou',form='unformatted')
00146         if(iq0pin==4) then
00147             qpgcut_psi=0d0
00148             qpgcut_cou=0d0
00149         endif
00150     !!
00151     call mkqg2(alat,plat,symops,ngroup,nnn,iq0pinxxx,
00152     & qpgcut_psi, qpgcut_cou, ifiqg, ifiqgc, gammacellctrl,lnq0iadd)
00153     write(6,*) ' OK! End of qg4gw '
00154     if(iq0pin ==1) call rx0( ' OK! qg4gw mode=1 normal mode')
00155     if(iq0pin ==2) call rx0( ' OK! qg4gw mode=2 Readin Q0P mode')
00156     if(iq0pin ==10002) call rx0( ' OK! qg4gw mode=10002 Readin Q0P. GammaCell skipped.')
00157     if(iq0pin ==20002) call rx0( ' OK! qg4gw mode=20002 Readin Q0P. GammaCell Only.')
00158     if(iq0pin ==3) call rx0( ' OK! qg4gw mode=3 band-plot mode')
00159     if(iq0pin ==4) call rx0( ' OK! qg4gw mode=4 Readin Q0P mode. Set ngp=ngc=0')
00160     end

```

## 4.39 Wannier/genMLWF File Reference

### Variables

- if [ $\$ \# - ne\ 3$ ][ $\$ 2 != -np$ ]
- then echo An example of [usage](#)
- then  $\$ echo\_run$  echo!Perform  
job\_band in advance exit fi  
source  $\$ nfpgw\ run\_arg$   
 $\$ echo\_run$  echo rm f SYML BNDS  
In s [syml](#)
- then mv sigm \$MATERIAL sigm  
\$MATERIAL bakup In s f sigm  
sigm \$MATERIAL  $\$ echo\_run$  echo  
sigm is used sigm \$MATERIAL is  
softlink to it fi else  
 $\$ echo\_run$  echo Neither sigm  
nor sigm \$MATERIAL [exists](#)
- run\_arg \$argin \$NO\_MPI \$nfpgw  
lmfgw llmfgw00 \$MATERIAL [argin](#) =1

### 4.39.1 Variable Documentation

4.39.1.1 run\_arg \$argin \$MPI\_SIZE \$nfpgw hx0fp0 lx0\_10011 argin =1

Definition at line 52 of file [genMLWF](#).

4.39.1.2 then mv sigm \$MATERIAL sigm \$MATERIAL bakup In s f sigm sigm \$MATERIAL  $\$ echo\_run$  echo sigm is used sigm  
\$MATERIAL is softlink to it fi else  $\$ echo\_run$  echo Neither sigm nor sigm \$MATERIAL exists

**Initial value:**

```

==> LDA '
fi

```

```
run_arg '---' $NO_MPI $nfpwgw /lmfa llmfa $MATERIAL # if lmfa is not yet.
run_arg '---' $MPI_SIZE $nfpwgw /lmf-MPIK llmf_start $MATERIAL
rm -f ewindow.${MATERIAL}* qbyl.${MATERIAL}* eigze.${MATERIAL}* # remove temporary files.
```

argin=0

Definition at line 42 of file [genMLWF](#).

4.39.1.3 then cat UUq0U \*UUq0U rm f UUq0U \*fi if

Definition at line 9 of file [genMLWF](#).

4.39.1.4 then \$echo\_run echo ! Perform job\_band in advance exit fi source \$nfpwgw run\_arg \$echo\_run echo rm f SYML BNDS ln s sym1

**Initial value:**

```
{MATERIAL} SYML
ln -s bnds.${MATERIAL} BNDS
```

```
if [ -e sigm ]
```

Definition at line 31 of file [genMLWF](#).

4.39.1.5 then echo An example of usage

Definition at line 19 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 nfpwgw=`dirname $0`
00015 MATERIAL=$1
00016 MPI_SIZE=$3
00017 NO_MPI=0
00018 ### end of processing input arguments ###
00019 if [ ! -e bnds.$1 ];then
00020     $echo_run echo "!!! Perform job_band in advance!"
00021     exit
00022 fi
00023
00024 ### Read functions run_arg and run_arg_tee defined in a file run_arg ###
00025 source $nfpwgw/run_arg
00026
00027
00028 ##### start here #####
00029 $echo_run echo "### START genMLWF: MPI size= " $MPI_SIZE, "MATERIAL= "$MATERIAL
00030 rm -f SYML BNDS
00031 ln -s sym1.${MATERIAL} SYML
00032 ln -s bnds.${MATERIAL} BNDS
00033 ## Make softlink from sigm --> sigm.$MATERIAL.
00034 ## If sigm and sigm.$MATERIAL coexist, sigm.$MATERIAL is moved to sigm.$MATERIAL.backup in advance.
00035 if [ -e sigm ] ; then
00036     if [ -e sigm.$MATERIAL ] ; then
00037         mv sigm.$MATERIAL sigm.$MATERIAL.backup
```



```

00038         ln -s -f sigm sigm.$MATERIAL
00039         $echo_run echo '--- sigm is used. sigm.$MATERIAL is softlink to it ---'
00040     fi
00041 else
00042     $echo_run echo '--- Neither sigm nor sigm.$MATERIAL exists. ==> LDA '
00043 fi
00044
00045 ##### lmf part #####
00046 run_arg '---' $NO_MPI $nfpwg /lmfa llmfa $MATERIAL # if lmfa is not yet.
00047 run_arg '---' $MPI_SIZE $nfpwg /lmf-MPIK llmf_start $MATERIAL
00048 rm -f ewindow.${MATERIAL}* qbyl.${MATERIAL}* eigze*.${MATERIAL}* # remove temporaly files.
00049
00050 ##### preparation of required inputs for GW (mainly prepare required eigenfuncitons) #####
00051 argin=0; run_arg $argin $NO_MPI $nfpwg /lmfgw llmfgw00 $MATERIAL
00052 argin=1; run_arg $argin $NO_MPI $nfpwg /qg4gw lqg4gw #Generate requied q+G vectors.
00053 argin=1; run_arg $argin $MPI_SIZE $nfpwg /lmfgw-MPIK llmfgw01 $MATERIAL
00054 run_arg '---' $NO_MPI $nfpwg /lmf2gw llmf2gw #reform data for gw
00055
00056 ##### GW related part (up to preparation of MPB) #####
00057 argin=0; run_arg $argin $NO_MPI $nfpwg /rdata4gw_v2 lrddata4gw_v2
00058 if [ -e ANFcond ];then # This is for ANFcond. Unused recently
00059     # cp EVU EVD
00060     $echo_run echo "Not maintained recently"
00061     exit 10
00062 fi
00063 argin=1; run_arg $argin $NO_MPI $nfpwg /heftet leftet # A file EFERMI for hx0fp0
00064 #argin=1; run_arg $argin $NO_MPI $nfpwg /hchknw lchknw # A file NW, containing nw for given QPNT (probably
    only for one-shot GW).
00065 argin=0; run_arg $argin $NO_MPI $nfpwg /hbasfp0 lbas # Product basis generation
00066
00067 ##### maxloc start here #####
00068 argin=1 ;run_arg $argin $NO_MPI $nfpwg /hmaxloc lmaxloc1 # b-vector BBVEC
00069 argin=1 ;run_arg $argin $MPI_SIZE $nfpwg /hpsig_MPI lpsig_MPI # PSIG* =<Psi|Gaussian>.
00070 # Gather all PSIG* into a file. (U meand UP isp=1, D means Down spin isp=2)
00071 cat PSIGU.* >PSIGU
00072 rm -f PSIGU.*
00073 if [ -e PSIGD.0000 ]; then
00074     cat PSIGD.* >PSIGD
00075     rm -f PSIGD.*
00076 fi
00077
00078 argin=2 ;run_arg $argin $MPI_SIZE $nfpwg /huumat_MPI luumat2 # UU (UUmatrix <u_k,i|u_k+b,j>) matrix are
    calculatued.
00079 # Gather all UU*.* into a file UUU/UUD.
00080 cat UUU.* >UUU
00081 rm -f UUU.*
00082 if [ -e UUD.0000 ]; then
00083     cat UUD.* >UUD
00084     rm -f UUD.*
00085 fi
00086 # -- Main part of Wannier (Both of Souza's and Marzari's and procedures sucessively).
00087 argin=2; run_arg $argin $NO_MPI $nfpwg /hmaxloc lmaxloc2 #(band plot data are generated.)
00088
00089
00090 ##### Wannier function plot. *.xsf for Xcrysden. #####
00091 run_arg '---' $NO_MPI $nfpwg /wanplot lwanplot
00092
00093
00094 ### Here on, we calculate W (v and W-v) for Wannier.#####
00095 # -- UUmatrix for Q0P (offset Gamma point) are required calculation v and W at the limit of q \to 0.
00096 argin=3; run_arg $argin $MPI_SIZE $nfpwg /huumat_MPI luumat3
00097 # Gather all UU*.* into a file UU*, PSIG* as well. (U meand UP isp=1, D means Down spin isp=2)
00098 if [ -e UUq0U.0000 ]; then
00099     cat UUq0U.* > UUq0U
00100     rm -f UUq0U.*
00101 fi
00102 if [ -e UUq0D.0000 ]; then
00103     cat UUq0D.* > UUq0D
00104     rm -f UUq0D.*
00105 fi
00106
00107 ### pkm4crpa file mode for crpa ###
00108 argin=10011; run_arg $argin 1 $nfpwg /hwmatK_MPI lpkm4crpa
00109
00110 ### Main part of v, W-v for Wanniers. #####
00111 argin=0; run_arg $argin $MPI_SIZE $nfpwg /hvccfp0 lvcc # Coulomb matrix v
00112 argin=1; run_arg $argin $MPI_SIZE $nfpwg /hwmatK_MPI lwmatK1 # Matrix elements of v for Wannier
00113 argin=111; run_arg $argin $MPI_SIZE $nfpwg /hx0fp0 lx0_111 # Screened Coulomb W minus v, W-v
00114 argin=2; run_arg $argin $MPI_SIZE $nfpwg /hwmatK_MPI lwmatK2 # Matrix element of W-v
00115 # $nfpwg/Cal_W.py
00116
00117 ##### crpa
00118 argin=10011; run_arg $argin $MPI_SIZE $nfpwg /hx0fp0 lx0_10011 # cRPA Screened Coulomb W minus v,
    W-v
00119 argin=100; run_arg $argin $MPI_SIZE $nfpwg /hwmatK_MPI lwmatK2crpa # Matrix element of W-v
00120 # $nfpwg/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, nlmt0)  
*read dimensions of wc,b,hb*
- subroutine [chk\\_cnkweight](#) (qbz, iko\_ix, iko\_fx, cnk, nqbz, nwf, nband, nlmt0)
- subroutine [chk\\_umn](#) (cnk, umnk, qbz, iko\_ix, iko\_fx, iko\_i, iko\_f, nwf, nqbz, nband, nlmt0)

#### 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*, *nlmt0* )

*read dimensions of wc,b,hb*

Definition at line [1165](#) 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*, *nlmt0* )

Definition at line [1234](#) 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*, *nlmt0* )

Definition at line [1303](#) 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 hwmf.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,nqbwz,nteti,ntetf,n1,n2,n3,qbas,ginv, !qbasmc,
00023      & dq_,qbz,wbz,qibz,wibz,qbwz,
00024      & idtetf,iblbz,idteti,
00025      & nstar,irk,nstbz
00026      use m_genallcf_v3,only: genallcf_v3,
00027      & nclass,natom,nspin,n1,nn,ngrp,
00028      & nlmtol,nlmmx, nctot,niw, !nw_input=>nw,ef
00029      & alat,delta,deltaw,esmr,symgrp,clabl,iclass,!,diw,dw
00030      & invg,il,in,im,nlnm,
00031      & plat,pos,ecore,symgg, konf,z,
00032      & spid
00033      use m_read_worb,only: s_read_worb, s_cal_worb,
00034      & nwf,nclass_mlwf,cbas_mlwf,nbasclass_mlwf,
00035      & classname_mlwf,iclassin,
00036      & iphi,iphidot,nphi,nphix
00037      use m_keyvalue,only: getkeyvalue
00038      implicit none
00039 c-----
00040      real(8),allocatable:: r0g(:,:), wphi(:,:)
00041      real(8) :: esmr2,shtw
00042      integer :: iclass2
00043      integer(4)::
00044      & ixc,iopen,ifhbed, nprecb,mrecb,mrece,nlmtot,nqbzt, nband,
00045      & ibas,ibasx,ngpmx,nxx,ngcmx,nbloch,ifqpnt,ifwd,ifbb,
00046      & nprecx,mrecl,nblochpmx2,nwt,niwt, nqnum,mdimx,nblochpmx,
00047      & ifrcw,ifrcwi, noccxv,maxocc2,noccx,ifvcfpout,iqall,iarf,ntq,
00048      & i,j,k,nspinmx, nq,is,ip,iq,idxk,ifoutsex,iclose,nq0i,ig,
00049      & mxkp,nqibzxx,ntet,nene,iqi, ix,iw,
00050      & nlnx4,niwx,irot,invr,invrot,ivsum, ifoutsec,ntqx,
00051      & ifmlw(2),ifmlwe(2) !,ifcphi
00052      & ,ifxc(2),ifsex(2), ifphiv(2),ifphic(2),ifec,ifexsp(2),
00053      & ifsecomg(2),ifexx,ifwand,ndble=8
00054      real(8) :: pi,tpia,vol,voltot,rs,alpha,
00055      & qfermi,efx,valn,efnew,edummy,efz,qm,xsex,egex,
00056      & zfac1,zfac2,dscdw1,dscdw2,dscdw,zfac,ef2=1d99,exx,exqx,exxelgas
00057      logical lqall,laf
00058
00059      integer(4),allocatable :: itq(:)
00060      real(8),allocatable :: q(:,:)
00061
00062 c takao
00063      integer(4),allocatable :: ngvecpb(:,:,:),!ngveccB(:,:,:),
00064      & ngvecp(:,:), ngvecc(:,:),iqib(:,:), !,ngpn(:,)ngcni(:,)
00065      & kount(:,:), nx(:,:),nblocha(:,)lx(:,) !ngveccBr(:,:,:)
00066      real(8),allocatable:: vxcfp(:,:,:),
00067      & wqt(:,) wgt0(:,:),q0i(:,:),
00068      & ppbrd(:,:,:),:,:,:),cgr(:,:,:),egt(:,)
00069      & ppbrdx(:,:,:),:,:,:),aaa(:,:), !symope(:,:,:)=symgg, ! qibz(:,:),
00070      & ppb(:,) eq(:,) !,pdb(:,),dpb(:,),ddb(:,)
00071      & eqx(:,:),eqx0(:,:,:),ekc(:,),coh(:,:)
00072      & ,rw_w(:,:,:),:,:),cw_w(:,:,:),:,:),
00073      & rw_iw(:,:,:),:,:),cw_iw(:,:,:),:,:),
00074      complex(8),allocatable:: geigb(:,:,:),
00075 c
00076      logical :: screen,exchange,cohtest,legas,tote
00077      real(8) :: rydberg,hartree
00078      real(8):: qreal(3), ntot,nocctotg2,trip1!,xxx(3,3)
00079      real(8):: qlat(3,3)
00080      logical ::nocore
00081
00082 c space group infirmation
00083      integer(4),allocatable :: iclasst(:,) invgx(:,) mlat(:,:)
00084      real(8),allocatable :: tiat(:,:,:),shtvg(:,:)
```

```

00085
00086 c
00087     real(8),allocatable :: eexl(:,:),exspl(:,:),qqexl(:,:,:)
00088     integer(4),allocatable:: nspex(:,:),ieord(:,:),itexl(:,:)
00089     real(8) :: qqex(1:3), eex,exsp,eee, exwgt,deltax0
00090     integer(4) :: itmx,ipex,itpex,itex,nspexmx,nnex,isig,iex,ifexspx
00091     & ,ifexspxx ,ifefsm, nq0ix,ifemesh,nz
00092     character(3) :: charnum3,sss
00093     character(12) :: filenameex
00094     logical :: exspwrite=.false.
00095     character*8 xt
00096
00097
00098     integer(4)::nqbze,ini,nq0it,idummy
00099     real(8),allocatable:: qbze(:,:)
00100
00101     real(8) :: ebm
00102     integer(4):: nbm
00103
00104     real(8):: volwgt
00105
00106     integer(4)::nwin, incwfin
00107     real(8)::efin,ddw
00108     integer(4),allocatable::imdim(:)
00109     real(8),allocatable::freqx(:),freqw(:),wwx(:),expa(:)
00110
00111     logical:: gaussssmear !readgwinput,
00112     integer(4)::ret
00113     character*(150):: ddd
00114
00115
00116     integer(4):: bzcage, ngpnl,mrecg,verbose,ngcnl,nwxx
00117     real(8) :: wgtq0p,quu(3)
00118
00119     integer(4):: iii,isx,ivsumxxx
00120
00121 c for maxloc
00122     real(8) :: wbb(12),wbbsum,bb(3,12),
00123     c eomin,eomax,eimin,eimax,
00124     c qwf0(3),dqwf0(3),qks(3),q0(3)
00125     complex(8),allocatable:: uumat(:,:,:),evecc(:,:),eveccs(:,:),
00126     c amnk(:,:,:),cnk(:,:,:),umnk(:,:,:)
00127     real(8),allocatable:: ku(:,:),kbu(:,:,:),eunk(:,:),eval(:,:),evals(:),
00128     c eks(:,:),rt(:,:),rt8(:,:,:),qbz0(:,:)
00129     integer(4):: nbb,isc,ifq0p,
00130     c nox,iko_ix,iko_fx,
00131     c noxs(2),iko_ixs(2),iko_fxs(2),
00132     c ieo_swt,iei_swt,itin_i,itin_f,itout_i,itout_f,
00133     c nbbelow,nbabove
00134     integer(4),allocatable:: ikbidx(:,:)
00135     integer(4),allocatable:: iki_i(:),iki_f(:),
00136     c ikbi_i(:,:),ikbi_f(:,:),
00137     c iko_i(:),iko_f(:),
00138     c ikbo_i(:,:),ikbo_f(:,:)
00139     logical :: leout,lein,lbin,lq0p,lsyml,lbnds
00140     logical :: debug=.false.
00141 !
00142     integer(4):: nlinex,ntmp
00143     parameter(nlinex=100)
00144     integer(4)::nline,np(nlinex)
00145     real(8):: qi(3,nlinex),qf(3,nlinex)
00146 c step 1
00147     complex(8),allocatable:: cnq0(:,:),
00148     c upu(:,:,:),cnk2(:,:,:),
00149     c zmn(:,:)
00150     complex(8):: ctmp
00151     real(8),allocatable:: omgik(:)
00152     real(8) :: omgi,omgiold,conv1,alpha1,dmg1,qtmp(3)
00153     integer(4):: nsc1,ndz,nin,ifhoev,ifuu0,ifpsig
00154 c step 2
00155     complex(8),allocatable:: mmn(:,:,:),mmn0(:,:,:),
00156     c rmn(:,:),smn(:,:),amn(:,:),
00157     c tmn(:,:),dwmn(:,:)
00158     real(8),allocatable:: rn(:,:),qn(:)
00159     real(8) :: omgd,omgod,omgdod,omgidod,omgdodold,dmgdod,
00160     c conv2,alpha2
00161     integer(4):: nsc2,ibb,ii,ij,ik
00162     logical :: lrmn,lmmn
00163 c step 3
00164     complex(8),allocatable:: hrotk(:,:,:),hrotr(:,:,:),hrotkp(:,:)
00165     c , hrotkps(:,:)
00166     real(8):: e1,e2,rcut
00167     integer(4):: iband,ifbnd,iftb,ifsh,nsh,nsh1,nsh2
00168     logical :: lsh
00169     real(8),allocatable :: rws(:,:),drws(:)
00170     integer(4),allocatable:: irws(:)
00171     integer(4):: nrws,ifham

```

```

00172
00173 c ixc=3
00174     character(20)::filename
00175     complex(8),allocatable:: hrotrcut(:, :, :)
00176     integer:: ifh
00177     real(8):: heps ,r_v
00178
00179     real(8)::qold(3)
00180     real(8),allocatable:: xq(:,),eval1(:,),eval2(:,),eval3(:,)
00181
00182     integer::npin
00183     real(8):: qiin(3),qfin(3)
00184
00185     integer(4),allocatable::
00186     & m_indx(:,),n_indx(:,),l_indx(:,),ibas_indx(:,),ibasiwf(:,)
00187     integer:: ifoc,iwf,ldim2,ixx,ifile_handle
00188
00189     real(8):: enwfmax,qxx(3),eeee,enwfmaxi, ef
00190     integer:: inii
00191     logical:: leauto,leinauto
00192
00193 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00194 c     open(1107,file='xxx1')
00195 c     open(1108,file='xxx2')
00196 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00197
00198 c-----
00199     hartree=2d0*rydberg()
00200
00201     iii=verbose()
00202     write(6,*)' verbose=',iii
00203
00204 c mode switch. -----
00205     write(6,*) ' --- Choose omodes below -----'
00206     write(6,*) ' bb vectors (1) or Wannier fn. (2) or TB Hamiltonian (3)'
00207     write(6,*) ' --- Put number above ! -----'
00208     call readin5(ixc,nz,idummy)
00209     write(6,*) ' ixc=',ixc
00210     if(ixc<1.or.ixc>3) call rx(' --- ixc=0 --- Choose computational mode!')
00211
00212 c--- readin BZDATA. See gwsrc/rwbzdata.f
00213 c-----readin data set when you call read_BZDATA -----
00214 c     integer(4)::ngrp,nqbz,nqibz,nqbwz,nteti,ntetf
00215 ccccc ! & ,n_index_qbz
00216 c     integer(4):: n1,n2,n3
00217 c     real(8):: qbas(3,3),ginv(3,3),qbasmc(3,3)
00218 c     real(8),allocatable:: qbz(:,),wbz(:,),qibz(:,)
00219 c     & ,wibz(:,),qbwz(:,)
00220 c     integer(4),allocatable:: idtetf(:,),iblbz(:,),idtet1(:,)
00221 c     & ,nstar(:,),irk(:,),nstbz(:,) !,index_qbz(:,),)
00222 c-----
00223     call read_bzdata()
00224     write(6,*)' nqibz ngrp=',nqibz,ngrp
00225     write(6,*)' nqbz =',nqbz
00226 c     write(6,*) qbz
00227 c     write(6,*)' irk=',irk
00228 c     write(6,*)' #### idtetf: ####'
00229 c     write(6,*) idtetf
00230
00231 c set up work array
00232 c     call wkinit (iwksize)
00233 c     call pshprt(60)
00234
00235 c--- readin GWIN and LMTO, then allocate and set datas.
00236 c     nwin =-999 !not readin NW file
00237 c     efin =-999d0 !not readin EFERMI
00238 c     efin = 0d0 !readin EFERMI
00239 c     call readfermi()
00240     incwfin= -1 !use 7th column for core at the end section of GWIN
00241     call genallcf_v3(incwfin) !in module m_genallcf_v3
00242     if(ngrp/= ngrp2) stop 'ngrp inconsistent: BZDATA and LMTO GWIN_V2'
00243 c--- These are allocated and setted.
00244 c     integer(4):: nclass,natom,nspin,nl,nn,nnv,nnc, ngrp,
00245 c     o nlmto,nlnx,nlnxv,nlnxc,nlnmx,nlnmxv,nlnmxc, nctot,niw, !not readin nw
00246 c     real(8) :: alat,ef, diw,dw,delta,deltaw,esmr
00247 c     character(120):: symgrp
00248 c     character(6),allocatable :: clabl(:)
00249 c     integer(4),allocatable:: iclass(:)
00250 c     & ,nindxv(:,),nindxc(:,),ncwf(:,),) ,
00251 c     o invg(:,), il(:,), in(:,), im(:,), ilnm(:,), nlnm(:,),
00252 c     o ilv(:,),inv(:,),imv(:,), ilnmv(:,), nlnmv(:,),
00253 c     o ilc(:,),inc(:,),imc(:,), ilnmc(:,), nlnmc(:,),
00254 c     o nindx(:,),konf(:,),icore(:,),ncore(:,),
00255 c     & occv(:,),unoccv(:,),)
00256 c     & ,occc(:,),unoccc(:,),)
00257 c     o nocc(:,),nunocc(:,),)
00258 c     real(8), allocatable::

```

```

00259 c      o plat(:,:),pos(:,:),z(:, ecore(:,:), symgg(:,,:) ! symgg=w(igrp),freq(:)
00260 c-----
00261
00262 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00263      do i=1,natom
00264          print *, ' iatom, spid= ',i,spid(i)
00265      enddo
00266 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00267
00268
00269
00270
00271 c--- Get maximums takao 18June03
00272      call getnemx(nbm,ebm,8,.true.) !8+1 th line of GWINO
00273
00274 c-----
00275 c      if (nclasse > mxclasse) stop ' hsfp0: increase mxclasse'
00276 c!!!! WE ASSUME iclass(iatom)= iatom !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
00277      if (nclasse /= natom ) stop ' hsfp0: nclasse /= natom ' ! We assume nclasse = natom.
00278      write(6,*) ' hsfp0: end of genallcf2'
00279 c
00280      call pshprt(30)
00281      pi = 4d0*datan(1d0)
00282      tpia = 2d0*pi/alat
00283
00284 c      call dinv33(plat,1,xxx,vol)
00285 c      call dinv33(plat,1,qat,vol)
00286 c      voltot = dabs(vol)*(alat**3)
00287      call minv33tp(plat,qat)
00288      voltot = abs(alat**3*tripl(plat,plat(1,2),plat(1,3)))
00289
00290      ifmlw(1) = iopen('MLWU',0,-1,0)
00291      ifmlwe(1)= iopen('MLWEU',0,-1,0)
00292      if (nspin == 2) then
00293          ifmlw(2) = iopen('MLWD',0,-1,0)
00294          ifmlwe(2)= iopen('MLWED',0,-1,0)
00295      endif
00296
00297 c>> read dimensions of wc,b,hb
00298      ifhbed = iopen('hbe.d',1,0,0)
00299      read (ifhbed,*) nprec,bmrecb,mrece,nlmtot,nqbzt, nband,mrece
00300      if (nprec == 4) stop 'hsfp0: b,hb in single precision'
00301
00302      call init_readEigen(ginv,nspin,nband,mrece) !initialization of readEigen
00303
00304 c --- get space group information -----
00305 c true class information in order to determine the space group -----
00306 c because the class in the generated GW file is dummy.(iclass(ibas)=ibas should be kept).
00307      open (102,file='CLASS')
00308      allocate(iclass(natom),invx(ngrp)
00309      & ,miat(natom,ngrp),tiat(3,natom,ngrp),shtvg(3,ngrp))
00310      write(6,*) ' --- Readingin CLASS info ---'
00311      do ibas = 1,natom
00312          read(102,*) ibasx, iclasst(ibas)
00313          write(6, "(2i10)") ibasx, iclasst(ibas)
00314      enddo
00315
00316 c Get space-group transformation information. See header of mptauof.
00317      call mptauof(symgg,ngrp,plat,natom,pos,iclassst
00318      o ,miat,tiat,invx,shtvg )
00319 c      write (*,*) 'tiat=', tiat(1:3,1:natom,invr),invr
00320
00321 c-----
00322      call pshprt(60)
00323
00324 c... Readin eigen functions
00325 c      ifev(1) = iopen('EVU', 0,0,mrece)
00326 c      if (nspin==2) ifev(2) = iopen('EVD', 0,0,mrece)
00327
00328 ! read EF from 'BNDS' if exists
00329      lbnds=.false.
00330      inquire(file='BNDS',exist=lbnds)
00331      if (lbnds) then
00332          write(*,*) 'Read EF from BNDS'
00333          ifh=ifile_handle()
00334          open(ifh,file='BNDS',status='old')
00335          read(ifh,*)ntmp,ef
00336          close(ifh)
00337      else ! lbnds
00338          call rx('you have to perform job_band in advance')
00339      endif
00340 c$$$c --- determine Fermi energy ef for given valn (legas case) or corresponding charge given by z and
      konf.
00341 c$$$! When esmr is negative, esmr is given automatically by efsimplef.
00342 c$$$      write(*,*) 'Calculate EF in efsimplef2a'
00343 c$$$      legas = .false.
00344 c$$$      call efsimplef2a(nspin,wibz,qibz,ginv,

```

```

00345 c$$$      i          nband,nqibz
00346 c$$$      i          ,konf,z,nl,natom,iclass,nclass
00347 c$$$      i          ,valn, legas, esmr,  !!! valn is input for legas=T, output otherwise.
00348 c$$$c
00349 c$$$      i          qbz,nqibz !index_qbz, n_index_qbz,
00350 c$$$      o          ,efnew)
00351 c$$$c
00352 c$$$c      write(6,*)' end of efsimple'
00353 c$$$      ef = efnew
00354 c$$$      endif ! lbnds
00355 c- check total ele number -----
00356      ntot = nocctotg2(nspin, ef,esmr, qbz,wbz, nband,nqibz) !wbz
00357      write(6,*)' ef      =' ,ef
00358      write(6,*)' esmr    =' ,esmr
00359      write(6,*)' valn    =' ,valn
00360      write(6,*)' ntot    =' ,ntot
00361
00362 c          ifcphi = iopen('CPHI',0,0,mrecb)
00363
00364      call init_read eigen2(mrecb,nlmt0,mrecg) !initialize m_read eigen
00365
00366 !c QPNT data
00367 ctm, 080222
00368 ! read QPNT from 'SYML' if exists
00369      lsyaml=.false.
00370      inquire(file='SYML',exist=lsyaml)
00371      if (lsyaml) then
00372          write(*,*)'Read k points for bands from SYML'
00373          lqall      = .false.
00374          laf        = .false.
00375          open(99,file='SYML',status='old')
00376          nline=0
00377          do i = 1,nlinex
00378              read(99,*,err=551,end=552)npin,qiin,qfin
00379              if (npin==0) exit
00380              nline = nline+1
00381              np(nline)=npin
00382              qi(1:3,nline)=qiin
00383              qf(1:3,nline)=qfin
00384 551          continue
00385      enddo
00386 552      continue
00387      if (nline.eq.nlinex) call rx('hmaxloc: too many lines in SYML')
00388      close(99)
00389      nq = 0
00390      do i = 1,nline
00391          nq = nq + np(i)
00392      enddo ! i
00393      allocate(q(3,nq),xq(nq))
00394      iq = 0
00395      xq=0d0
00396      gold=q(:,1)
00397      do i = 1,nline
00398          do j = 0,np(i)-1
00399              iq = iq + 1
00400              q(:,iq) = qi(:,i) + (qf(:,i)-qi(:,i))*dble(j)/dble(np(i)-1)
00401              if(iq>1) then
00402                  xq(iq)= xq(iq-1) + dsqrt( sum((q(:,iq)-gold)**2) )
00403              endif
00404              gold=q(:,iq)
00405          enddo ! j
00406      enddo ! i
00407      else ! lsyaml
00408          write(*,*)'Read k points for bands from GWinput'
00409          call getkeyvalue("GWinput","<QPNT>",unit=ifqpnt,status=ret)
00410          write(6,*)' ifqpnt ret=',ifqpnt,ret
00411 c
00412          lqall      = .false.
00413          laf        = .false.
00414          call readx(ifqpnt,10)
00415          read (ifqpnt,*) iqall,iaf
00416          if (iqall == 1) lqall = .true.
00417          if (iaf == 1)   laf = .true.
00418          call readx(ifqpnt,100)
00419 ctm 040622
00420          read (ifqpnt,*)
00421          read (ifqpnt,*)
00422
00423          if (lqall) then !all q-points case
00424              nq      = nqibz
00425              allocate(q(3,nq))
00426              call dcopy(3*nqibz,qibz,1,q,1)
00427          else
00428              call readx(ifqpnt,100)
00429              read (ifqpnt,*) nq
00430              allocate(q(3,nq))
00431              do      k = 1,nq

```

```

00432         read (ifqpnt,*) i,q(1,k),q(2,k),q(3,k)
00433         write(6,'(i3,f13.6)') i,q(1,k),q(2,k),q(3,k)
00434         enddo
00435     endif ! lqall
00436     close(ifqpnt)
00437     allocate(xq(nq))
00438     xq=0d0
00439     endif ! syml
00440 c
00441     nspinmx = nspin
00442     if (laf) nspinmx =1
00443 c-----
00444 c input parameters specific to MAXLOC
00445     call s_read_worb()
00446
00447
00448     do iclass2=1,nclass_mlwf
00449         write(*,*)'output:',iclassin(iclass2), nwf
00450         & ,trim(classname_mlwf(iclass2)),cbas_mlwf(1:nbasclass_mlwf(iclass2),iclass2)
00451     enddo
00452
00453     call s_cal_worb()
00454
00455     allocate (r0g(nphix,nwf), wphi(nphix,nwf))
00456
00457     r0g = 2d0
00458     wphi = 1d0
00459
00460
00461
00462     call wan_input(leout,lein,lbin,ieo_swt,iei_swt,
00463         & eomin,eomax,itout_i,itout_f,nbbelow,nbabove,
00464         & eimin,eimax,itin_i,itin_f,
00465         & nsc1,nsc2,conv1,conv2,alpha1,alpha2,rcut)
00466 c
00467
00468 cskino
00469     r_v=rcut
00470     call getkeyvalue("GWinput",'wan_tbcut_rcut',heps,default=r_v)
00471     call getkeyvalue("GWinput",'wan_tbcut_heps',heps,default=0.0d0)
00472     write(*,*) 'mloc.heps ', heps
00473 cekino
00474
00475
00476 cc --- read LDA eigenvalues
00477     ntq = nwf
00478     ntp0=ntq
00479 c     allocate(eqx(ntq,nq,nspin),eqx0(ntq,nq,nspin),eqt(nband))
00480 c     do      is = 1,nspin
00481 c     do      ip = 1,nq
00482 cc         iq      = idxk (q(1,ip),qbze,nqbze)
00483 cc         call rwdl (ifev(is), iq, nband, eqt) !direct access read b,hb and e(q,t)
00484 c         call readeval(q(1,ip),is,eqt)
00485 cc         write(6,*)' eqt=',eqt
00486 c         eqx0(1:ntq,ip,is) = eqt(itq(1:ntq))
00487 c         eqx (1:ntq,ip,is) = rydberg()*(eqt(itq(1:ntq))- ef)
00488 c     enddo
00489 c     enddo
00490 c     deallocate(eqt)
00491
00492 c --- info
00493     call winfo(6,nspin,nq,ntq,is,nbloch
00494         & ,0,0,nqbz,nqibz,ef,deltaw,alat,esmr)
00495
00496 c
00497     iii=ivsumxxx(irk,nqibz*ngroup)
00498     write(6,*) " sum of nonzero iirk=",iii, nqbz
00499
00500 c-----
00501 c debug:
00502 c     allocate(eqt(nband))
00503 c     do ip = 1,nqbz
00504 c         call readeval(qbz(1,ip),1,eqt)
00505 c         write(80,"('***',3f10.5)")qbz(:,ip)
00506 c         do is=1,nband
00507 c             write(80,"(i5,f12.6)")is,eqt(is)
00508 c         enddo
00509 c     enddo
00510
00511 c Rt vectors
00512     allocate (rt(3,nqbz),rt8(3,8,nqbz),qbz0(3,nqbz))
00513 c     write(6,"(a,9f9.4)")'qbas=',qbas
00514 c     write(6,"(a,9f9.4)")'plat=',plat
00515     call getrt(qbz,qbas,plat,n1,n2,n3,nqbz,
00516         o      rt,rt8,qbz0)
00517
00518 c b vectors

```



```

00519      call getbb(plat,alat,n1,n2,n3,
00520      o          nbb,wbb,wbbsum,bb)
00521
00522 c index for k and k+bb
00523      allocate (ku(3,nqbz),kbu(3,nbb,nqbz),ikbidx(nbb,nqbz))
00524
00525      call kbbindx(qbz,ginv,bb,
00526      d          nqbz,nbb,
00527      o          ikbidx,ku,kbu)
00528
00529
00530      allocate (iko_i(nqbz),iko_f(nqbz),
00531      &          iki_i(nqbz),iki_f(nqbz),
00532      &          ikbo_i(nbb,nqbz),ikbo_f(nbb,nqbz),
00533      &          ikbi_i(nbb,nqbz),ikbi_f(nbb,nqbz))
00534
00535 !! takao list eigen -----
00536      enwfmax =-1d9
00537      enwfmaxi=1d9
00538      allocate(eqt(1:nband))
00539      do is = 1,nspin
00540      do iq = 1,nqbz
00541          qxx = qbz(:,iq)
00542          call readeval(qxx,is,eqt)
00543          ini=1
00544          do i=1,nband
00545      c          write(6,*)'eqeq',eqt(i),eomin,eqt(nwf)
00546          if (eqt(i)>eomin) then
00547              inii=i
00548              exit
00549          endif
00550      enddo
00551      eeee = (eqt(nwf+inii-1)-ef)*rydberg()
00552      write(6,>('elist: q iq is nwfi nwfe e(nwf)= ',3f9.4,i5,i2,2i5,f10.3)) qxx,iq,is,inii,nwf+inii-1,
00553      eeee
00554      if (enwfmax < eeee) enwfmax = eeee
00555      if (enwfmaxi > eeee) enwfmaxi = eeee
00556      enddo
00557      deallocate(eqt)
00558      write(6,>('elist max enwf enwfmaxi=',2f13.5)) enwfmax,enwfmaxi
00559      call getkeyvalue("GWinput","wan_out_emax_auto",leauto,default=.false.)
00560      if(leauto) then
00561          eomax= enwfmax + 1d-4
00562          write(6,*)
00563          write(6,>(' WE USE wan_out_emax_auto on ==> +1d-3 ==> eomax=',3f13.5)) eomax
00564      endif
00565      call getkeyvalue("GWinput","wan_in_emax_auto",leinauto,default=.false.)
00566      if(leinauto) then
00567          eimax= enwfmaxi + 1d-4
00568          write(6,*)
00569          write(6,>(' WE USE wan_in_emax_auto on ==> +1d-3 ==> eimax=',3f13.5)) eimax
00570      endif
00571
00572 c      stop 'qqqqqqqqqqqqqqqqqq'
00573
00574 !! ixc = 1 -----
00575      if (ixc.eq.1) then
00576      do is = 1,nspin
00577          call ewindow(is,ieo_swt,iei_swt,itout_i,itout_f,itin_i,itin_f,
00578      i          eomin,eomax,eimin,eimax,ef,qbz,ikbidx,
00579      i          nbbelow,nbabove,
00580      d          nqbz,nbb,nband,nwf,nspin,
00581      o          iko_i,iko_f,iki_i,iki_f,
00582      o          ikbo_i,ikbo_f,ikbi_i,ikbi_f,
00583      o          iko_ixs(is),iko_fxs(is),noxs(is),
00584      o          leout,lein)
00585      enddo
00586
00587 c write bb vectors to 'BBVEC'
00588      call writebb(ifbb,wbb(1:nbb),bb(1:3,1:nbb),
00589      i          ikbidx,ku,kbu,
00590      i          iko_ixs,iko_fxs,noxs,
00591      d          nspin,nqbz,nbb)
00592
00593 ctm, 060923 !!!
00594      ifwand = iopen('wan.d',1,-1,0)
00595      iko_ix = iko_ixs(1)
00596      iko_fx = iko_fxs(1)
00597      if (nspin.eq.2) then
00598          if (iko_ixs(2).lt.iko_ix) iko_ix = iko_ixs(2)
00599          if (iko_fxs(2).gt.iko_fx) iko_fx = iko_fxs(2)
00600      endif
00601      write(ifwand,*)nqbz,nwf,iko_ix,iko_fx
00602      write(ifwand,*)nspin
00603      do is = 1,nspin
00604          write(ifwand,*)nqbz,nwf,iko_ixs(is),iko_fxs(is)

```

```

00605         enddo
00606         isx = iclose('wan.d')
00607         call rx0('hmaxloc: ixc=1 ok')
00608     endif
00609
00610 !! loop over spin -----
00611     do 1000 is = 1,nspin
00612         write(*,*)'is =',is,' out of',nspin
00613 c energy window
00614         call ewindow(is,ieo_swt,iei_swt,itout_i,itout_f,itin_i,itin_f,
00615             i          eomin,eomax,eimin,eimax,ef,qbz,ikbidx,
00616             i          nbbelow,nbabove,
00617             d          nqbz,nbb,nband,nwf,nspin,
00618             o          iko_i,iko_f,iki_i,iki_f,
00619             o          ikbo_i,ikbo_f,ikbi_i,ikbi_f,
00620             o          iko_ix,iko_fx,nox,
00621             o          leout,lein)
00622 !         call chk_ewindow(ifbb,is,nspin,nqbz,nbb,iko_ix,iko_fx)
00623
00624 cccccccccccccccccccccccc
00625 c         do iq=1,nqbz
00626 c         write(6,*)"iiii: iq and internal window region=',3i5")iq,iki_i(iq),iki_f(iq)
00627 c         enddo
00628 cccccccccccccccccccccccc
00629
00630 c read uu-matrix
00631         allocate (uumat(iko_ix:iko_fx,iko_ix:iko_fx,nbb,nqbz))
00632         call readuu(is,iko_ix,iko_fx,ikbidx,
00633             d          nqbz,nbb,
00634             o          uumat)
00635         call chkuu(is,iko_ix,iko_fx,ikbidx,uumat,
00636             d          nqbz,nbb)
00637
00638 !! step 1 -- choose Hilbert space -- determine cnk
00639         write(*,*)'Step 1: Hilbert space branch'
00640         write(6,*)" iko_ix iko_fx=',iko_ix,iko_fx
00641         allocate (amnk(iko_ix:iko_fx,nwf,nqbz),
00642             &      upu(iko_ix:iko_fx,iko_ix:iko_fx,nbb,nqbz),
00643             &      cnk(iko_ix:iko_fx,nwf,nqbz),
00644             &      cnk2(iko_ix:iko_fx,nwf,nqbz),
00645             &      omgik(nqbz))
00646 ! amnk appered in Eq.22 in Ref.II. <psi|Gaussian>
00647         call init_unkg(is,qbz,ginv,ef,lein,
00648             i          iko_ix,iko_fx,iko_i,iko_f,
00649             i          iki_i,iki_f,
00650             d          nwf,nband,nqbz,
00651             o          amnk,cnk)
00652 !         call chk_amnkweight(qbz,iko_ix,iko_fx,amnk,
00653 !             &      nqbz,nwf,nband,nlmt)
00654 !         call chk_cnkweight(qbz,iko_ix,iko_fx,cnk,
00655 !             &      nqbz,nwf,nband,nlmt)
00656         do isc = 1,nscl
00657             do iq = 1,nqbz
00658                 call dimz(lein,iko_i(iq),iko_f(iq),iki_i(iq),iki_f(iq),
00659                     ndz,nin)
00660                 if (nwf.gt.nin) then
00661                     if (ndz.lt.1) call rx('ndz < 1')
00662 c (1-2) <u_mnk | P_k+b | u_nk>
00663                     call getupu(isc,
00664                         i          uumat(:, :, :, iq),cnk,
00665                         i          lein,alpha1,iq,ikbidx(:, iq),
00666                         i          iko_ix,iko_fx,
00667                         i          iko_i(iq),iko_f(iq),
00668                         i          iki_i(iq),iki_f(iq),
00669                         i          ikbo_i(:, iq),ikbo_f(:, iq),
00670                         i          ikbi_i(:, iq),ikbi_f(:, iq),
00671                         d          nwf,nbb,nqbz,
00672                         u          upu(:, :, :, iq))
00673 c (1-3) Zmn(k) > phi,eval
00674                     allocate (zmn(ndz,ndz),evecc(ndz,ndz),eval(ndz))
00675                     call getzmn(upu(:, :, :, iq),wbb,lein,
00676                         i          iko_ix,iko_fx,
00677                         i          iko_i(iq),iko_f(iq),
00678                         i          iki_i(iq),iki_f(iq),
00679                         d          nwf,nbb,nqbz,ndz,
00680                         o          zmn)
00681
00682                     call chk_hm(zmn,ndz)
00683                     call diag_hm(zmn,ndz,eval,evecc)
00684                     call new_cnk(cnk(:, :, iq),evecc,iq,
00685                         i          iko_ix,iko_fx,
00686                         i          iko_i(iq),iko_f(iq),
00687                         i          iki_i(iq),iki_f(iq),
00688                         d          nwf,ndz,
00689                         o          cnk2(:, :, iq))
00690 c (1-3) w_I(k) eq.(18)
00691                     call chk_eval(wbb,eval,nbb,ndz)

```

```

00692      call get_omgik(wbb,eval,
00693      i          iko_i(iq),iko_f(iq),
00694      i          iki_i(iq),iki_f(iq),
00695      d          nbb,nwf,ndz,
00696      o          omgik(iq))
00697      deallocate (zmn,evecc,eval)
00698      else
00699      omgik(iq) = 0d0
00700      cnk2(:, :, iq) = cnk(:, :, iq)
00701 c end if (ndz>1)
00702      endif
00703 c end of iq-loop
00704      enddo
00705 c (1-5) w_I(k) > Omega_I eq.(11)
00706      omgi = sum(omgik(:)*wbz(:))
00707 c (1-6) check self-consistency
00708      write(*, '('#SC-loop, conv.', i5, d13.5)')isc,omgi
00709      if (isc.ge.2) then
00710          domgi = dabs((omgiold - omgi) / omgiold)
00711          if (domgi .lt. conv1) then
00712              write(*,*) 'step1: converged!'
00713              goto 810
00714          endif
00715      endif
00716 c update
00717      omgiold = omgi
00718      cnk      = cnk2
00719 c end of self-consistent loop
00720      enddo
00721      write(*,*) 'step1: not converged'
00722 810 continue
00723      deallocate(upu,cnk2)
00724
00725 c      call chk_cnkweight(qbz,iko_ix,iko_fx,cnk,
00726 c      &      nqbz,nwf,nband,nlmt0)
00727
00728 !! NOTE: cnk is the final results of step 1
00729 !!      cnk(iko_ix:iko_fx,nwf,nqbz)
00730 !!      cnk(iko_i(iq):iko_f(iq),nwf,iq) gives nwf-dimensional space.
00731 !!      step 1 (minimization of Omega_I)
00732
00733
00734 !! === step 2 -- localize Wannier fn. =====
00735      write(*,*) 'Step 2: Wannier fn. branch'
00736
00737      allocate (mmn(nwf,nwf,nbb,nqbz),mmn0(nwf,nwf,nbb,nqbz),
00738      &      umnk(nwf,nwf,nqbz),
00739      &      rmn(nwf,nwf),amn(nwf,nwf),smn(nwf,nwf),
00740      &      rn(3,nwf),qn(nwf),tmn(nwf,nwf),dwmn(nwf,nwf),
00741      &      eunk(nwf,nqbz))
00742
00743 !! (2-0) construct initial u~ from u
00744 !! eunk(= e-) of {H-}_mn: eigenvalue within the nwf-dimentional Hilbert space
00745      call diag_eunk(is,qbz,
00746      i          iko_ix,iko_fx,iko_i,iko_f,
00747      d          nband,nwf,nqbz,
00748      u          cnk,
00749      o          eunk)
00750
00751 !      call chk_cnkweight(qbz,iko_ix,iko_fx,cnk,
00752 !      &      nqbz,nwf,nband,nlmt0)
00753 !
00754 ! check ortho-normality of u~'s
00755 !      call chk_cnk(cnk,
00756 !      i          iko_ix,iko_fx,iko_i,iko_f,
00757 !      d          nband,nwf,nqbz)
00758 !
00759 ! check: eunk vs. KS energy
00760 !      call chk_eunk(is,qbz,eunk,ef,
00761 !      d          nqbz,nband,nwf)
00762
00763 !! (2-1) initial: uumat -> M_mn(0) Eq.58 in Ref.[1]
00764      call init_mmn(cnk,uumat,ikbidx,
00765      i          iko_ix,iko_fx,iko_i,iko_f,ikbo_i,ikbo_f,
00766      d          nwf,nqbz,nbb,
00767      o          mmn0)
00768
00769 !! (2-2) initial U
00770 !!      umnk= U(m,n) = ( A S^{-1/2} )_mn. See Eq.23 in Ref.II.
00771      call init_umnk(amnk,cnk,
00772      i          iko_ix,iko_fx,iko_i,iko_f,
00773      d          nwf,nqbz,
00774      o          umnk)
00775
00776 !      call chk_umn(cnk,umnk,qbz,
00777 !      i          iko_ix,iko_fx,iko_i,iko_f,
00778 !      d          nwf,nqbz,nband,nlmt0)

```

```

00779
00780      call updt_mmn(umnk,mmn0,ikbidx,
00781                  d      nwf,nqbz,nbb,
00782                  u      mmn)
00783
00784 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00785 c      do i=1,nbb
00786 c          write(1106+is,"(a,i4,13f13.5)"')'bbbb',i,bb(1:3,i),wbb(i)
00787 c      enddo
00788 c      do i=1,nqbz
00789 c          write(1106+is,"(a,i4,13f13.5)"')'www',i,wbz(i)
00790 c      enddo
00791 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00792
00793
00794
00795 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00796 c      do isc = 1, nsc2
00797 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00798 c      mmn=mmn+(0d0,1d-8)
00799 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00800
00801 c <r_n> ([1] eq.31)
00802      call get_rn(mmn,bb,wbb,wbz,
00803                  d      nwf,nqbz,nbb,
00804                  o      rn)
00805 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00806 c      do i=1,nwf
00807 c          write(1106+is,"(a,3f13.5)"')'rrrrrn',rn(1:3,i)
00808 c      enddo
00809 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00810
00811 c      do iq = 1,nqbz
00812 c          dwmn = (0d0,0d0)
00813 c      do ibb = 1,nbb
00814 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00815 c      do i = 1,nwf
00816 c      do j = 1,nwf
00817 c          write(1106+is,"(a,4i5,2f13.3)"')' mmmmmmm ',i,j,ibb,iq,mmn(i,j,ibb,iq)+(0d0,0.0001)
00818 c      enddo
00819 c      enddo
00820 c      do i = 1,nwf
00821 c      do j = 1,nwf
00822 c          write(1106+is,"(a,4i5,2f13.3)"')' nnnnnnn ',i,j,ibb,iq,mmn0(i,j,ibb,iq)+(0d0,0.0001)
00823 c      enddo
00824 c      enddo
00825 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00826
00827 c (2-3) A[R] matrix
00828      call getrmn(mmn(:, :, ibb, iq),
00829                  d      nwf,
00830                  o      rmn)
00831      call getamn(rmn,
00832                  d      nwf,
00833                  o      amn)
00834
00835 c (2-4) S[T] matrix
00836      call gettmn(rn,mmn(:, :, ibb, iq),bb(:, ibb),
00837                  d      nwf,
00838                  o      qn,tmn)
00839      call getsmn(tmn,
00840                  d      nwf,
00841                  o      smn)
00842
00843 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00844 c      smn=0d0
00845 c      amn=0d0
00846 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00847
00848 c DW(k) ([1] eq.57)
00849      dwmn(:, :) = dwmn(:, :)
00850      &      + wbb(ibb) * (amn(:, :) - smn(:, :)) * alpha2 / wbbsum
00851
00852 c end of ibb-loop
00853 c      enddo
00854
00855 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00856 c      dwmn=0d0
00857 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00858 c (2-5) DW(k) -> U(k) ([1] eq.60)
00859      call updt_uk(dwmn,
00860                  d      nwf,
00861                  u      umnk(:, :, iq))
00862 c      call chk_um(umnk(:, :, iq),nwf)
00863
00864 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00865 c      do i = 1,nwf

```

```

00866 c      do j = 1,nwf
00867 c          write(1106+is,"(a,3i5,2f13.3)")' zzzzz',iq,i,j,umnk(i,j,iq)
00868 c      enddo
00869 c      enddo
00870 ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
00871
00872 c end of iq-loop
00873 c      enddo
00874
00875
00876
00877 c update Mmn ([1] eq.61)
00878 c      call updt_mmn(umnk,mmn0,ikbidx,
00879 c          d          nwf,nqbz,nbb,
00880 c          u          mmn)
00881
00882 c (2-6) Omg_I, Omega_D and Omega_OD ([1] eq.34,35,36)
00883 c      call getomg(mmn,rn,bb,bbb,wbz,
00884 c          d          nwf,nqbz,nbb,
00885 c          o          omgi,omgd,omgod,omgdod,omgidod)
00886
00887 c check self-consistency
00888 c      write(*,*)'#SC-loop, conv.',isc,omgdod
00889 c      write(*,950)'Omg: I, OD, D ',omgi,omgod,omgd
00890 c      write(*,")'#SC-loop, conv.',i6,e13.5,' Omg:I,OD,_D= ',3f17.10)")
00891 c      & isc,omgdod,omgi,omgod,omgd
00892 c      if (isc.ge.2) then
00893 c          domgdod = dabs(omgdodold - omgdod) / omgdodold)
00894 c          if (domgdod.lt. conv2) then
00895 c              write(*,*) 'step2: converged!'
00896 c              goto 820
00897 c          endif
00898 c      endif
00899 c      omgdodold = omgdod
00900
00901 c end of self-consistent loop
00902 c      enddo
00903 c      write(*,*)'step2: not converged'
00904 c      820 continue
00905
00906 !      call chk_dnk(is,eunk,qbz,
00907 !          i          umnk,cnk,
00908 !          i          iko_ix,iko_fx,iko_i,iko_f,
00909 !          d          nband,nwf,nqbz)
00910 !
00911 !      call chk_umn(cnk,umnk,qbz,
00912 !          i          iko_ix,iko_fx,iko_i,iko_f,
00913 !          d          nwf,nqbz,nband,nlmt0)
00914
00915 c output
00916 c      write(*,*)"----- wlxloc isp =",is
00917 c      call wmaxloc(ifmlw(is),ifmlwe(is),
00918 c          i          qbz,umnk,cnk,eunk,
00919 c          i          iko_ix,iko_fx,iko_i,iko_f,
00920 c          d          nwf,nqbz,nband,nlmt0, is)
00921 c      call writeomg(is,mmn,rn,bb,bbb,wbz,tpia,
00922 c          d          nwf,nqbz,nbb)
00923 c 070824
00924 c      call getkeyvalue("GWinput","wan_write_rmn",lrnm,default=.false.)
00925 c      if (lrnm)
00926 c          & call writermn(is,mmn,bb,bbb,qbz,qbz0,wbz,rt,
00927 c          d          nwf,nqbz,nbb,n1,n2,n3)
00928 c 070830
00929 c      call getkeyvalue("GWinput","wan_write_mmn",lmmn,default=.false.)
00930 c      if (lmmn)
00931 c          & call writemmn(is,mmn,bb,bbb,qbz,wbz,rt,
00932 c          d          nwf,nqbz,nbb,n1,n2,n3)
00933
00934 c      deallocate(uumat,amnk,omgik,mmn,mmn0,
00935 c          &          rmn,amn,smn,rn,qn,tmn,dwmn)
00936
00937
00938 !! step 3 -- reduced Hamiltonian -----
00939 c      write(*,*)'Step 3: reduced Hamiltonian branch'
00940 c open file
00941 c      if (is .eq. 1) then
00942 c          ifbnd = iopen('bnds.maxloc.up',1,-1,0)
00943 c          iftb = iopen('bnds.tb.up',1,-1,0)
00944 c      else
00945 c          ifbnd = iopen('bnds.maxloc.dn',1,-1,0)
00946 c          iftb = iopen('bnds.tb.dn',1,-1,0)
00947 c      endif
00948 c      write(ifbnd,*)nq
00949 c      write(ifbnd,*)nwf
00950 c      write(iftb,*)nq
00951 c      write(iftb,*)nwf
00952 c allocate

```

```

00953         if(allocated(hrotk)) deallocate(hrotk,hrotkp,evecc,eval)
00954         allocate (hrotk(nwf,nwf,nqbz), ! hrotr(nwf,nwf,nqbz),
00955         o      hrotkp(nwf,nwf),evecc(nwf,nwf),eval(nwf))
00956 c for small Hamiltonian
00957 call getkeyvalue("GWinput","wan_small_ham",lsh,default=.false.)
00958 if (lsh) then
00959     call getkeyvalue("GWinput","wan_nsh1",nsh1, default=1 )
00960     call getkeyvalue("GWinput","wan_nsh2",nsh2, default=2 )
00961     write(*,*) 'SmallHam on',nsh1,nsh2
00962     nsh = nsh2 - nsh1 + 1
00963     if (is .eq. 1) then
00964         ifsh = iopen('bnds.sh.up',1,-1,0)
00965     else
00966         ifsh = iopen('bnds.sh.dn',1,-1,0)
00967     endif
00968     write(ifsh,*)nq
00969     write(ifsh,*)nsh
00970     allocate (hrotkps(nsh,nsh),eveccs(nsh,nsh),evals(nsh))
00971     endif
00972 c (3-1) ~H(k) -> Hrot(k): note eunk is eigenvalues in the basis of cnk
00973     call rot_hmnk(umnk,eunk,
00974     d      nwf,nqbz,
00975     o      hrotk) !rotated Hamiltonian in MLW basis.
00976 c (3-2) Hrot_mn(R)
00977     if(allocated(irws)) deallocate(irws,rws,drws)
00978     allocate(irws(n1*n2*n3*8),rws(3,n1*n2*n3*8),drws(n1*n2*n3*8))
00979     call wigner_seitz(alat,plat,n1,n2,n3,nrws,rws,irws,drws)
00980     if(allocated(hrotr)) deallocate(hrotr)
00981     allocate(hrotr(nwf,nwf,nrws)) !real space Hamiltonian in Wannier funciton basis
00982 c     write(*,*) 'xxxxxxxxx1'
00983     if (ixc.eq.2) then
00984         call get_hrotr_ws(hrotk,qbz,wbz,
00985         i      rws,irws,drws,
00986         d      nwf,nqbz,nrws,
00987         o      hrotr)
00988 c     skino
00989 c     write hrotr and *rws
00990         if (is .eq. 1) then
00991             ifh = iopen('hrotr.up',1,-1,0)
00992         else
00993             ifh = iopen('hrotr.dn',1,-1,0)
00994         endif
00995
00996         call write_hrotr(ifh, hrotr,
00997         i      rws,irws,drws,
00998         d      nwf,nrws )
00999
01000         close (ifh)
01001 c     ekino
01002 c     skino
01003     else if (ixc.eq.3) then
01004         if (is .eq. 1) then
01005             filename='hrotr.up'
01006         else
01007             filename = 'hrotr.dn'
01008         endif
01009         call read_hrotr(filename,nwf,nrws,
01010         o      hrotr)
01011         if (is .eq. 1) then
01012             ifh = iopen('hrotr.cut.up',1,-1,0)
01013         else
01014             ifh = iopen('hrotr.cut.dn',1,-1,0)
01015         endif
01016         allocate(hrotrcut(nwf,nwf,nrws))
01017         call make_hrotrcut( hrotr,
01018         i      rws,irws,drws,
01019         i      rcut,heps,
01020         d      nwf,nrws,
01021         o      hrotrcut )
01022         call write_hrotr(ifh, hrotrcut,
01023         i      rws,irws,drws,
01024         d      nwf,nrws )
01025         close (ifh)
01026         deallocate(hrotrcut)
01027 c     ekino
01028     endif
01029 c     write(*,*) 'xxxxxxxxx2'
01030
01031 !! -----
01032 !! k-point mesh
01033     call get_nqbze(nqbz,nqbze)
01034     allocate(qbze(3,nqbze))
01035     call get_qbze(qbz,nqbz,
01036     o      qbze,nqbze)
01037     write(ifmlw(is))nqbze,nwf
01038     write(ifmlwe(is))nqbze,nwf
01039     do iq = 1,nqbze

```

```

01040 c      write(*,*)'goto get_hrotkp_ws iq=',iq,nqbze
01041      call get_hrotkp_ws(hrotr,rws,drws,irws,qbze(:,iq), !july2014 qbz->qbze
01042          d      nwf,nqbz,nrws,
01043          o      hrotkp)
01044      call diag_hm(hrotkp,nwf,eval,evecc)
01045      call wmaxloc_diag(ifmlw(is),ifmlwe(is),
01046          i      iq,qbze(1:3,iq),umnk,cnk,eunk,evecc,eval,
01047          i      iko_ix,iko_fx,iko_i,iko_f,
01048          d      nwf,nqbz)
01049      enddo
01050 c      write(6,*)'eeeeeeeeee'
01051      deallocate(qbze)
01052 ccc      write(*,990)'iq =',iq,qbze(1:3,iq)
01053 cc      if (iq.le.nqbz) then
01054 cc          do iband = 1,nwf
01055 cc              e1 = (eval(iband) -ef)*rydberg()
01056 cc              e2 = (eunk(iband,iq)-ef)*rydberg()
01057
01058
01059 !! -----
01060 c --- Readin nlam index
01061      ifoc = iopen('@MNLA_CPHI',1,0,0)
01062      ldim2 = nlmto
01063      read(ifoc,*)
01064      if(allocated(m_indx)) deallocate(m_indx,n_indx,l_indx,ibas_indx,ibasiwf)
01065      allocate(m_indx(ldim2),n_indx(ldim2),l_indx(ldim2),ibas_indx(ldim2))
01066      do ix = 1,ldim2
01067          read(ifoc,*)m_indx(ix),n_indx(ix),l_indx(ix),ibas_indx(ix),ixx
01068          if(ixx/=ix) call rx('failed to readin @MNLA_CPHI')
01069      enddo
01070      ix = iclose('@MNLA_CPHI')
01071      allocate(ibasiwf(nwf))
01072      do iw=1,nwf
01073          ibasiwf(iwf) = ibas_indx(iphil,iwf))
01074      enddo
01075
01076
01077 !! write HrotRS
01078      ifh=ifile_handle()
01079      if(is==1) open(ifh,file='HrotRS.up',form='unformatted')
01080      if(is==2) open(ifh,file='HrotRS.dn',form='unformatted')
01081      write(ifh)alat,plat,natom
01082      write(ifh)pos
01083      write(ifh)ef
01084      write(ifh)nwf,nrws,n1,n2,n3
01085      write(ifh) irws,rws,hrotr, ibasiwf
01086      close(ifh)
01087
01088      ifh = ifile_handle()
01089
01090      call write_hopping_output(is, ifh, hrotr,
01091          &      rws,irws,alat,plat,qlat,pos,natom,
01092          &      ibasiwf, nwf,nrws,spid , m_indx, l_indx,
01093          &      nphix, iphi, ldim2)
01094
01095      close(ifh)
01096
01097
01098 !! other k-points
01099      write(ifbnd,*)ef,' ef'
01100      write(iftb,*)ef,' ef'
01101      if (lsh) write(ifsh,*)ef,' ef'
01102      allocate(evall(nwf,nq),eval3(nwf,nq))
01103      if(lsh) allocate(eval2(nwf,nq))
01104      do iq = 1,nq
01105 c      write(6,*)' got get_hrotkp_ws iq =',iq
01106 c (3-3) Hrot_mn(k')
01107      call get_hrotkp_ws(hrotr,rws,drws,irws,q(:,iq),
01108          d      nwf,nqbz,nrws,
01109          o      hrotkp)
01110 c (3-4) diagonalize
01111      call diag_hm(hrotkp,nwf,eval,evecc)
01112      eval1(1:nwf,iq)=eval
01113 c (3-4) diagonalize -- Small Hamiltonian --
01114      if (lsh) then
01115          hrotkps(1:nsh,1:nsh) = hrotkp(nsh1:nsh2,nsh1:nsh2)
01116          call diag_hm(hrotkps,nsh,evals,eveccs)
01117          write(ifsh,*)'iq =',iq
01118          write(ifsh,990)q(1:3,iq)
01119          eval2(1:nsh,iq)= evals(1:nsh)
01120      endif
01121 c (3-3) Hrot_mn(k') -- Tight-binding ---
01122      call get_hrotkp_tb_ws(rcut,plat,alat,
01123          i      hrotr,rws,drws,irws,q(:,iq), ibasiwf,pos,natom,
01124          d      nwf,nqbz,nrws,
01125          o      hrotkp)
01126 c      (3-4) diagonalize -- Tight-binding --

```





```

01214      do iwf=1,nwf
01215      do ibas=1,nbas
01216          write(*,*)ibas,iwf,wbas(ibas,iwf)
01217      enddo
01218      write(*,*)
01219      enddo
01220      write(*,*)'*** ibas,wbas'
01221      do ibas=1,nbas
01222          w = 0d0
01223          do iwf=1,nwf
01224              w = w + wbas(ibas,iwf)
01225          enddo
01226          write(*,*)ibas,w
01227      enddo
01228
01229      deallocate(cphil,cphi2,wbas,m_indx,l_indx,n_indx,ibas_indx)
01230      ix = iclose('@MNLA_CPHI')
01231
01232      end
01233  c -----
01234  subroutine chk_cnkweight(qbz,iko_ix,iko_fx,cnk,
01235      & nqbz,nwf,nband,nlmt0)
01236      use m_readqg
01237      use m_readeigen
01238      implicit real*8(a-h,o-z)
01239
01240      complex(8) :: cnk(iko_ix:iko_fx,nwf,nqbz)
01241      complex(8),allocatable:: cphil(:,cphi2(:,
01242      real(8) :: qbz(3,nqbz),q(3),quu(3)
01243      real(8),allocatable:: wbas(:,
01244      integer(4),allocatable::
01245      & m_indx(:),n_indx(:),l_indx(:),ibas_indx(:)
01246
01247  c --- Readin nlam index
01248      ifoc = iopen('@MNLA_CPHI',1,0,0)
01249      ldim2 = nlmt0
01250      read(ifoc,*)
01251      allocate(m_indx(ldim2),n_indx(ldim2),l_indx(ldim2),ibas_indx(ldim2))
01252      do ix =1,ldim2
01253          read(ifoc,*)m_indx(ix),n_indx(ix),l_indx(ix),ibas_indx(ix),ixx
01254          if(ixx/=ix) call rx('failed to readin @MNLA_CPHI')
01255      enddo
01256
01257      nbas = ibas_indx(nlmt0)
01258      allocate(cphil(nlmt0,nband),cphi2(nlmt0,nwf),wbas(nbas,nwf))
01259      wbas = 0d0
01260      cphi2=0d0
01261
01262      do iq = 1,nqbz
01263          q = qbz(:,iq)
01264          call readcphi(q,nlmt0,1,quu,cphil)
01265
01266      do iwf=1,nwf
01267          do ib=iko_ix,iko_fx
01268              cphi2(:,iwf) = cphi2(:,iwf) + cphil(:,ib)*cnk(ib,iwf,iq)
01269          enddo
01270      enddo
01271
01272      enddo ! iq
01273
01274      do iwf=1,nwf
01275          do ia=1,nlmt0
01276              ibas = ibas_indx(ia)
01277              wbas(ibas,iwf) = wbas(ibas,iwf) +
01278      & conjg(cphi2(ia,iwf))*cphi2(ia,iwf)
01279          enddo ! ia
01280      enddo ! iwf
01281      wbas = wbas / dble(nqbz*nqbz)
01282
01283      write(*,*)'*** ibas,iwf,wbas'
01284      do iwf=1,nwf
01285      do ibas=1,nbas
01286          write(*,*)ibas,iwf,wbas(ibas,iwf)
01287      enddo
01288      write(*,*)
01289      enddo
01290
01291      write(*,*)'*** ibas,wbas'
01292      do ibas=1,nbas
01293          w = 0d0
01294          do iwf=1,nwf
01295              w = w + wbas(ibas,iwf)
01296          enddo
01297          write(*,*)ibas,w
01298      enddo
01299      deallocate(cphil,cphi2,wbas,m_indx,l_indx,n_indx,ibas_indx)
01300      ix = iclose('@MNLA_CPHI')

```

```

01301         end
01302 c-----
01303     subroutine chk_umn(cnk,umnk,qbz,
01304 i                     iko_ix,iko_fx,iko_i,iko_f,
01305 d                     nwf,nqbz,nband,nlmt0)
01306     use m_readqg
01307     use m_readeigen
01308
01309     implicit real*8(a-h,o-z)
01310
01311     complex(8) :: cnk(iko_ix:iko_fx,nwf,nqbz),
01312 &               dnk(iko_ix:iko_fx,nwf,nqbz),
01313 &               umnk(nwf,nwf,nqbz)
01314     real(8) :: qbz(3,nqbz)
01315     integer(4) :: iko_i(nqbz),iko_f(nqbz)
01316
01317     dnk = (0d0,0d0)
01318     do iq = 1,nqbz
01319         do imp = iko_i(iq),iko_f(iq)
01320             do in = 1,nwf
01321                 do im = 1,nwf
01322                     dnk(imp,in,iq) = dnk(imp,in,iq)
01323 &               + umnk(im,in,iq) * cnk(imp,im,iq)
01324             enddo ! im
01325         enddo ! in
01326     enddo ! imp
01327 enddo ! iq
01328
01329     call chk_cnkweight(qbz,iko_ix,iko_fx,dnk,
01330 &               nqbz,nwf,nband,nlmt0)
01331
01332     end

```

## 4.43 /home/takao/ecalj/lm7K/run\_arg File Reference

### 4.44 run\_arg

```

00001 ### bash subroutine used in gwsc and so on. ###
00002 ### See fpgw/exec/gwsc for usage
00003 # T.Kotani Jan.2015
00004 # SeungWoo Jang Sep.2014
00005 echo_run="" # standard
00006 serial_run="" # standard
00007 #echo_run="aprun" # cray
00008 #serial_run="aprun" # cray
00009 function run_arg
00010 {
00011     local argin=$1
00012     local MPI_SIZE=$2
00013     local nfpgw=$3
00014     local command=$4
00015     local output=$5
00016     local TARGET=${@:6:($#-2)}
00017     local mpi_run="mpirun -np $MPI_SIZE" # standard
00018     #local pi_run="aprun -n $LSB_PROCS -d $LSB_CPUS -N $LSB_PPN" # cray
00019     $echo_run echo -n 'OK! --> Start'
00020     $echo_run echo $argin > _IN_
00021     if [ $MPI_SIZE == '0' ]; then
00022         $echo_run echo " echo $argin | $nfpgw$command $TARGET > $output "
00023         $serial_run $nfpgw$command $TARGET < _IN_ > $output
00024     else
00025         $echo_run echo " echo $argin | mpirun -np $MPI_SIZE $nfpgw$command $TARGET > $output "
00026         $mpi_run $nfpgw$command $TARGET < _IN_ > $output
00027     fi
00028     if [ $? != 0 ]; then
00029         $echo_run echo Error in $command input_arg=$argin. See OutputFile=$output
00030         exit 10
00031     fi
00032 }
00033
00034 echo "NOTE: Use run_arg defined in $nfpgw/run_arg"

```