Contents

1 Routine/Function Prologues

1.0.1 writelinen (Source File: writelinen.f90)

INTERFACE:

```
Subroutine writelinen
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Writes the linearisation energies for all APW and local-orbital functions to the file `LINENGY.OUT`.

REVISION HISTORY:

```
Created February 2004 (JKD)
```

1.0.2 writeefg (Source File: writeefg.f90)

INTERFACE:

```
Subroutine writeefg
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Computes the electric field gradient (EFG) tensor for each atom, α, and writes it to the file `EFG.OUT` along with its eigenvalues. The EFG is defined by

$$V_{ij}^{\alpha} \equiv \left.\frac{\partial^2 V_{\rm C}'(\mathbf{r})}{\partial \mathbf{r}_i \partial \mathbf{r}_j}\right|_{\mathbf{r}=\mathbf{r}_{\alpha}},$$

where $V_{\rm C}'$ is the Coulomb potential with the $l = m = 0$ component removed in each muffin-tin. The derivatives are computed explicitly using the routine `gradrfmt`.

REVISION HISTORY:

```
Created May 2004 (JKD)
Fixed serious problem, November 2006 (JKD)
```

1.0.3 rhoplot (Source File: rhoplot.f90)

INTERFACE:

```
Subroutine rhoplot
```

USES:

```
Use modinput
Use modmain
use modplotlabels
use modmpi, only : rank
```

DESCRIPTION:

Outputs the charge density and the charge density gradients (modulus) read in from `STATE.OUT`, for 1D, 2D or 3D plotting.

REVISION HISTORY:

```
Created June 2003 (JKD)
Density gradients are added, March 2011 (DIN)
```

1.0.4 symrvfir (Source File: symrvfir.f90)

Subroutine symrvfir (ngv, rvfir) **USES:**

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
ngv   : number of G-vectors to be used for the Fourier space rotation
        (in,integer)
rvfir : real interstitial vector function (inout,real(ngrtot,ndmag))
```

DESCRIPTION:

Symmetrises a real interstitial vector function. See routines `symrvf` and `symrfir` for details.

REVISION HISTORY:

```
Created July 2007 (JKD)
```

1.0.5 atom (Source File: atom.f90)

INTERFACE:

```
Subroutine atom (ptnucl, zn, nst, n, l, k, occ, xctype, xcgrad, nr, &
&  r, eval, rho, vr, rwf, mtnr, dirac_eq)
```

USES:

```
Use modxcifc
use modinput
```

INPUT/OUTPUT PARAMETERS:

```
ptnucl : .true. if the nucleus is a point particle (in,logical)
zn     : nuclear charge (in,real)
nst    : number of states to solve for (in,integer)
n      : priciple quantum number of each state (in,integer(nst))
l      : quantum number l of each state (in,integer(nst))
k      : quantum number k (l or l+1) of each state (in,integer(nst))
occ    : occupancy of each state (inout,real(nst))
xctype : exchange-correlation type (in,integer)
xcgrad : 1 for GGA functional, 0 otherwise (in,integer)
nr     : number of radial mesh points (in,integer)
r      : radial mesh (in,real(nr))
eval   : eigenvalue without rest-mass energy for each state (out,real(nst))
rho    : charge density (out,real(nr))
vr     : self-constistent potential (out,real(nr))
rwf    : major and minor components of radial wavefunctions for each state
         (out,real(nr,2,nst))
```

DESCRIPTION:

Solves the Dirac-Kohn-Sham equations for an atom using the exchange-correlation functional `xctype` and returns the self-consistent radial wavefunctions, eigenvalues, charge densities and potentials. The variable `np` defines the order of polynomial used for performing numerical integration. Requires the exchange-correlation interface routine `xcifc`.

REVISION HISTORY:

```
Created September 2002 (JKD)
Fixed s.c. convergence problem, October 2003 (JKD)
Added support for GGA functionals, June 2006 (JKD)
Almost entirely rewritten 2013 (Andris)
```

1.0.6 writestate (Source File: writestate.f90)

INTERFACE:

```
Subroutine writestate
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Writes the charge density, potentials and other relevant variables to the file `STATE.OUT`. Note to developers: changes to the way the variables are written should be mirrored in `readstate`.

REVISION HISTORY:

```
Created May 2003 (JKD)
```

1.0.7 getevecsv (Source File: getevecsv.f90)

INTERFACE:

```
Subroutine getevecsv (vpl, evecsv)
```

USES:

```
Use modinput
Use modmpi
Use mod_eigenvalue_occupancy, only: nstfv, nstsv
Use mod_names, only: filetag_evecsv
Use mod_kpoint, only: vkl_ptr
Use mod_symmetry, only: lspnsymc, symlatc
```

DESCRIPTION:

The file where the (second-variational) eigenvectors are stored is `EVECSV.OUT`. It is a direct-access binary file, the record length of which can be determined with the help of the array sizes and data type information. One record of this file has the following structure

k_{lat}	N_{stsv}	Φ

The following table explains the parts of the record in more detail

name	type	shape	description
k_{lat}	real(8)	3	k-point in lattice coordinates
N_{stsv}	integer	1	number of (second-variational) states (without core states)
Φ	complex(8)	$N_{\mathrm{stsv}} \times N_{\mathrm{stsv}}$	(second-variational) eigenvector array

REVISION HISTORY:

```
Documentation added, Dec 2009 (S. Sagmeister)
```

1.0.8 genylmg (Source File: genylmg.f90)

INTERFACE:

```
Subroutine genylmg
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Generates a set of spherical harmonics, $Y_{lm}(\hat{\mathbf{G}})$, with angular momenta up to `lmaxvr` for the set of $\mathbf{G}$-vectors.

REVISION HISTORY:

```
Created June 2003 (JKD)
```

1.0.9 gencfun (Source File: gencfun.f90)

INTERFACE:

```
Subroutine gencfun
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Generates the smooth characteristic function. This is the function which is 0 within the muffin-tins and 1 in the intersitial region and is constructed from radial step function form-factors with $G < G_{\max}$. The form factors are given by

$$\tilde{\Theta}_i(G) = \begin{cases} \frac{4\pi R_i^3}{3\Omega} & G = 0 \\ \frac{4\pi R_i^3}{\Omega}\frac{j_1(GR_i)}{GR_i} & 0 < G \le G_{\max} \\ 0 & G > G_{\max} \end{cases},$$

where R_i is the muffin-tin radius of the ith species and Ω is the unit cell volume. Therefore the characteristic function in G-space is

$$\tilde{\Theta}(\mathbf{G}) = \delta_{G,0} - \sum_{ij} \exp(-i\mathbf{G}\cdot\mathbf{r}_{ij})\tilde{\Theta}_i(G),$$

where $\mathbf{r}_{ij}$ is the position of the jth atom of the ith species.

REVISION HISTORY:

```
Created January 2003 (JKD)
```

1.0.10 kinetic (Source File: kinetic.f90)

INTERFACE:

```
Subroutine KineticEnergy(ik,evecfv,apwalm,ngp,vgpc,igpig)
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Calculates the kinetic energy directly.

REVISION HISTORY:

```
Created December 2014 (A. Gulans)
```

1.0.11 genwfsv (Source File: genwfsv.f90)

INTERFACE:

```
Subroutine genwfsv (tocc, ngp, igpig, evalsvp, apwalm, evecfv, evecsv, &
& wfmt, wfir)
```

USES:

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
tocc    : .true. if only occupied wavefunctions are required (in,logical)
ngp     : number of G+p-vectors (in,integer)
igpig   : index from G+p-vectors to G-vectors (in,integer(ngkmax))
evalsvp : second-variational eigenvalue for every state (in,real(nstsv))
apwalm  : APW matching coefficients
          (in,complex(ngkmax,apwordmax,lmmaxapw,natmtot))
evecfv  : first-variational eigenvectors (in,complex(nmatmax,nstfv))
evecsv  : second-variational eigenvectors (in,complex(nstsv,nstsv))
wfmt    : muffin-tin part of the wavefunctions for every state in spherical
          coordinates (out,complex(lmmaxvr,nrcmtmax,natmtot,nspinor,nstsv))
wfir    : interstitial part of the wavefunctions for every state
          (out,complex(ngrtot,nspinor,nstsv))
```

DESCRIPTION:

Calculates the second-variational spinor wavefunctions in both the muffin-tin and interstitial regions for every state of a particular k-point. The wavefunctions in both regions are stored

on a real-space grid. A coarse radial mesh is assumed in the muffin-tins with with angular momentum cut-off of `lmaxvr`. If `tocc` is `.true.`, then only the occupied states (those below the Fermi energy) are calculated.

REVISION HISTORY:

```
Created November 2004 (Sharma)
Modified April 2014 (UW)
```

1.0.12 mossbauer (Source File: mossbauer.f90)

INTERFACE:

```
Subroutine mossbauer
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Computes the contact charge density and contact magnetic hyperfine field for each atom and outputs the data to the file `MOSSBAUER.OUT`. The nuclear radius used for the contact quantities is approximated by the empirical formula $R_{\rm N} = 1.25Z^{1/3}$ fm, where Z is the atomic number.

REVISION HISTORY:

```
Created May 2004 (JKD)
```

1.0.13 genfftmap (Source File: genfftmap.f90)

INTERFACE:

```
Subroutine genfftmap(fftmap,gmaxcustom)
```

USES:

```
Use modinput
Use modmain
use modxs, only : fftmap_type
```

DESCRIPTION:

Prepares a new FFT grid with given $G_{\rm max}$.

REVISION HISTORY:

```
Created October 2014 (Andris)
```

1.0.14 zfftifc (Source File: zfftifc.f90)

INTERFACE:

```
Subroutine zfftifc (nd, n, sgn, z)
```

INPUT/OUTPUT PARAMETERS:

```
nd   : number of dimensions (in,integer)
n    : grid sizes (in,integer(nd))
sgn  : FFT direction, -1: forward; 1: backward (in,integer)
z    : array to transform (inout,complex(n(1)*n(2)*...*n(nd)))
```

DESCRIPTION:

Interface to the double-precision complex fast Fourier transform routine. This is to allow machine-optimised routines to be used without affecting the rest of the code. See routine `nfftifc`.

REVISION HISTORY:

```
Created October 2002 (JKD)
```

1.0.15 init_relax (Source File: init_relax.f90)

INTERFACE:

```
Subroutine init_relax
```

USES:

```
Use modinput
Use modmain
```

1.0.16 writehistory (Source File: writehistory.f90)

INTERFACE:

```
Subroutine writehistory
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Outputs the atomic positions and forces to file, in the XYZ format, which may be then used to visualize the optimization trajectory

REVISION HISTORY:

1.0.17 forcek (Source File: forcek.f90)

INTERFACE:

```
Subroutine forcek (ik, ffacg)
```

USES:

```
Use modinput
Use modmain
Use mod_eigensystem
Use modfvsystem
 use mod_constants
```

DESCRIPTION:

Computes the **k**-dependent contribution to the incomplete basis set (IBS) force. See the calling routine `force` for a full description.

REVISION HISTORY:

```
Created June 2006 (JKD)
Updated for spin-spiral case, May 2007 (Francesco Cricchio and JKD)
```

1.0.18 symrfir (Source File: symrfir.f90)

Subroutine symrfir (ngv, rfir) **USES:**

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
ngv  : number of G-vectors to be used for the Fourier space rotation
       (in,integer)
rfir : real intersitial function (inout,real(ngrtot))
```

DESCRIPTION:

Symmetrises a real scalar interstitial function. The function is first Fourier transformed to G-space, and then averaged over each symmetry by rotating the Fourier coefficients and multiplying them by a phase factor corresponding to the symmetry translation.

REVISION HISTORY:

```
Created July 2007 (JKD)
```

1.0.19 ws_weight (Source File: m_wsweight.f90)

INTERFACE:

```
Subroutine ws_weight (vrl, vrsl, vpl, zwght, kgrid)
```

INPUT/OUTPUT PARAMETERS:

```
vrl   : R vector in lattice coordinates (in,real(3))
vrsl  : R+s vector (vector from atom 1 in cell at origin to atom 2 in
        unit cell at R) in lattice coordinates (in,real(3))
vpl   : q vector in reciprocal lattice coordinates (in,real(3))
zwght : combined complex weight to be used in the Fourier
        transformation (out,complex)
```

DESCRIPTION:

Folds the $\mathbf{R}+\mathbf{s}$ vector back into the Wigner-Seitz cell of the supercell, $\mathbf{s}$ being the atom basis.
Then equivalent vectors (total number n) are searched for and the combined weights and phase factors for the Fourier transformation computed and returned:

$$\frac{1}{n}\sum_{j=1}^{n} e^{-i\mathbf{q}\cdot\mathbf{R}_j}$$

REVISION HISTORY:

```
Created September 2012 (STK)
```

1.0.20 dos (Source File: dos.f90)

INTERFACE:

```
Subroutine dos
```

USES:

```
Use modinput
Use modmain
Use FoX_wxml
Use modmpi, Only: rank, splittfile
```

DESCRIPTION:

Produces a total and partial density of states (DOS) for plotting. The total DOS is written to the file `TDOS.OUT` while the partial DOS is written to the file `PDOS_Sss_Aaaaa.OUT` for atom `aaaa` of species `ss`. In the case of the partial DOS, each symmetrised (l,m)-projection is written consecutively and separated by blank lines. If the global variable `lmirep` is `.true.`, then the density matrix from which the (l,m)-projections are obtained is first rotated into a irreducible representation basis, i.e. one that block diagonalises all the site

symmetry matrices in the Y_{lm} basis. Eigenvalues of a quasi-random matrix in the Y_{lm} basis which has been symmetrised with the site symmetries are written to `ELMIREP.OUT`. This allows for identification of the irreducible representations of the site symmetries, for example e_g or t_{2g}, by the degeneracies of the eigenvalues. In the plot, spin-up is made positive and spin-down negative. See the routines `gendmat` and `brzint`.

REVISION HISTORY:

```
Created January 2004 (JKD)
```

1.0.21 rhonorm (Source File: rhonorm.f90)

INTERFACE:

```
Subroutine rhonorm
```

USES:

```
Use modmain
```

DESCRIPTION:

Loss of precision of the calculated total charge can result because the muffin-tin density is computed on a set of (θ, ϕ) points and then transformed to a spherical harmonic representation. This routine adds a constant to the density so that the total charge is correct. If the error in total charge exceeds a certain tolerance then a warning is issued.

REVISION HISTORY:

```
Created April 2003 (JKD)
Changed from rescaling to adding, September 2006 (JKD)
```

1.0.22 gengpvec (Source File: gengpvec.f90)

INTERFACE:

```
Subroutine gengpvec (vpl, vpc, ngp, igpig, vgpl, vgpc, gpc, tpgpc)
 Subroutine gengpvec (vpl, vpc, ngp, igpig, vgpl, vgpc, gpc, tpgpc,fftmapping)
  The commented version with fftmapping is useful when FFTs of wavefunctions need to be calcul
```

USES:

```
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
vpl   : p-point vector in lattice coordinates (in,real(3))
vpc   : p-point vector in Cartesian coordinates (in,real(3))
ngp   : number of G+p-vectors returned (out,integer)
```

```
igpig : index from G+p-vectors to G-vectors (out,integer(ngkmax))
vgpl  : G+p-vectors in lattice coordinates (out,real(3,ngkmax))
vgpc  : G+p-vectors in Cartesian coordinates (out,real(3,ngkmax))
gpc   : length of G+p-vectors (out,real(ngkmax))
tpgpc : (theta, phi) coordinates of G+p-vectors (out,real(2,ngkmax))
```

DESCRIPTION:

Generates a set of $\mathbf{G}+\mathbf{p}$-vectors for the input p-point with length less than `gkmax`. These are used as the plane waves in the APW functions. Also computes the spherical coordinates of each vector.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

1.0.23 seceqn (Source File: seceqn.f90)

Subroutine seceqn (ik, evalfv, evecfv, evecsv) **USES:**

```
Use modinput
Use modmain
Use modmpi
Use sclcontroll
```

INPUT/OUTPUT PARAMETERS:

```
ik     : k-point number (in,integer)
evalfv : first-variational eigenvalues (out,real(nstfv))
evecfv : first-variational eigenvectors (out,complex(nmatmax,nstfv))
evecsv : second-variational eigenvectors (out,complex(nstsv,nstsv))
```

DESCRIPTION:

Solves the first- and second-variational secular equations. See routines `match`, `seceqnfv`, `seceqnss` and `seceqnsv`.

REVISION HISTORY:

```
Created March 2004 (JKD)
```

1.0.24 genveffig (Source File: genveffig.f90)

INTERFACE:

```
Subroutine genveffig
```

USES:

```
Use modmain
```

DESCRIPTION:

Generates the Fourier transform of the effective potential in the intersitial region. The potential is first multiplied by the characteristic function which zeros it in the muffin-tins. See routine `gencfun`.

REVISION HISTORY:

```
Created January 2004 (JKD)
```

1.0.25 zfmtinp (Source File: zfmtinp.f90)

INTERFACE:

```
Complex (8) Function zfmtinp (tsh, lmax, nr, r, ld, zfmt1, zfmt2)
```

INPUT/OUTPUT PARAMETERS:

```
tsh   : .true. if the functions are in spherical harmonics (in,logical)
lmax  : maximum angular momentum
nr    : number of radial mesh points (in,integer)
r     : radial mesh (in,real(nr))
ld    : leading dimension (in,integer)
zfmt1 : first complex muffin-tin function in spherical harmonics/
        coordinates (in,complex(ld,nr))
zfmt2 : second complex muffin-tin function in spherical harmonics/
        coordinates (in,complex(ld,nr))
```

DESCRIPTION:

Calculates the inner product of two complex fuctions in the muffin-tin. In other words, given two complex functions of the form

$$f(\mathbf{r}) = \sum_{l=0}^{l_{\max}} \sum_{m=-l}^{l} f_{lm}(r) Y_{lm}(\hat{\mathbf{r}}),$$

the function returns

$$I = \sum_{l=0}^{l_{\max}} \sum_{m=-l}^{l} \int f_{lm}^{1*}(r) f_{lm}^{2}(r) r^2 \, dr \; .$$

Note that if `tsh` is `.false.` the functions are in spherical coordinates rather than spherical harmonics. In this case I is multiplied by $4\pi/(l_{\max}+1)^2$.

REVISION HISTORY:

```
Created November 2003 (Sharma)
Modified April 2014 (UW)
```

1.0.26 rtorat (Source File: mod_kpoint.F90)

INTERFACE:

```
Subroutine rtorat (eps, n, x, k, div)
```

DESCRIPTION:

This subroutine factorizes the real coordinates of a vector $\mathbf{x}$. The output is an integer vector $\mathbf{k}$, such that

$$|x(i) - k(i)/\mathrm{div}| < \mathrm{eps}$$

for all $i = 1, \ldots, n$.

REVISION HISTORY:

```
Created July 2008 by Sagmeister
```

1.0.27 potplot (Source File: potplot.f90)

INTERFACE:

```
Subroutine potplot
```

USES:

```
Use modinput
Use modmain
use modplotlabels
use modmpi, only : rank
```

DESCRIPTION:

Outputs the exchange, correlation and Coulomb potentials, read in from `STATE.OUT`, for 1D, 2D or 3D plotting.

REVISION HISTORY:

```
Created June 2003 (JKD)
```

1.0.28 symrfmt (Source File: symrfmt.f90)

INTERFACE:

```
Subroutine symrfmt (lrstp, is, rot, rfmt, srfmt)
```

USES:

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
lrstp : radial step length (in,integer)
is    : species number (in,integer)
rot   : rotation matrix (in,real(3,3))
rfmt  : input muffin-tin function (in,real(lmmaxvr,nrmtmax))
srfmt : output muffin-tin function (out,real(lmmaxvr,nrmtmax))
```

DESCRIPTION:

Applies a symmetry operation (in the form of a rotation matrix) to a real muffin-tin function. The input function can also be the output function. See the routines `rtozflm` and `rotzflm`.

REVISION HISTORY:

```
Created May 2003 (JKD)
```

1.0.29 poteff (Source File: poteff.f90)

INTERFACE:

```
Subroutine poteff
```

USES:

```
Use modmain
```

DESCRIPTION:

Computes the effective potential by adding together the Coulomb and exchange-correlation potentials. See routines `potcoul` and `potxc`.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

1.0.30 moment (Source File: moment.f90)

INTERFACE:

```
Subroutine moment
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Computes the muffin-tin, interstitial and total moments by integrating the magnetisation.

REVISION HISTORY:

```
Created January 2005 (JKD)
```

1.0.31 allatoms (Source File: allatoms.f90)

INTERFACE:

```
Subroutine allatoms(verbosity)
```

USES:

```
Use modinput
Use modmain
Use FoX_wxml
Use modmpi, only : rank, finitMPI
```

DESCRIPTION:

Solves the Kohn-Sham-Dirac equations for each atom type in the solid and finds the self-consistent radial wavefunctions, eigenvalues, charge densities and potentials. The atomic densities can then be used to initialise the crystal densities, and the atomic self-consistent potentials can be appended to the muffin-tin potentials to solve for the core states. Note that, irrespective of the value of `xctype`, exchange-correlation functional type 3 is used. See also `atoms`, `rhoinit`, `gencore` and `modxcifc`.

REVISION HISTORY:

```
Created September 2002 (JKD)
Modified for GGA, June 2007 (JKD)
Modified for DFT-1/2, 2015 (Ronaldo)
```

1.0.32 zpotclmt (Source File: zpotclmt.f90)

INTERFACE:

```
Subroutine zpotclmt (ptnucl, lmax, nr, r, zn, ld, zrhomt, zvclmt)
```

INPUT/OUTPUT PARAMETERS:

```
ptnucl : .true. if the nucleus is a point particle (in,logical)
lmax   : maximum angular momentum (in,integer)
nr     : number of radial mesh points (in,integer)
r      : radial mesh (in,real(nr))
zn     : nuclear charge at the atomic center (in,real)
ld     : leading dimension (in,integer)
zrhomt : muffin-tin charge density (in,complex(ld,nr))
zvclmt : muffin-tin Coulomb potential (out,complex(ld,nr))
```

DESCRIPTION:

Solves the Poisson equation for the charge density contained in an isolated muffin-tin using the Green's function approach. In other words, the spherical harmonic expansion of the

Coulomb potential, V_{lm}, is obtained from the density expansion, ρ_{lm}, by

$$V_{lm}(r) = \frac{4\pi}{2l+1}\left(\frac{1}{r^{l+1}}\int_0^r \rho_{lm}(r')r'^{l+2}dr' + r^l\int_r^R \frac{\rho_{lm}(r')}{r'^{l-1}}dr'\right) + \frac{1}{Y_{00}}\frac{z}{r}\delta_{l,0}$$

where the last term is the monopole arising from the point charge z, and R is the muffin-tin radius.

REVISION HISTORY:

Created April 2003 (JKD)

1.0.33 writesym (Source File: writesym.f90)

INTERFACE:

```
Subroutine writesym
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Outputs the Bravais, crystal and site symmetry matrices to files `SYMLAT.OUT`, `SYMCRYS.OUT` and `SYMSITE.OUT`, respectively. Also writes out equivalent atoms and related crystal symmetries to `EQATOMS.OUT`.

REVISION HISTORY:

Created October 2002 (JKD)

1.0.34 rfmtinp (Source File: rfmtinp.f90)

INTERFACE:

```
Real (8) Function rfmtinp (lrstp, lmax, nr, r, ld, rfmt1, rfmt2)
```

INPUT/OUTPUT PARAMETERS:

```
lrstp : radial step length (in,integer)
lmax  : maximum angular momentum (in,integer)
nr    : number of radial mesh points (in,integer)
r     : radial mesh (in,real(nr))
ld    : the leading dimension (in,integer)
rfmt1 : first real function inside muffin-tin (in,real(ld,nr))
rfmt2 : second real function inside muffin-tin (in,real(ld,nr))
```

DESCRIPTION:

Calculates the inner product of two real fuctions in the muffin-tin. So given two real functions of the form

$$f(\mathbf{r}) = \sum_{l=0}^{l_{\max}} \sum_{m=-l}^{l} f_{lm}(r) R_{lm}(\hat{\mathbf{r}})$$

where R_{lm} are the real spherical harmonics, the function returns

$$I = \int \sum_{l=0}^{l_{\max}} \sum_{m=-l}^{l} f^1_{lm}(r) f^2_{lm}(r) r^2 dr \ .$$

The radial integral is performed using a high accuracy cubic spline method. See routines `genrlm` and `fderiv`.

REVISION HISTORY:

```
Created November 2003 (Sharma)
```

1.0.35 occupy (Source File: occupy.f90)

INTERFACE:

```
Subroutine occupy
```

USES:

```
Use modinput
Use modmain
use mod_opt_tetra
```

DESCRIPTION:

Finds the Fermi energy and sets the occupation numbers for the second-variational states using the routine `fermi`.

REVISION HISTORY:

```
Created February 2004 (JKD)
Modifiactions for tetrahedron method, November 2007 (RGA alias
  Ricardo Gomez-Abal)
Modifications for tetrahedron method, 2007-2010 (Sagmeister)
Modifications for tetrahedron method, 2011 (DIN)
Simplicistic method for systems with gap added, 2013 (STK)
```

1.0.36 gencore (Source File: gencore.f90)

INTERFACE:

```
Subroutine gencore
```

USES:

```
      use modinput, only: input
      use mod_atoms, only: natoms, idxas, spvr, spnr, nspecies, &
        & spnst, spcore, spn, spr, spk, spl, spocc, natmmax, &
        & spnrmax
      use mod_symmetry, only: eqatoms
      use mod_muffin_tin, only: nrmt
      use mod_potential_and_density, only: veffmt
      use mod_constants, only: y00, fourpi
      use mod_corestate, only: rhocr, rwfcr, evalcr
  Use modmain
```

DESCRIPTION:

Computes the core radial wavefunctions, eigenvalues and densities. The radial Dirac equation is solved in the spherical part of the effective potential to which the atomic potential has been appended for $r > R^{\mathrm{MT}}$. In the case of spin-polarised calculations, the effective magnetic field is ignored.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

1.0.37 symvect (Source File: symvect.f90)

INTERFACE:

```
Subroutine symvect (tspin, vc)
```

USES:

```
  Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
tspin : .true. if the global spin rotations should be used instead of the
        spatial rotations (in,logical)
vc    : vectors in Cartesian coordinates for all atoms (in,real(3,natmtot))
```

DESCRIPTION:

Symmetrises a 3-vector at each atomic site by rotating and averaging over equivalent atoms. Depending on `tspin`, either the spatial or spin part of the crystal symmetries are used.

REVISION HISTORY:

```
Created June 2004 (JKD)
Modified for spin rotations, May 2008 (JKD)
```

1.0.38 genrmesh (Source File: genrmesh.f90)

INTERFACE:

```
Subroutine genrmesh
```

USES:

```
Use modinput
Use mod_atoms
Use mod_muffin_tin
```

DESCRIPTION:

Generates the coarse and fine radial meshes for each atomic species in the crystal. Also determines which points are in the inner part of the muffin-tin using the value of `fracinr`. For species i the radial mesh starts from the value $R_0 = r(1)$, hits the muffin-tin surface at $R_{\rm MT} = r(N)$, and ends at the effective infinity value $R_\infty = r(N_\infty)$. The number of points up to the effective infinity are determined by the number of points N up to the muffin-tin radius as well as by the smallest and largest mesh point and the muffin-tin radius, and is given by

$$N_\infty = \text{round}\left[\frac{(N-1)\ln(R_\infty/R_0)}{\ln(R_{\rm MT}/R_0)}\right] + 1.$$

The radial mesh points are finally defined by

$$r(j) = R_0\left(\frac{R_{\rm MT}}{R_0}\right)^{\frac{j-1}{N-1}},$$

for $j = 1, \ldots, N_\infty$.
Note: The number of mesh points initially defined in species file is adapted to be commensurate with the coarse mesh of step size `lradstep`

$$N = N^* - \mod(N^*, \texttt{lradstep}),$$

if N^* was the number of points defined in the species file. The number of mesh points $\tilde{N}$ of the coarse mesh $\tilde{r}(j)$ reads

$$\tilde{N} = \left\lfloor \frac{N-1}{\texttt{lradstep}} \right\rfloor + 1.$$

It is given by

$$\tilde{r}(j) = r([j-1] * \texttt{lradstep} + 1),$$

for $j = 1, \ldots, \tilde{N}$ and has the properties $\tilde{r}(1) = r(1) = R_0$ and $\tilde{r}(\tilde{N}) = r(N) = R_{\rm MT}$.
REVISION HISTORY:

```
Created September 2002 (JKD)
Revised and updated documentation, April 2011 (S. Sagmeister)
```

1.0.39 findsymcrys (Source File: findsymcrys.f90)

INTERFACE:

```
Subroutine findsymcrys
```

USES:

```
      Use modinput
      Use modmain
#ifdef XS
      Use modxs
#endif
```

DESCRIPTION:

Finds the complete set of symmetries which leave the crystal structure (including the magnetic fields) invariant. A crystal symmetry is of the form $\{\alpha_S|\alpha_R|\mathbf{t}\}$, where $\mathbf{t}$ is a translation vector, α_R is a spatial rotation operation and α_S is a global spin rotation. Note that the order of operations is important and defined to be from right to left, i.e. translation followed by spatial rotation followed by spin rotation. In the case of spin-orbit coupling $\alpha_S = \alpha_R$. In order to determine the translation vectors, the entire atomic basis is shifted so that the first atom in the smallest set of atoms of the same species is at the origin. Then all displacement vectors between atoms in this set are checked as possible symmetry translations. If the global variable `tshift` is set to `.false.` then the shift is not performed. See L. M. Sandratskii and P. G. Guletskii, *J. Phys. F: Met. Phys.* **16**, L43 (1986) and the routine `findsym`.

REVISION HISTORY:

```
Created April 2007 (JKD)
```

1.0.40 match (Source File: match.f90)

INTERFACE:

```
Subroutine match (ngp, gpc, tpgpc, sfacgp, apwalm)
```

USES:

```
      use modinput
      use mod_constants, only: fourpi, zil
      use mod_lattice, only: omega
      use mod_APW_LO, only: apwordmax, apword, apwfr
      use mod_atoms, only: nspecies, natoms, idxas, spr, natmtot
      use mod_muffin_tin, only: rmt, nrmt, idxlm, lmmaxapw
      use mod_Gkvector, only: ngkmax_ptr
 Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
ngp    : number of G+p-vectors (in,integer)
gpc    : length of G+p-vectors (in,real(ngkmax_ptr))
tpgpc  : (theta, phi) coordinates of G+p-vectors (in,real(2,ngkmax_ptr))
sfacgp : structure factors of G+p-vectors (in,complex(ngkmax_ptr,natmtot))
apwalm : APW matching coefficients
         (out,complex(ngkmax_ptr,apwordmax,lmmaxapw,natmtot))
```

DESCRIPTION:

Computes the $(\mathbf{G}+\mathbf{p})$-dependent matching coefficients for the APW basis functions. Inside muffin-tin α, the APW functions are given by

$$\phi^{\alpha}_{\mathbf{G}+\mathbf{p}}(\mathbf{r}) = \sum_{l=0}^{l_{\max}} \sum_{m=-l}^{l} \sum_{j=1}^{M^{\alpha}_{l}} A^{\alpha}_{jlm}(\mathbf{G}+\mathbf{p}) u^{\alpha}_{jl}(r) Y_{lm}(\hat{\mathbf{r}}),$$

where $A^{\alpha}_{jlm}(\mathbf{G}+\mathbf{p})$ is the matching coefficient, M^{α}_{l} is the order of the APW and u^{α}_{jl} is the radial function. In the interstitial region, an APW function is a plane wave, $\exp(i(\mathbf{G}+\mathbf{p})\cdot\mathbf{r})/\sqrt{\Omega}$, where Ω is the unit cell volume. Ensuring continuity up to the $(M^{\alpha}_{l}-1)$th derivative across the muffin-tin boundary therefore requires that the matching coefficients satisfy

$$\sum_{j=1}^{M^{\alpha}_{l}} D_{ij} A^{\alpha}_{jlm}(\mathbf{G}+\mathbf{p}) = b_i \ ,$$

where

$$D_{ij} = \left. \frac{d^{i-1} u^{\alpha}_{jl}(r)}{dr^{i-1}} \right|_{r=R_{\alpha}}$$

and

$$b_i = \frac{4\pi i^l}{\sqrt{\Omega}} |\mathbf{G}+\mathbf{p}|^{i-1} j_l^{(i-1)}(|\mathbf{G}+\mathbf{p}|R_{\alpha}) \exp(i(\mathbf{G}+\mathbf{p})\cdot\mathbf{r}_{\alpha}) Y^{*}_{lm}(\widehat{\mathbf{G}+\mathbf{p}}),$$

with $\mathbf{r}_{\alpha}$ the atomic position and R_{α} the muffin-tin radius. See routine `wavefmt`.

REVISION HISTORY:

```
Created April 2003 (JKD)
Fixed documentation, June 2006 (JKD)
```

1.0.41 checkmt (Source File: checkmt.f90)

INTERFACE:

```
Subroutine checkmt
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Checks for overlapping muffin-tins. If any muffin-tins are found to intersect the program is terminated with error.

REVISION HISTORY:

```
Created May 2003 (JKD)
Revised October 2014 (PAS)
```

1.0.42 zfinp (Source File: zfinp.f90)

INTERFACE:

```
Complex (8) Function zfinp (tsh, zfmt1, zfmt2, zfir1, zfir2)
```

USES:

```
Use modmain
Use modinput
```

INPUT/OUTPUT PARAMETERS:

```
tsh   : .true. if the muffin-tin functions are in spherical harmonics
        (in,logical)
zfmt1 : first complex function in spherical harmonics/coordinates for all
        muffin-tins (in,complex(lmmaxvr,nrcmtmax,natmtot))
zfmt2 : second complex function in spherical harmonics/coordinates for all
        muffin-tins (in,complex(lmmaxvr,nrcmtmax,natmtot))
zfir1 : first complex interstitial function in real-space
        (in,complex(ngrtot))
zfir2 : second complex interstitial function in real-space
        (in,complex(ngrtot))
```

DESCRIPTION:

Calculates the inner product of two complex fuctions over the entire unit cell. The muffin-tin functions should be stored on the coarse radial grid and have angular momentum cut-off `lmaxvr`. In the intersitial region, the integrand is multiplied with the characteristic function, to remove the contribution from the muffin-tin. See routines `zfmtinp` and `gencfun`.

REVISION HISTORY:

```
Created July 2004 (Sharma)
Modified March 2014 (UW)
```

1.0.43 rvfcross (Source File: rvfcross.f90)

INTERFACE:

```
Subroutine rvfcross (rvfmt1, rvfmt2, rvfir1, rvfir2, rvfmt3, rvfir3)
```

USES:

```
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
rvfmt1 : first input muffin-tin field (in,real(lmmaxvr,nrmtmax,natmtot,3))
rvfmt2 : second input muffin-tin field (in,real(lmmaxvr,nrmtmax,natmtot,3))
rvfir1 : first input interstitial field (in,real(ngrtot,3))
rvfir2 : second input interstitial field (in,real(ngrtot,3))
rvfmt3 : output muffin-tin field (out,real(lmmaxvr,nrmtmax,natmtot,3))
rvfir3 : output interstitial field (out,real(ngrtot,3))
```

DESCRIPTION:

Given two real vector fields, $\mathbf{f}_1$ and $\mathbf{f}_2$, defined over the entire unit cell, this routine computes the local cross product

$$\mathbf{f}_3(\mathbf{r}) \equiv \mathbf{f}_1(\mathbf{r}) \times \mathbf{f}_2(\mathbf{r}).$$

REVISION HISTORY:

```
Created February 2007 (JKD)
```

1.0.44 charge (Source File: charge.f90)

INTERFACE:

```
Subroutine charge
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Computes the muffin-tin, interstitial and total charges by integrating the density.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

1.0.45 vecplot (Source File: vecplot.f90)

INTERFACE:

```
Subroutine vecplot
```

DESCRIPTION:

Use modinput use modplotlabels use modmpi, only : rank Outputs a 2D or 3D vector field for plotting. The vector field can be the magnetisation vector field, $\mathbf{m}$; the exchange-correlation magnetic field, $\mathbf{B}_{\rm xc}$; or the electric field $\mathbf{E} \equiv -\nabla V_{\rm C}$. The magnetisation is obtained from the spin density matrix, $\rho_{\alpha\beta}$, by solving

$$\rho_{\alpha\beta}(\mathbf{r}) = \frac{1}{2}\left(n(\mathbf{r})\delta_{\alpha\beta} + \sigma \cdot \mathbf{m}(\mathbf{r})\right),$$

where $n \equiv {\rm tr}\,\rho_{\alpha\beta}$ is the total density. In the case of 2D plots, the magnetisation vectors are still 3D, but are in the coordinate system of the plane.

REVISION HISTORY:

```
Created August 2004 (JKD)
Included electric field plots, August 2006 (JKD)
```

1.0.46 nfftifc (Source File: nfftifc.f90)

INTERFACE:

```
Subroutine nfftifc (n)
```

INPUT/OUTPUT PARAMETERS:

```
n : required/avalable grid size (in,integer)
```

DESCRIPTION:

Interface to the grid requirements of the fast Fourier transform routine. Most routines restrict n to specific prime factorisations. This routine returns the next largest grid size allowed by the FFT routine.

REVISION HISTORY:

```
Created October 2002 (JKD)
```

1.0.47 init0 (Source File: init0.f90)

INTERFACE:

```
Subroutine init0
```

USES:

```
      Use modinput
      Use modmain
      Use modxcifc
#ifdef XS
      Use modxs
#endif
```

DESCRIPTION:

Performs basic consistency checks as well as allocating and initialising global variables not dependent on the k-point set.

REVISION HISTORY:

```
Created January 2004 (JKD)
```

1.0.48 init1 (Source File: init1.f90)

INTERFACE:

```
Subroutine init1
```

USES:

```
      Use modinput
      Use modmain
#ifdef TETRA
      Use modtetra
#endif
#ifdef XS
      Use modxs
#endif
      use modfvsystem
```

DESCRIPTION:

Generates the k-point set and then allocates and initialises global variables which depend on the k-point set.

REVISION HISTORY:

```
Created January 2004 (JKD)
```

1.0.49 writekpts (Source File: writekpts.f90)

INTERFACE:

```
subroutine writekpts
```

USES:

```
  use mod_misc, only: filext
  use mod_kpoint, only: nkpt, vkl, wkpt
  use mod_eigensystem, only: nmat
```

DESCRIPTION:

Writes the k-points in lattice coordinates, weights and number of $\mathbf{G}+\mathbf{k}$-vectors to the file `KPOINTS.OUT`.

REVISION HISTORY:

```
Created June 2003 (JKD)
```

1.0.50 elfplot (Source File: elfplot.f90)

INTERFACE:

```
Subroutine elfplot
```

USES:

```
Use modinput
Use modmain
use modplotlabels
use modmpi, only : rank
```

DESCRIPTION:

Outputs the electron localisation function (ELF) for 1D, 2D or 3D plotting. The spin-averaged ELF is given by

$$f_{\rm ELF}(\mathbf{r}) = \frac{1}{1+[D(\mathbf{r})/D^0(\mathbf{r})]^2},$$

where

$$D(\mathbf{r}) = \frac{1}{2}\left(\tau(\mathbf{r}) - \frac{1}{4}\frac{[\nabla n(\mathbf{r})]^2}{n(\mathbf{r})}\right)$$

and

$$\tau(\mathbf{r}) = \sum_{i=1}^{N} |\nabla \Psi_i(\mathbf{r})|^2$$

is the spin-averaged kinetic energy density from the spinor wavefunctions. The function D^0 is the kinetic energy density for the homogeneous electron gas evaluated for $n(\mathbf{r})$:

$$D^0(\mathbf{r}) = \frac{3}{5}(6\pi^2)^{2/3}\left(\frac{n(\mathbf{r})}{2}\right)^{5/3}.$$

The ELF is useful for the topological classification of bonding. See for example T. Burnus, M. A. L. Marques and E. K. U. Gross [Phys. Rev. A 71, 10501 (2005)].

REVISION HISTORY:

```
Created September 2003 (JKD)
Fixed bug found by F. Wagner (JKD)
```

1.0.51 writeeval (Source File: writeeval.f90)

INTERFACE:

```
Subroutine writeeval
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Outputs the second-variational eigenvalues and occupation numbers to the file `EIGVAL.OUT`.

REVISION HISTORY:

```
Created June 2003 (JKD)
```

1.0.52 genidxlo (Source File: genidxlo.f90)

INTERFACE:

```
Subroutine genidxlo
```

USES:

```
Use modmain
```

DESCRIPTION:

Generates an index array which maps the local-orbitals in each atom to their locations in the overlap or Hamiltonian matrices. Also finds the total number of local-orbitals.

REVISION HISTORY:

```
Created June 2003 (JKD)
```

1.0.53 force (Source File: force.f90)

INTERFACE:

```
Subroutine force
```

USES:

```
Use modinput
Use modmain
Use modmpi
```

DESCRIPTION:

Computes the various contributions to the atomic forces. In principle, the force acting on a nucleus is simply the gradient at that site of the classical electrostatic potential from the other nuclei and the electronic density. This is a result of the Hellmann-Feynman theorem. However because the basis set is dependent on the nuclear coordinates and is not complete,

the Hellman-Feynman force is inacurate and corrections to it are required. The first is the core correction which arises because the core wavefunctions were determined by neglecting the non-spherical parts of the effective potential v_s. Explicitly this is given by

$$\mathbf{F}^{\alpha}_{\rm core} = \int_{{\rm MT}_\alpha} v_{\rm s}(\mathbf{r}) \nabla \rho^{\alpha}_{\rm core}(\mathbf{r})\, d\mathbf{r}$$

for atom α. The second, which is the incomplete basis set (IBS) correction, is due to the position dependence of the APW functions, and is derived by considering the change in total energy if the eigenvector coefficients were fixed and the APW functions themselves were changed. This would result in changes to the first-variational Hamiltonian and overlap matrices given by

$$\delta H^{\alpha}_{\mathbf{G},\mathbf{G}'} = i(\mathbf{G}-\mathbf{G}')\left(H^{\alpha}_{\mathbf{G}+\mathbf{k},\mathbf{G}'+\mathbf{k}} - \frac{1}{2}(\mathbf{G}+\mathbf{k})\cdot(\mathbf{G}'+\mathbf{k})\tilde{\Theta}_{\alpha}(\mathbf{G}-\mathbf{G}')e^{-i(\mathbf{G}-\mathbf{G}')\cdot\mathbf{r}_{\alpha}}\right)$$
$$\delta O^{\alpha}_{\mathbf{G},\mathbf{G}'} = i(\mathbf{G}-\mathbf{G}')\left(O^{\alpha}_{\mathbf{G}+\mathbf{k},\mathbf{G}'+\mathbf{k}} - \tilde{\Theta}_{\alpha}(\mathbf{G}-\mathbf{G}')e^{-i(\mathbf{G}-\mathbf{G}')\cdot\mathbf{r}_{\alpha}}\right)$$

where both $\mathbf{G}$ and $\mathbf{G}'$ run over the APW indices; $\tilde{\Theta}_{\alpha}$ is the form factor of the smooth step function for muffin-tin α; and H^{α} and O^{α} are the muffin-tin Hamiltonian and overlap matrices, respectively. The APW-local-orbital part is given by

$$\delta H^{\alpha}_{\mathbf{G},\mathbf{G}'} = i(\mathbf{G}+\mathbf{k})H^{\alpha}_{\mathbf{G}+\mathbf{k},\mathbf{G}'+\mathbf{k}}$$
$$\delta O^{\alpha}_{\mathbf{G},\mathbf{G}'} = i(\mathbf{G}+\mathbf{k})O^{\alpha}_{\mathbf{G}+\mathbf{k},\mathbf{G}'+\mathbf{k}}$$

where $\mathbf{G}$ runs over the APW indices and $\mathbf{G}'$ runs over the local-orbital indices. There is no contribution from the local-orbital-local-orbital part of the matrices. We can now write the IBS correction in terms of the basis of first-variational states as

$$\mathbf{F}^{\alpha\mathbf{k}}_{ij} = \sum_{\mathbf{G},\mathbf{G}'} b^{i\mathbf{k}*}_{\mathbf{G}} b^{j\mathbf{k}}_{\mathbf{G}'} \left(\delta H^{\alpha}_{\mathbf{G},\mathbf{G}'} - \epsilon_j \delta O^{\alpha}_{\mathbf{G},\mathbf{G}'}\right),$$

where $b^{i\mathbf{k}}$ is the first-variational eigenvector. Finally, the $\mathbf{F}^{\alpha\mathbf{k}}_{ij}$ matrix elements can be multiplied by the second-variational coefficients, and contracted over all indices to obtain the IBS force:

$$\mathbf{F}^{\alpha}_{\rm IBS} = \sum_{\mathbf{k}} w_{\mathbf{k}} \sum_{l\sigma} n_{l\mathbf{k}} \sum_{ij} c^{l\mathbf{k}*}_{\sigma i} c^{l\mathbf{k}}_{\sigma j} \mathbf{F}^{\alpha\mathbf{k}}_{ij} + \int_{{\rm MT}_\alpha} v_{\rm s}(\mathbf{r}) \nabla \left[\rho(\mathbf{r}) - \rho^{\alpha}_{\rm core}(\mathbf{r})\right] d\mathbf{r},$$

where $c^{l\mathbf{k}}$ are the second-variational coefficients, $w_{\mathbf{k}}$ are the k-point weights, $n_{l\mathbf{k}}$ are the occupancies, and $v_{\rm s}$ is the Kohn-Sham effective potential. See routines `hmlaa`, `olpaa`, `hmlalo`, `olpalo`, `energy`, `seceqn` and `gencfun`.

REVISION HISTORY:

```
Created January 2004 (JKD)
Fixed problem with second-variational forces, May 2008 (JKD)
k-point parallelisation of IBS forces, October 2013 (Andris)
```

1.0.54 findsymlat (Source File: findsymlat.f90)

Subroutine findsymlat **USES:**

```
Use modinput
Use modmain
```

DESCRIPTION:

Finds the point group symmetries which leave the Bravais lattice invariant. Let A be the matrix consisting of the lattice vectors in columns, then

$$g = A^{\mathrm{T}} A$$

is the metric tensor. Any 3×3 matrix S with elements -1, 0 or 1 is a point group symmetry of the lattice if $\det(S)$ is -1 or 1, and

$$S^{\mathrm{T}} g S = g.$$

The first matrix in the set returned is the identity.
REVISION HISTORY:

```
Created January 2003 (JKD)
Removed arguments and simplified, April 2007 (JKD)
```

1.0.55 readfermi (Source File: readfermi.f90)

INTERFACE:

```
subroutine readfermi
```

USES:

```
use mod_misc, only: filext
use mod_eigenvalue_occupancy, only: efermi
```

DESCRIPTION:

Reads the Fermi energy from the file `EFERMI.OUT`.
REVISION HISTORY:

```
Created March 2005 (JKD)
```

1.0.56 wavefmt (Source File: wavefmt.f90)

INTERFACE:

```
Subroutine wavefmt (lrstp, lmax, is, ia, ngp, apwalm, evecfv, ld, wfmt)
```

USES:

```
Use modinput
use mod_atoms, only: idxas, natmtot
use mod_muffin_tin, only: idxlm, nrmt, lmmaxapw
use mod_eigensystem, only: idxlo, nmatmax_ptr
use mod_APW_LO, only: apword, apwordmax, apwfr,&
                    & lofr, nlorb, lorbl
use mod_Gkvector, only: ngkmax_ptr
```

INPUT/OUTPUT PARAMETERS:

```
lrstp  : radial step length (in,integer)
lmax   : maximum angular momentum required (in,integer)
is     : species number (in,integer)
ia     : atom number (in,integer)
ngp    : number of G+p-vectors (in,integer)
apwalm : APW matching coefficients
         (in,complex(ngkmax,apwordmax,lmmaxapw,natmtot))
evecfv : first-variational eigenvector (in,complex(nmatmax))
ld     : leading dimension (in,integer)
wfmt   : muffin-tin wavefunction (out,complex(ld,*))
```

DESCRIPTION:

Calculates the first-variational wavefunction in the muffin-tin in terms of a spherical harmonic expansion. For atom α and a particular k-point $\mathbf{p}$, the r-dependent (l,m)-coefficients of the wavefunction for the ith state are given by

$$\Phi^{i\mathbf{p}}_{\alpha lm}(r)=\sum_{\mathbf{G}} b^{i\mathbf{p}}_{\mathbf{G}}\sum_{j=1}^{M^{\alpha}_{l}} A^{\alpha}_{jlm}(\mathbf{G}+\mathbf{p})u^{\alpha}_{jl}(r)+\sum_{j=1}^{N^{\alpha}} b^{i\mathbf{p}}_{(\alpha,j,m)}v^{\alpha}_{j}(r)\delta_{l,l_j},$$

where $b^{i\mathbf{p}}$ is the ith eigenvector returned from routine `seceqn`; $A^{\alpha}_{jlm}(\mathbf{G}+\mathbf{p})$ is the matching coefficient; M^{α}_{l} is the order of the APW; u^{α}_{jl} is the APW radial function; N^{α} is the number of local-orbitals; v^{α}_{j} is the jth local-orbital radial function; and (α,j,m) is a compound index for the location of the local-orbital in the eigenvector. See routines `genapwfr`, `genlofr`, `match` and `seceqn`.

REVISION HISTORY:

```
Created April 2003 (JKD)
Fixed description, October 2004 (C. Brouder)
Removed argument ist, November 2006 (JKD)
```

1.0.57 wavefmt_add (Source File: wavefmt.f90)

INTERFACE:

```
Subroutine wavefmt_add (nr, ld, wfmt, a, b, lrstp, fr)
```

INPUT/OUTPUT PARAMETERS:

```
nr     : number of radial mesh points (in,integer)
ld     : leading dimension (in,integer)
wfmt   : complex muffin-tin wavefunction passed in as a real array
         (inout,real(2*ld,*))
a      : real part of complex constant (in,real)
b      : imaginary part of complex constant (in,real)
lrstp  : radial step length (in,integer)
fr     : real radial function (in,real(lrstp,*))
```

DESCRIPTION:

Adds a real function times a complex constant to a complex muffin-tin wavefunction as efficiently as possible. See routine `wavefmt`.

REVISION HISTORY:

```
Created December 2006 (JKD)
```

1.0.58 getngkmax (Source File: getngkmax.f90)

INTERFACE:

```
Subroutine getngkmax
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Determines the largest number of $\mathbf{G}+\mathbf{k}$-vectors with length less than `gkmax` over all the k-points and stores it in the global variable `ngkmax`. This variable is used for allocating arrays.

REVISION HISTORY:

```
Created October 2004 (JKD)
```

1.0.59 symrf (Source File: symrf.f90)

INTERFACE:

```
Subroutine symrf (lrstp, rfmt, rfir)
```

USES:

```
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
lrstp : radial step length (in,integer)
rfmt  : real muffin-tin function (inout,real(lmmaxvr,nrmtmax,natmtot))
rfir  : real intersitial function (inout,real(ngrtot))
```

DESCRIPTION:

Symmetrises a real scalar function defined over the entire unit cell using the full set of crystal symmetries. In the muffin-tin of a particular atom the spherical harmonic coefficients of every equivlent atom are rotated and averaged. The interstitial part of the function is first Fourier transformed to G-space, and then averaged over each symmetry by rotating the Fourier coefficients and multiplying them by a phase factor corresponding to the symmetry translation. See routines `symrfmt` and `symrfir`.

REVISION HISTORY:

```
Created May 2007 (JKD)
```

1.0.60 gensdmat (Source File: gensdmat.f90)

INTERFACE:

```
Subroutine gensdmat (evecsv, sdmat)
```

USES:

```
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
evecsv : second-variational eigenvectors (in,complex(nstsv,nstsv))
sdmat  : spin density matrices (out,complex(nspinor,nspinor,nstsv))
```

DESCRIPTION:

Computes the spin density matrices for a set of second-variational states.

REVISION HISTORY:

```
Created September 2008 (JKD)
```

1.0.61 writeiad (Source File: writeiad.f90)

INTERFACE:

```
Subroutine writeiad (topt)
```

USES:

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

topt : if .true. then the filename will be {\tt IADIST_OPT.OUT}, otherwise {\tt IADIST.OUT} (in,logical)

DESCRIPTION:

Outputs the interatomic distances to file.

REVISION HISTORY:

Created May 2005 (JKD)

1.0.62 relax (Source File: relax.f90)

INTERFACE:

```
!------------------------------------------------------------------------------80.
! Copyright (C) 2013- exciting team
! This file is distributed under the terms of the GNU General Public License.
! Created       on 15-10-2013 Pasquale Pavone (exciting team)
! Modified      on 15-11-2013 Pasquale Pavone (exciting team)
! Last modified on 14-06-2014 Pasquale Pavone (exciting team)
!------------------------------------------------------------------------------80

subroutine relax
```

USES:

```
Use modinput
Use modmain
Use modmpi
Use scl_xml_out_Module
```

DESCRIPTION:

REVISION HISTORY:

Created February 2013 (DIN)

1.0.63 genpchgs (Source File: genpchgs.F90)

INTERFACE:

```
subroutine genpchgs(ik,evecfv,evecsv)
```

USES:

```
      use modinput
      use modmain
```

DESCRIPTION:

Generate partial charges for each state j, atom α and for each (lm) combination.

REVISION HISTORY:

```
Created 2010 (Sagmeister)
```

1.0.64 writewiq2 (Source File: writewiq2.f90)

INTERFACE:

```
Subroutine writewiq2
```

USES:

```
      Use modmain
```

DESCRIPTION:

Outputs the integrals of $1/q^2$ in the small parallelepiped around each q-point to the file `WIQ2.OUT`. Note that the integrals are calculated after the q-point has been mapped to the first Brillouin zone. See routine genwiq2.

REVISION HISTORY:

```
Created June 2005 (JKD)
```

1.0.65 genshtmat (Source File: genshtmat.f90)

INTERFACE:

```
Subroutine genshtmat
```

USES:

```
      Use modinput
      Use modmain
#ifdef XS
      Use modxs
#endif
```

DESCRIPTION:

Generates the forward and backward spherical harmonic transformation (SHT) matrices using the spherical covering set produced by the routine `sphcover`. These matrices are used to transform a function between its (l,m)-expansion coefficients and its values at the (θ,ϕ) points on the sphere.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

1.0.66 genlofr (Source File: genlofr.f90)

INTERFACE:

```
Subroutine genlofr
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Generates the local-orbital radial functions. This is done by integrating the scalar relativistic Schrödinger equation (or its energy deriatives) at the current linearisation energies using the spherical part of the effective potential. For each local-orbital, a linear combination of `lorbord` radial functions is constructed such that its radial derivatives up to order `lorbord`-1 are zero at the muffin-tin radius. This function is normalised and the radial Hamiltonian applied to it. The results are stored in the global array `lofr`.

REVISION HISTORY:

```
Created March 2003 (JKD)
```

1.0.67 vhalfinit (modified from rhoinit) (Source File: vhalfinit.f90)

INTERFACE:

```
Subroutine vhalfinit
```

USES:

```
Use modinput
Use modmain
#ifdef USEOMP
use omp_lib
#endif
```

DESCRIPTION:

Initialises the crystal charge density. Inside the muffin-tins it is set to the spherical atomic density. In the interstitial region it is taken to be constant such that the total charge is correct. Requires that the atomic densities have already been calculated.

REVISION HISTORY:

```
Created January 2015 (Ronaldo R Pela)
```

1.0.68 gengvec (Source File: gengvec.f90)

INTERFACE:

```
Subroutine gengvec
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Generates a set of **G**-vectors used for the Fourier transform of the charge density and potential and sorts them according to length. Integers corresponding to the vectors in lattice coordinates are stored, as well as the map from these integer coordinates to the **G**-vector index. A map from the **G**-vector set to the standard FFT array structure is also generated. Finally, the number of **G**-vectors with magnitude less than `gmaxvr` is determined.

REVISION HISTORY:

```
Created October 2002 (JKD)
Increased number of G-vectors to ngrtot, July 2007 (JKD)
```

1.0.69 symrvf (Source File: symrvf.f90)

INTERFACE:

```
Subroutine symrvf (lrstp, rvfmt, rvfir)
```

USES:

```
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
lrstp : radial step length (in,integer)
rvfmt : real muffin-tin vector field
        (in,real(lmmaxvr,nrmtmax,natmtot,ndmag))
rvfir : real interstitial vector field
        (in,real(ngrtot,ndmag))
```

DESCRIPTION:

Symmetrises a vector field defined over the entire unit cell using the full set of crystal symmetries. If a particular symmetry involves rotating atom 1 into atom 2, then the spatial and spin rotations of that symmetry are applied to the vector field in atom 2 (expressed in spherical harmonic coefficients), which is then added to the field in atom 1. This is repeated for all symmetry operations. The fully symmetrised field in atom 1 is then rotated and copied to atom 2. Symmetrisation of the interstitial part of the field is performed by `symrvfir`. See also `symrfmt` and `findsym`.

REVISION HISTORY:

```
Created May 2007 (JKD)
Fixed problem with improper rotations, February 2008 (L. Nordstrom,
 F. Bultmark and F. Cricchio)
```

1.0.70 seceqnfv (Source File: seceqnfv.f90)

INTERFACE:

```
Subroutine seceqnfv(ispn, ik, nmatp, ngp, igpig, vgpc, apwalm, evalfv, evecfv)
```

USES:

```
Use modinput
Use mod_Gkvector, only: ngkmax
Use mod_APW_LO, only: apwordmax
Use mod_atoms, only: natmtot
Use mod_muffin_tin, only: lmmaxapw
Use mod_eigensystem, only: nmatmax, h1on, h1aa, h1loa, h1lolo
Use mod_eigenvalue_occupancy, only: nstfv
Use mod_potential_and_density, only: ex_coef
Use modfvsystem
Use mod_hybrids, only: ihyb, vnlmat
use m_plotmat
```

INPUT/OUTPUT PARAMETERS:

```
nmatp  : order of overlap and Hamiltonian matrices (in,integer)
ngp    : number of G+k-vectors for augmented plane waves (in,integer)
igpig  : index from G+k-vectors to G-vectors (in,integer(ngkmax))
vgpc   : G+k-vectors in Cartesian coordinates (in,real(3,ngkmax))
apwalm : APW matching coefficients
         (in,complex(ngkmax,apwordmax,lmmaxapw,natmtot))
evalfv : first-variational eigenvalues (out,real(nstfv))
evecfv : first-variational eigenvectors (out,complex(nmatmax,nstfv))
```

DESCRIPTION:

Solves the secular equation,

$$(H - \epsilon O)b = 0,$$

for the all the first-variational states of the input k-point.

REVISION HISTORY:

```
Created March 2004 (JKD)
```

1.0.71 gensfacgp (Source File: gensfacgp.f90)

INTERFACE:

```
Subroutine gensfacgp(ngp, vgpc, ld, sfacgp)
```

USES:

```
Use mod_atoms, only: nspecies, natoms, natmtot, idxas, atposc
```

INPUT/OUTPUT PARAMETERS:

```
ngp    : number of G+p-vectors (in,integer)
vgpc   : G+p-vectors in Cartesian coordinates (in,real(3,*))
ld     : leading dimension (in,integer)
sfacgp : structure factors of G+p-vectors (out,complex(ld,natmtot))
```

DESCRIPTION:

Generates the atomic structure factors for a set of $\mathbf{G}+\mathbf{p}$-vectors:

$$S_\alpha(\mathbf{G}+\mathbf{p}) = \exp(i(\mathbf{G}+\mathbf{p})\cdot\mathbf{r}_\alpha),$$

where $\mathbf{r}_\alpha$ is the position of atom α.

REVISION HISTORY:

```
Created January 2003 (JKD)
Added OMP, removed modmpi 2017 (BA)
```

1.0.72 readstate (Source File: readstate.f90)

INTERFACE:

```
Subroutine readstate
```

USES:

```
      Use modinput
      Use modmain
#ifdef XS
      Use modxs, Only: isreadstate0
#endif
```

DESCRIPTION:

Reads in the charge density and other relevant variables from the file `STATE.OUT`. Checks for version and parameter compatibility.

REVISION HISTORY:

```
Created May 2003 (JKD)
```

1.0.73 xsgrids_init (Source File: mod_xsgrids.f90)

INTERFACE:

```
subroutine xsgrids_init(vqmtl, gkmax, gqmax_, reducek_,&
  & reduceq_, makegk_, makegq_)
```

USES:

```
use modxs, only: unitout
```

INPUT/OUTPUT PARAMETERS:

```
In:
real(8) :: vqmtl(3) ! Momentum transfer vector in lattice coordinates
real(8) :: gkmax    ! Cutoff for G+k
real(8), optional :: gqmax_    ! Cutoff for G+q
logical, optional :: reducek_  ! Flag for k-point symmetry reduction
logical, optional :: reduceq_  ! Flag for q-point symmetry reduction
logical, optional :: makegk_   ! Construct the G+k vectors or not
logical, optional :: makegq_   ! Construct the G+q vectors or not
```

DESCRIPTION:

Given a momentum transfer vector $\vec{Q}_{\rm mt} = \vec{G}_{\rm mt} + \vec{q}_{\rm mt}$, the routine sets up the associated k-grids $\{\vec{k}\}$, $\{\vec{k}_+ = \vec{k} + \vec{q}_{\rm mt}/2\}$ and $\{\vec{k}_- = \vec{k} - \vec{q}_{\rm mt}/2\}$. Additionally, the q-grids formed from $\{\vec{q} = \vec{k}' - \vec{k}\}$ and $\{\vec{q} = -\vec{k}'_+ - \vec{k}_-\}$ are set up. Various index maps are created relating the points of the different grids. If requested also the corresponding $\vec{G} + \vec{k}$ and $\vec{G} + \vec{q}$ are created.

REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.0.74 xsgrids_finalize (Source File: mod_xsgrids.f90)

INTERFACE:

```
subroutine xsgrids_finalize()
```

DESCRIPTION:

Clears module variables.

REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.0.75 xsgrids_write_grids (Source File: mod_xsgrids.f90)

INTERFACE:

```
subroutine xsgrids_write_grids(iqmt)
```

USES:

```
use m_getunit
```

INPUT/OUTPUT PARAMETERS:

```
In:
integer(4) :: iqmt ! Considered Q-point
```

DESCRIPTION:

Writes out detailed information about the generated k-grids. Creates the folder `XSGRIDS`.

REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.0.76 rhovalk (Source File: genrhoir.f90)

INTERFACE:

```
Subroutine genrhoir (ik, evecfv, evecsv)
```

USES:

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
ik     : k-point number (in,integer)
evecfv : first-variational eigenvectors (in,complex(nmatmax,nstfv,nspnfv))
evecsv : second-variational eigenvectors (in,complex(nstsv,nstsv))
```

DESCRIPTION:

Generates the partial valence charge density from the eigenvectors at k-point `ik`. In the muffin-tin region, the wavefunction is obtained in terms of its (l, m)-components from both the APW and local-orbital functions. Using a backward spherical harmonic transform (SHT), the wavefunction is converted to real-space and the density obtained from its modulus squared. This density is then transformed with a forward SHT and accumulated in the global variable `rhomt`. A similar proccess is used for the intersitial density in which the wavefunction in real-space is obtained from a Fourier transform of the sum of APW functions. The interstitial density is added to the global array `rhoir`. See routines `wavefmt`, `genshtmat` and `seceqn`.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

1.0.77 autoradmt (Source File: autoradmt.f90)

INTERFACE:

```
Subroutine autoradmt
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Automatically determines the muffin-tin radii from the formula

$$R_i \propto 1 + \zeta |Z_i|^{1/3},$$

where Z_i is the atomic number of the ith species, ζ is a user-supplied constant (~ 0.625). The parameter ζ is stored in `rmtapm(1)` and the value which governs the distance between the muffin-tins is stored in `rmtapm(2)`. When `rmtapm(2)` $= 1$, the closest muffin-tins will touch.

REVISION HISTORY:

```
Created March 2005 (JKD)
Changed the formula, September 2006 (JKD)
```

1.0.78 portstate (Source File: portstate.F90)

INTERFACE:

```
Subroutine portstate (act)
```

USES:

```
Use ioarray
use mod_misc, only: refversion_gitstate
```

DESCRIPTION:

Toggle file format of `STATE.OUT`. If tb2a is true an ASCII file with the name `STATE.xml` is generated and the data from `STATE.OUT` is transferred. If tb2a is false the conversion goes in the other direction. Based upon the routines `readstate` and `writestate`.

REVISION HISTORY:

```
Created 2007 (Sagmeister)
```

1.0.79 energy (Source File: energy.f90)

INTERFACE:

```
Subroutine energy
```

USES:

```
Use modinput
Use modmain
Use mod_hybrids, only: ihyb, exnl
```

DESCRIPTION:

The `energy` subroutine computes the total energy and its individual contributions. The total energy is composed of kinetic, Coulomb, and exchange-correlation energy,

$$E_{\rm tot} = T_{\rm s} + E_C + E_{\rm xc}. \tag{1}$$

The kinetic energy of the non-interacting system is given by

$$T_{\rm s} = \sum_i n_i \epsilon_i - V_{\rm eff}, \tag{2}$$

where n_i are the occupancies and ϵ_i are the eigenvalues of both the core and valence states. The effective potential energy, $V_{\rm eff}$, can be expressed as

$$\begin{aligned} V_{\rm eff} &= \int \rho(\mathbf{r})\, v_{\rm C}(\mathbf{r})\, d\mathbf{r} + \int \rho(\mathbf{r})\, v_{\rm xc}(\mathbf{r})\, d\mathbf{r} \\ &+ \int \mathbf{m}(\mathbf{r}) \cdot [\mathbf{B}_{\rm xc}(\mathbf{r}) + \mathbf{B}_{\rm ext}(\mathbf{r})]\, d\mathbf{r}. \end{aligned} \tag{3}$$

The first and second term of Eq. (3) are the Coulomb potential energy, V_C, and exchange-correlation potential energy, $V_{\rm xc}$, respectively. $\mathbf{m}(\mathbf{r})$ is the magnetization density, and $\mathbf{B}_{\rm xc}$ and $\mathbf{B}_{\rm ext}$ are the exchange-correlation effective magnetic and the external magnetic fields, respectively.
The Coulomb energy consists of the Hartree energy, $E_{\rm H}$, the electron-nuclear energy, $E_{\rm en}$, and the nuclear-nuclear energy, $E_{\rm nn}$,

$$\begin{aligned} E_{\rm C} &= E_{\rm H} + E_{\rm en} + E_{\rm nn} \\ &= \underbrace{(E_{\rm H} + \frac{1}{2}E_{\rm en})}_{\frac{1}{2}V_{\rm C}} + \underbrace{(\frac{1}{2}E_{\rm en} + E_{\rm nn})}_{E_{\rm Madelung}}. \end{aligned} \tag{4}$$

The Madelung energy is given by

$$E_{\rm Madelung} = \frac{1}{2}\sum_\alpha z_\alpha R_\alpha, \tag{5}$$

where for each atom α with nuclear charge z_α

$$R_\alpha = \lim_{r\to 0}\left(v^{\rm C}_{\alpha,00}(r)Y_{00} + \frac{z_\alpha}{r}\right) \tag{6}$$

with $v^{\rm C}_{\alpha,00}$ being the $l = 0$ component of the spherical harmonic expansion of $v_{\rm C}$ in the muffin-tin region. Using Eq. (4), the electron-nuclear and Hartree energies can be expressed as

$$E_{\rm en} = 2\,(E_{\rm Madelung} - E_{\rm nn}) \tag{7}$$

and

$$E_{\rm H} = \frac{1}{2}(V_{\rm C} - E_{\rm en}). \tag{8}$$

$E_{\rm xc}$ is obtained either by integrating the exchange-correlation energy density,

$$E_{\rm xc} \;=\; \int \rho(\mathbf{r})\,\epsilon_{\rm xc}(\mathbf{r})\,d\mathbf{r}, \tag{9}$$

or in the case of exact exchange, the explicit calculation of the Fock exchange integral.
The energy from the external magnetic fields in the muffin-tins, `bfcmt`, is always removed from the total since these fields are non-physical: their field lines do not close. The energy of the physical external field, `bfieldc`, is also not included in the total because this field, like those in the muffin-tins, is used for breaking spin symmetry and taken to be infinitesimal. If this field is intended to be finite, then the associated energy, `engybext`, should be added to the total by hand. See `potxc`, `exxengy` and related subroutines.

REVISION HISTORY:

```
Created Jun 2013
```

1.0.80 addrhocr (Source File: addrhocr.f90)

INTERFACE:

```
Subroutine addrhocr
```

USES:

```
Use modmain
```

DESCRIPTION:

Adds the core density to the muffin-tin and interstitial densities. A uniform background density is added in the interstitial region to take into account leakage of core charge from the muffin-tin spheres.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

1.0.81 writeinfo (Source File: writeinfo.f90)

INTERFACE:

```
Subroutine writeinfo (fnum)
```

USES:

```
      Use modinput
      Use modmain
      use modmpi, only: procs
#ifdef TETRA
      Use modtetra
#endif
```

INPUT/OUTPUT PARAMETERS:

```
    fnum : unit specifier for INFO.OUT file (in,integer)
```

DESCRIPTION:

Outputs basic information about the run to the file `INFO.OUT`. Does not close the file afterwards.

REVISION HISTORY:

```
    Created January 2003 (JKD)
```

1.0.82 packeff (Source File: packeff.f90)

INTERFACE:

```
Subroutine packeff (tpack, n, nu)
```

USES:

```
      Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
    tpack : .true. for packing, .false. for unpacking (in,logical)
    n     : total number of real values stored (out,integer)
    nu    : packed potential (inout,real(*))
```

DESCRIPTION:

Packs/unpacks the muffin-tin and interstitial parts of the effective potential and magnetic field or density into/from the single array `nu`. This array can then be passed directly to the mixing routine. See routine `rfpack`.

REVISION HISTORY:

```
    Created June 2003 (JKD)
    Modified Feb 2014 (UW)
```

1.0.83 rotzflm (Source File: rotzflm.f90)

INTERFACE:

```
Subroutine rotzflm (rot, lmax, n, ld, zflm1, zflm2)
```

INPUT/OUTPUT PARAMETERS:

```
rot   : rotation matrix (in,real(3,3))
lmax  : maximum angular momentum (in,integer)
n     : number of functions to rotate (in,integer)
ld    : leading dimension (in,integer)
zflm1 : coefficients of complex spherical harmonic expansion for each
        function (in,complex(ld,n))
zflm2 : coefficients of rotated functions (out,complex(ld,n))
```

DESCRIPTION:

Rotates a set of functions

$$f_i(\mathbf{r}) = \sum_{lm} f_{lm}^i Y_{lm}(\hat{\mathbf{r}})$$

for all i, given the coefficients f_{lm}^i and a rotation matrix R. This is done by first the computing the Euler angles (α, β, γ) of R^{-1} (see routine `euler`) and then generating the rotation matrix for spherical harmonics, $D_{mm'}^l(\alpha, \beta, \gamma)$, with which

$$Y_{lm}(\theta', \phi') = \sum_{m'} D_{mm'}^l(\alpha, \beta, \gamma) Y_{lm'}(\theta, \phi),$$

where (θ', ϕ') are the angles (θ, ϕ) rotated by R. The matrix D is given explicitly by

$$D_{mm'}^l(\alpha, \beta, \gamma) = \sum_i \frac{(-1)^i \sqrt{(l+m)!(l-m)!(l+m')!(l-m')!}}{(l-m'-i)!(l+m-i)!i!(i+m'-m)!} \times \left(\cos\frac{\beta}{2}\right)^{2l+m-m'-2i} \left(\sin\frac{\beta}{2}\right)^{2i+m'-m} e^{-i(m\alpha+m'\gamma)},$$

where the sum runs over all i which make the factorial arguments non-negative. For improper rotations, i.e. those which are a combination of a rotation and inversion, the rotation is first made proper with $R \to -R$ and D is modified with $D_{mm'}^l \to (-1)^l D_{mm'}^l$.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

1.0.84 fsmfield (Source File: fsmfield.f90)

INTERFACE:

```
Subroutine fsmfield
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Updates the effective magnetic field, $\mathbf{B}_{\rm FSM}$, required for fixing the spin moment to a given value, $\boldsymbol{\mu}_{\rm FSM}$. This is done by adding a vector to the field which is proportional to the difference between the moment calculated in the ith self-consistent loop and the required moment:

$$\mathbf{B}_{\rm FSM}^{i+1}=\mathbf{B}_{\rm FSM}^{i}+\lambda\left(\boldsymbol{\mu}^{i}-\boldsymbol{\mu}_{\rm FSM}\right),$$

where λ is a scaling factor.

REVISION HISTORY:

```
Created March 2005 (JKD)
```

1.0.85 gndstate (Source File: gndstate.f90)

INTERFACE:

```
Subroutine gndstate
```

USES:

```
Use modinput
Use modmain
Use modmpi
Use scl_xml_out_Module
```

DESCRIPTION:

Computes the self-consistent Kohn-Sham ground-state. General information is written to the file `INFO.OUT`. First- and second-variational eigenvalues, eigenvectors and occupancies are written to the unformatted files `EVALFV.OUT`, `EVALSV.OUT`, `EVECFV.OUT`, `EVECSV.OUT` and `OCCSV.OUT`.

REVISION HISTORY:

```
Created October 2002 (JKD)
last revision July 2014 (PP)
```

1.0.86 reciplat (Source File: reciplat.f90)

INTERFACE:

```
Subroutine reciplat
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Generates the reciprocal lattice vectors from the real-space lattice vectors

$$\mathbf{b}_1 = \frac{2\pi}{s}(\mathbf{a}_2 \times \mathbf{a}_3)$$
$$\mathbf{b}_2 = \frac{2\pi}{s}(\mathbf{a}_3 \times \mathbf{a}_1)$$
$$\mathbf{b}_3 = \frac{2\pi}{s}(\mathbf{a}_1 \times \mathbf{a}_2)$$

and finds the unit cell volume $\Omega = |s|$, where $s = \mathbf{a}_1 \cdot (\mathbf{a}_2 \times \mathbf{a}_3)$.
REVISION HISTORY:

```
Created September 2002 (JKD)
```

1.0.87 hartfock (Source File: hartfock.f90)

INTERFACE:

```
Subroutine hartfock
```

USES:

```
Use modmain
Use modinput
```

DESCRIPTION:

Computes the self-consistent Hartree Fock ground state.
REVISION HISTORY:

```
...
```

1.0.88 getevalfv (Source File: getevalfv.f90)

INTERFACE:

```
Subroutine getevalfv (vpl, evalfv)
```

USES:

```
Use modmain
Use modinput
Use modmpi
```

DESCRIPTION:

The file where the (first-variational) eigenvalues are stored is `EVALFV.OUT`. It is a direct-access binary file, the record length of which can be determined with the help of the array sizes and data type information. One record of this file has the following structure

k_{lat}	N_{stfv}	N_{spfv}	E

The following table explains the parts of the record in more detail

name	type	shape	description
k_{lat}	real(8)	3	k-point in lattice coordinates
N_{stfv}	integer	1	number of (first-variational) states (without core states)
N_{spfv}	integer	1	first-variational spins (always equals 1)
E	real(8)	$N_{\text{stfv}} \times N_{\text{spfv}}$	(first-variational) eigenvalue array

REVISION HISTORY:

```
Documentation added, Dec 2009 (S. Sagmeister)
```

1.0.89 findprim (Source File: findprim.f90)

INTERFACE:

```
Subroutine findprim
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

This routine finds the smallest primitive cell which produces the same crystal structure as the conventional cell. This is done by searching through all the vectors which connect atomic positions and finding those which leave the crystal structure invariant. Of these, the three shortest which produce a non-zero unit cell volume are chosen.

REVISION HISTORY:

```
Created April 2007 (JKD)
```

1.0.90 potxc (Source File: potxc.f90)

INTERFACE:

```
subroutine potxc
```

USES:

```
use modmain
use modxcifc
```

DESCRIPTION:

Computes the exchange-correlation potential and energy density. In the muffin-tin, the density is transformed from spherical harmonic coefficients ρ_{lm} to spherical coordinates (θ, ϕ) with a backward spherical harmonic transformation (SHT). Once calculated, the exchange-correlation potential and energy density are transformed with a forward SHT.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

1.0.91 zpotcoul (Source File: zpotcoul.f90)

INTERFACE:

```
Subroutine zpotcoul (nr, nrmax, ld, r, igp0, gpc, jlgpr, ylmgp, sfacgp, &
& zn, zrhomt, zrhoir, zvclmt, zvclir, zrho0)
      Use modinput
```

USES:

```
      Use modmain
#ifdef USEOMP
      use omp_lib
#endif
```

INPUT/OUTPUT PARAMETERS:

```
nr     : number of radial points for each species (in,integer(nspecies))
nrmax  : maximum nr over all species (in,integer)
ld     : leading dimension of r (in,integer)
r      : radial mesh for each species (in,real(ld,nspecies))
igp0   : index of the shortest G+p-vector (in,integer)
gpc    : G+p-vector lengths (in,real(ngvec))
jlgpr  : spherical Bessel functions for evergy G+p-vector and muffin-tin
         radius (in,real(0:lmaxvr+npsden+1,ngvec,nspecies))
ylmgp  : spherical harmonics of the G+p-vectors (in,complex(lmmaxvr,ngvec))
sfacgp : structure factors of the G+p-vectors (in,complex(ngvec,natmtot))
zn     : nuclear charges at the atomic centers (in,real(nspecies))
zrhomt : muffin-tin charge density (in,complex(lmmaxvr,nrmax,natmtot))
zrhoir : interstitial charge density (in,complex(ngrtot))
zvclmt : muffin-tin Coulomb potential (out,complex(lmmaxvr,nrmax,natmtot))
zvclir : interstitial Coulomb potential (out,complex(ngrtot))
zrho0  : G+p=0 term of the pseudocharge density (out,complex)
```

DESCRIPTION:

Calculates the Coulomb potential of a complex charge density by solving Poisson's equation. First, the multipole moments of the muffin-tin charge are determined for the jth atom of the ith species by

$$q_{ij;lm}^{\mathrm{MT}} = \int_0^{R_i} r^{l+2} \rho_{ij;lm}(r) dr + z_{ij} Y_{00}\, \delta_{l,0} \ ,$$

where R_i is the muffin-tin radius and z_{ij} is a point charge located at the atom center (usually the nuclear charge, which should be taken as **negative**). Next, the multipole moments of the continuation of the interstitial density, ρ^{I}, into the muffin-tin are found with

$$q_{ij;lm}^{\mathrm{I}} = 4\pi i^l R_i^{l+3} \sum_{\mathbf{G}} \frac{j_{l+1}(GR_i)}{GR_i} \rho^{\mathrm{I}}(\mathbf{G}) \exp(i\mathbf{G}\cdot\mathbf{r}_{ij}) Y_{lm}^*(\hat{\mathbf{G}}),$$

remembering that

$$\lim_{x\to 0} \frac{j_{l+n}(x)}{x^n} = \frac{1}{(2n+1)!!}\delta_{l,0}$$

should be used for the case $\mathbf{G} = 0$. A pseudocharge is now constructed which is equal to the real density in the interstitial region and whose multipoles are the difference between the real and interstitial muffin-tin multipoles. This pseudocharge density is smooth in the sense that it can be expanded in terms of the finite set of $\mathbf{G}$-vectors. In each muffin-tin the pseudocharge has the form

$$\rho_{ij}^{\mathrm{P}}(\mathbf{r}) = \rho^{\mathrm{I}}(\mathbf{r}-\mathbf{r}_{ij}) + \sum_{lm} \rho_{ij;lm}^{\mathrm{P}} \frac{1}{R_i^{l+3}} \left(\frac{r}{R_i}\right)^l \left(1 - \frac{r^2}{R_i^2}\right)^{N_i} Y_{lm}(\hat{\mathbf{r}})$$

where

$$\rho_{ij;lm}^{\mathrm{P}} = \frac{(2l+2N_i+3)!!}{2_i^N N_i!(2l+1)!!} \left(q_{ij;lm}^{\mathrm{MT}} - q_{ij;lm}^{\mathrm{I}}\right)$$

and $N_i \approx \frac{1}{2} R_i G_{\max}$ is generally a good choice. The pseudocharge in reciprocal-space is given by

$$\rho^{\mathrm{P}}(\mathbf{G}) = \rho^{\mathrm{I}}(\mathbf{G}) + \sum_{ij;lm} 2^{N_i} N_i! \frac{4\pi(-i)^l}{\Omega R_i^l} \frac{j_{l+N_i+1}(GR_i)}{(GR_i)^{N_i+1}} \rho_{ij;lm}^{\mathrm{P}} \exp(-i\mathbf{G}\cdot\mathbf{r}_{ij}) Y_{lm}(\hat{\mathbf{G}})$$

which may be used for solving Poisson's equation directly

$$V^{\mathrm{P}}(\mathbf{G}) = \begin{cases} 4\pi \frac{\rho^{\mathrm{P}}(\mathbf{G})}{G^2} & G > 0 \\ 0 & G = 0 \end{cases} .$$

The usual Green's function approach is then employed to determine the potential in the muffin-tin sphere due to charge in the sphere. In other words

$$V_{ij;lm}^{\mathrm{MT}}(r) = \frac{4\pi}{2l+1} \left(\frac{1}{r^{l+1}} \int_0^r \rho_{ij;lm}^{\mathrm{MT}}(r') r'^{l+2} dr' + r^l \int_r^{R_i} \frac{\rho_{ij;lm}^{\mathrm{MT}}(r')}{r'^{l-1}} dr' \right) + \frac{1}{Y_{00}} \frac{z_{ij}}{r} \delta_{l,0}$$

where the last term is the monopole arising from the point charge. All that remains is to add the homogenous solution of Poisson's equation,

$$V_{ij}^{\mathrm{H}}(\mathbf{r}) = \sum_{lm} V_{ij;lm}^{\mathrm{H}} \left(\frac{r}{R_i}\right)^l Y_{lm}(\hat{\mathbf{r}}),$$

to the muffin-tin potential so that it is continuous at the muffin-tin boundary. Therefore the coefficients, $\rho^{\rm H}_{ij;lm}$, are given by

$$V^{\rm H}_{ij;lm} = 4\pi i^l \sum_{\mathbf{G}} j_l(Gr) V^{\rm P}(\mathbf{G}) \exp(i\mathbf{G}\cdot\mathbf{r}_{ij}) Y^*_{lm}(\hat{\mathbf{G}}) - V^{\rm MT}_{ij;lm}(R_i).$$

Finally note that the **G**-vectors passed to the routine can represent vectors with a non-zero offset, $\mathbf{G}+\mathbf{p}$ say, which is required for calculating Coulomb matrix elements.
REVISION HISTORY:

```
Created April 2003 (JKD)
```

1.0.92 rfmtctof (Source File: rfmtctof.f90)

INTERFACE:

```
Subroutine rfmtctof (rfmt)
```

INPUT/OUTPUT PARAMETERS:

```
  Use modinput
rfmt : real muffin-tin function (in,real(lmmaxvr,nrmtmax,natmtot))
```

DESCRIPTION:

Converts a real muffin-tin function from a coarse to a fine radial mesh by using cubic spline interpolation. See routines `rfinterp` and `spline`.
REVISION HISTORY:

```
Created October 2003 (JKD)
```

1.0.93 lchargelinene (Source File: lchargelinene.f90)

INTERFACE:

```
subroutine lchargelinene
```

USES:

```
use modinput
use modmain
use modmpi, only: rank
```

DESCRIPTION:

Calculate the "Optimal Energy Parameters" by S.Bluegel **REVISION HISTORY:**

```
Created October 2013 (DIN)
```

1.0.94 rfinp (Source File: rfinp.f90)

INTERFACE:

```
Function rfinp (lrstp, rfmt1, rfmt2, rfir1, rfir2)
```

USES:

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
lrstp : radial step length (in,integer)
rfmt1 : first function in real spherical harmonics for all muffin-tins
        (in,real(lmmaxvr,nrmtmax,natmtot))
rfmt2 : second function in real spherical harmonics for all muffin-tins
        (in,real(lmmaxvr,nrmtmax,natmtot))
rfir1 : first real interstitial function in real-space (in,real(ngrtot))
rfir2 : second real interstitial function in real-space (in,real(ngrtot))
```

DESCRIPTION:

Calculates the inner product of two real fuctions over the entire unit cell. The input muffin-tin functions should have angular momentum cut-off `lmaxvr`. In the intersitial region, the integrand is multiplied with the characteristic function, $\tilde{\Theta}(\mathbf{r})$, to remove the contribution from the muffin-tin. See routines `rfmtinp` and `gencfun`.

REVISION HISTORY:

```
Created July 2004 (JKD)
```

1.0.95 gridsize (Source File: gridsize.f90)

INTERFACE:

```
Subroutine gridsize
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Finds the **G**-vector grid which completely contains the vectors with $G < G_{\max}$ and is compatible with the FFT routine. The optimal sizes are given by

$$n_i = \frac{G_{\max}|\mathbf{a}_i|}{\pi} + 1,$$

where $\mathbf{a}_i$ is the ith lattice vector.

REVISION HISTORY:

```
Created July 2003 (JKD)
```

1.0.96 potcoul (Source File: potcoul.f90)

INTERFACE:

```
Subroutine potcoul
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Calculates the Coulomb potential of the real charge density stored in the global variables `rhomt` and `rhoir` by solving Poisson's equation. These variables are converted to complex representations and passed to the routine `zpotcoul`.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

1.0.97 bandstr (Source File: bandstr.f90)

INTERFACE:

```
Subroutine bandstr
```

USES:

```
Use modinput
Use modmain
Use modmpi
Use FoX_wxml
```

DESCRIPTION:

Produces a band structure along the path in reciprocal-space which connects the vertices in the array `vvlp1d`. The band structure is obtained from the second-variational eigenvalues and is written to the file `BAND.OUT` with the Fermi energy set to zero. If required, band structures are plotted to files `BAND_Sss_Aaaaa.OUT` for atom `aaaa` of species `ss`, which include the band characters for each l component of that atom in columns 4 onwards. Column 3 contains the sum over l of the characters. Vertex location lines are written to `BANDLINES.OUT`.

REVISION HISTORY:

```
Created June 2003 (JKD)
Modified June 2012 (DIN)
Modified March 2014 (UW)
Modified June 2018 (SeTi)
```

1.0.98 ggair (Source File: ggair.f90)

INTERFACE:

```
Subroutine ggair (grhoir, gupir, gdnir, g2upir, g2dnir, g3rhoir, &
& g3upir, g3dnir)
```

INPUT/OUTPUT PARAMETERS:

```
  Use modinput
grhoir  : |grad rho| (out,real(ngrtot))
gupir   : |grad rhoup| (out,real(ngrtot))
gdnir   : |grad rhodn| (out,real(ngrtot))
g2upir  : grad^2 rhoup (out,real(ngrtot))
g2dnir  : grad^2 rhodn (out,real(ngrtot))
g3rhoir : (grad rho).(grad |grad rho|) (out,real(ngrtot))
g3upir  : (grad rhoup).(grad |grad rhoup|) (out,real(ngrtot))
g3dnir  : (grad rhodn).(grad |grad rhodn|) (out,real(ngrtot))
```

DESCRIPTION:

Computes $|\nabla\rho|$, $|\nabla\rho^{\uparrow}|$, $|\nabla\rho^{\downarrow}|$, $\nabla^2\rho^{\uparrow}$, $\nabla^2\rho^{\downarrow}$, $\nabla\rho\cdot(\nabla|\nabla\rho|)$, $\nabla\rho^{\uparrow}\cdot(\nabla|\nabla\rho^{\uparrow}|)$ and $\nabla\rho^{\downarrow}\cdot(\nabla|\nabla\rho^{\downarrow}|)$ for the interstitial charge density, as required by the generalised gradient approximation functional for spin-polarised densities. In the case of spin unpolarised calculations, $|\nabla\rho|$, $\nabla^2\rho$ and $\nabla\rho\cdot(\nabla|\nabla\rho|)$ are returned in the arrays `gupir`, `g2upir` and `g3upir`, respectively, while `grhoir`, `gdnir`, `g2dnir`, `g3rhoir` and `g3dnir` are not referenced. See routines `potxc` and `modxcifc`.

REVISION HISTORY:

```
Created October 2004 (JKD)
```

1.0.99 findsym (Source File: findsym.f90)

INTERFACE:

```
Subroutine findsym (apl1, apl2, nsym, lspl, lspn, iea)
```

USES:

```
  Use modinput
  Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
apl1 : first set of atomic positions in lattice coordinates
       (in,real(3,maxatoms,maxspecies))
apl2 : second set of atomic positions in lattice coordinates
       (in,real(3,maxatoms,maxspecies))
nsym : number of symmetries (out,integer)
```

```
lspl : spatial rotation element in lattice point group for each symmetry
       (out,integer(48))
lspn : spin rotation element in lattice point group for each symmetry
       (out,integer(48))
iea  : equivalent atom index for each symmetry
       (out,integer(iea(natmmax,nspecies,48))
```

DESCRIPTION:

Finds the symmetries which rotate one set of atomic positions into another. Both sets of positions differ only by a translation vector and have the same muffin-tin magnetic fields (stored in the global array `bfcmt`). Any symmetry element consists of a spatial rotation of the atomic position vectors followed by a global magnetic rotation: $\{\alpha_S|\alpha_R\}$. In the case of spin-orbit coupling $\alpha_S = \alpha_R$. The symmetries are returned as indices of elements in the Bravais lattice point group. An index to equivalent atoms is stored in the array `iea`.

REVISION HISTORY:

```
Created April 2007 (JKD)
Fixed use of proper rotations for spin, February 2008 (L. Nordstrom)
```

1.0.100 readinput (Source File: setdefault.f90)

INTERFACE:

```
Subroutine setdefault
```

USES:

```
      Use modinput
      Use modmain
#ifdef TETRA
      Use modtetra
#endif
#ifdef XS
      Use modmpi, Only: rank
      Use modxs
#endif
      Use sclcontroll
```

DESCRIPTION:

Sets default values for the input parameters.

REVISION HISTORY:

```
Created September 2002 (JKD)
Additional parmeters for excited states and tetrahedron method
  2004-2008 (Sagmeister)
```

1.0.101 getevecfv (Source File: getevecfv.f90)

INTERFACE:

```
Subroutine getevecfv (vpl, vgpl, evecfv)
```

USES:

```
  Use modmpi
  Use modinput
 use mod_kpoint, only: vkl
 use mod_Gkvector, only: ngkmax, vgkl, ngk
 use mod_eigensystem, only: nmatmax, nmat
  use mod_kpoint, only: vkl_ptr
  use mod_Gkvector, only: ngkmax_ptr, vgkl_ptr, ngk_ptr
  use mod_eigensystem, only: nmatmax_ptr, nmat_ptr
  use mod_eigensystem, only: idxlo
  use mod_eigenvalue_occupancy, only: nstfv
  use mod_spin, only: nspnfv
  use mod_names, only: filetag_evecfv
  use mod_symmetry, only: lsplsymc, isymlat, symlat, symlatc, vtlsymc, ieqatom
  use mod_constants, only: twopi
  use mod_APW_LO, only: nlotot, nlorb, lorbl
  use mod_muffin_tin, only: idxlm
  use mod_atoms, only: natoms, nspecies, idxas
```

DESCRIPTION:

The file where the (first-variational) eigenvectors are stored is `EVECFV.OUT`. It is a direct-access binary file, the record length of which can be determined with the help of the array sizes and data type information. One record of this file corresponds to one k-point in the irreducible Brillouin zone and has the following structure

k_{lat}	N_{mat}	N_{stfv}	N_{spfv}	Φ

The following table explains the parts of the record in more detail

name	type	shape	description
k_{lat}	real(8)	3	k-point in lattice coordinates
N_{mat}	integer	1	(L)APW basis size including local orbitals (maximum over k-points)
N_{stfv}	integer	1	number of (first-variational) states (without core states)
N_{spfv}	integer	1	first-variational spins (2 for spin-spirals, 1 otherwise)
Φ	complex(8)	$N_{\mathrm{mat}} \times N_{\mathrm{stfv}} \times N_{\mathrm{spfv}}$	(first-variational) eigenvector array

REVISION HISTORY:

```
Created Feburary 2007 (JKD)
Fixed transformation error, October 2007 (JKD, Anton Kozhevnikov)
Documentation added, Dec 2009 (S. Sagmeister)
```

```
Fixed l.o. rotation, June 2010 (A. Kozhevnikov)
```

1.0.102 getoccsv (Source File: getoccsv.f90)

INTERFACE:

```
Subroutine getoccsv (vpl, occsvp)
```

USES:

```
Use modmain
Use modinput
Use modmpi
```

DESCRIPTION:

The file where the (second-variational) occupation numbers are stored is `OCCSV.OUT`. The maximum occupancies for spin-unpolarized systems is 2, whereas for spin-polarized systems it is 1. It is a direct-access binary file, the record length of which can be determined with the help of the array sizes and data type information. One record of this file has the following structure

k_{lat}	N_{stsv}	o

The following table explains the parts of the record in more detail

name	type	shape	description
k_{lat}	real(8)	3	k-point in lattice coordinates
N_{stsv}	integer	1	number of (second-variational) states (without core states)
o	real(8)	N_{stsv}	(second-variational) occupation number array

REVISION HISTORY:

```
Documentation added, Dec 2009 (S. Sagmeister)
```

1.0.103 genveffig (Source File: genmeffig.f90)

INTERFACE:

```
Subroutine genmeffig
```

USES:

```
Use modmain
```

DESCRIPTION:

Generates the Fourier transform of the effective potential in the intersitial region. The potential is first multiplied by the characteristic function which zeros it in the muffin-tins. See routine `gencfun`.

REVISION HISTORY:

```
Created January 2004 (JKD)
```

1.0.104 writefermi (Source File: writefermi.f90)

INTERFACE:

```
Subroutine writefermi
```

USES:

```
Use modmain
```

DESCRIPTION:

Writes the Fermi energy to the file `EFERMI.OUT`.

REVISION HISTORY:

```
Created March 2005 (JKD)
```

1.0.105 chgdist (Source File: chgdist.F90)

INTERFACE:

```
Subroutine chgdist(rhomtref,rhoirref)
```

USES:

```
use modmain
```

DESCRIPTION:

Calculated the charge distance between two charge densities of the current and of the last iteration according to the expression

$$\Delta Q = \int_{\Omega} \mathrm{d}^3 r [\rho^{(n)}(\mathbf{r}) - \rho^{(n-1)}(\mathbf{r})].$$

Based on the routine `charge`.

REVISION HISTORY:

```
Created 2010 (Sagmeister)
Modified 2014 (DIN)
```

1.0.106 writegeometryxml (Source File: writegeometryxml.f90)

INTERFACE:

```
Subroutine writegeometryxml (topt)
```

USES:

```
Use modmain
Use modinput
Use modspdeflist
Use modspdb
Use FoX_wxml
```

INPUT/OUTPUT PARAMETERS:

```
topt : if .true. then the filename will be {\ttgeometry_opt.xml}, otherwise
       {\tt geometry.xml} (in,logical)
```

DESCRIPTION:

Outputs the lattice vectors and atomic positions to file, in a format which may be then used directly in `input.xml`.

REVISION HISTORY:

```
Created January 2004 (JKD)
```

1.0.107 plot1d (Source File: plot1d.f90)

INTERFACE:

```
Subroutine plot1d (labels, nf, lmax, ld, rfmt, rfir, plotdef)
```

USES:

```
Use modinput
use modmpi
Use FoX_wxml
use mod_muffin_tin
use mod_atoms
use mod_Gvector
use modplotlabels
use mod_plotting
```

INPUT/OUTPUT PARAMETERS:

```
lables : plot labels (character*)
nf     : number of functions (in,integer)
lmax   : maximum angular momentum (in,integer)
ld     : leading dimension (in,integer)
rfmt   : real muffin-tin function (in,real(ld,nrmtmax,natmtot,nf))
rfir   : real intersitial function (in,real(ngrtot,nf))
plotdef: type(plot1d) defining plotregion
```

DESCRIPTION:

Produces a 1D plot of the real functions contained in arrays `rfmt` and `rfir` along the lines connecting the vertices in the global array `vvlp1d`. See routine `rfarray`.

REVISION HISTORY:

```
Created June 2003 (JKD)
```

1.0.108 findband (Source File: findband.f90)

INTERFACE:

```
Subroutine findband (findlinentype, l, k, nr, r, vr, de0, etol, e0, tfnd)
```

INPUT/OUTPUT PARAMETERS:

```
l   : angular momentum quantum number (in,integer)
k   : quantum number k, zero if Dirac eqn. is not to be used (in,integer)
nr  : number of radial mesh points (in,integer)
r   : radial mesh (in,real(nr))
vr  : potential on radial mesh (in,real(nr))
de0 : default energy step size (in,real)
e   : input energy and returned band energy (inout,real)
```

DESCRIPTION:

Finds the band energies for a given radial potential and angular momentum. This is done by first searching upwards in energy until the radial wavefunction at the muffin-tin radius is zero. This is the energy at the top of the band, denoted $E_{\rm t}$. A downward search is now performed from $E_{\rm t}$ until the slope of the radial wavefunction at the muffin-tin radius is zero. This energy, $E_{\rm b}$, is at the bottom of the band. The band energy is taken as $(E_{\rm t}+E_{\rm b})/2$. If either $E_{\rm t}$ or $E_{\rm b}$ cannot be found then the band energy is set to the input value.

REVISION HISTORY:

```
Created September 2004 (JKD)
```

1.0.109 ggamt (Source File: ggamt.f90)

INTERFACE:

```
Subroutine ggamt (is, ia, grhomt, gupmt, gdnmt, g2upmt, g2dnmt, &
& g3rhomt, g3upmt, g3dnmt)
```

USES:

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
is      : species number (in,integer)
ia      : atom number (in,integer)
grhomt  : |grad rho| (out,real(lmmaxvr,nrmtmax))
gupmt   : |grad rhoup| (out,real(lmmaxvr,nrmtmax))
gdnmt   : |grad rhodn| (out,real(lmmaxvr,nrmtmax))
g2upmt  : grad^2 rhoup (out,real(lmmaxvr,nrmtmax))
g2dnmt  : grad^2 rhodn (out,real(lmmaxvr,nrmtmax))
g3rhomt : (grad rho).(grad |grad rho|) (out,real(lmmaxvr,nrmtmax))
g3upmt  : (grad rhoup).(grad |grad rhoup|) (out,real(lmmaxvr,nrmtmax))
g3dnmt  : (grad rhodn).(grad |grad rhodn|) (out,real(lmmaxvr,nrmtmax))
```

DESCRIPTION:

Computes $|\nabla\rho|$, $|\nabla\rho^{\uparrow}|$, $|\nabla\rho^{\downarrow}|$, $\nabla^2\rho^{\uparrow}$, $\nabla^2\rho^{\downarrow}$, $\nabla\rho\cdot(\nabla|\nabla\rho|)$, $\nabla\rho^{\uparrow}\cdot(\nabla|\nabla\rho^{\uparrow}|)$ and $\nabla\rho^{\downarrow}\cdot(\nabla|\nabla\rho^{\downarrow}|)$ for a muffin-tin charge density, as required by the generalised gradient approximation functional for spin-polarised densities. In the case of spin unpolarised calculations, $|\nabla\rho|$, $\nabla^2\rho$ and $\nabla\rho\cdot(\nabla|\nabla\rho|)$ are returned in the arrays `gupmt`, `g2upmt` and `g3upmt`, respectively, while `grhomt`, `gdnmt`, `g2dnmt`, `g3rhomt` and `g3dnmt` are not referenced. The input densities are in terms of real spherical harmonic expansions but the returned functions are in spherical coordinates. See routines `potxc`, `modxcifc`, `gradrfmt`, `genrlm` and `genshtmat`.

REVISION HISTORY:

```
Created April 2004 (JKD)
```

1.0.110 plot2d (Source File: plot2d.f90)

INTERFACE:

```
Subroutine plot2d (labels, nf, lmax, ld, rfmt, rfir, plotdef)
```

USES:

```
Use modinput
use mod_muffin_tin
use mod_atoms
use mod_Gvector
Use FoX_wxml
use modmpi
use modplotlabels
use mod_plotting
```

INPUT/OUTPUT PARAMETERS:

```
fname : plot file name character(len=*)
nf    : number of functions (in,integer)
lmax  : maximum angular momentum (in,integer)
ld    : leading dimension (in,integer)
```

```
rfmt : real muffin-tin function (in,real(ld,nrmtmax,natmtot,nf))
rfir : real intersitial function (in,real(ngrtot,nf))
plotdef:type(plot2d) defines plot region
```

DESCRIPTION:

Produces a 2D plot of the real functions contained in arrays `rfmt` and `rfir` on the parallelogram defined by the corner vertices in the global array `vclp2d`. See routine `rfarray`.

REVISION HISTORY:

```
Created June 2003 (JKD)
```

1.0.111 linengy (Source File: linengy.f90)

INTERFACE:

```
Subroutine linengy
```

USES:

```
Use modinput
Use modmain
Use scl_xml_out_Module
```

DESCRIPTION:

Calculates the new linearisation energies for both the APW and local-orbital radial functions. See the routine `findband`.

REVISION HISTORY:

```
Created May 2003 (JKD)
```

1.0.112 plot3d (Source File: plot3d.f90)

INTERFACE:

```
Subroutine plot3d (plotlabels3d, nf, lmax, ld, rfmt, rfir, plotdef)
```

USES:

```
use modplotlabels
use modinput
use mod_muffin_tin
use mod_atoms
use mod_Gvector
use FoX_wxml
use modmpi
```

INPUT/OUTPUT PARAMETERS:

```
plotlabels : plot file number (in,integer)
nf   : number of functions (in,integer)
lmax : maximum angular momentum (in,integer)
ld   : leading dimension (in,integer)
rfmt : real muffin-tin function (in,real(ld,nrmtmax,natmtot,nf))
rfir : real intersitial function (in,real(ngrtot,nf))
```

DESCRIPTION:

Produces a 3D plot of the real functions contained in arrays `rfmt` and `rfir` in the parallelepiped defined by the corner vertices in the global array `vclp3d`. See routine `rfarray`.

REVISION HISTORY:

```
Created June 2003 (JKD)
Modified, October 2008 (F. Bultmark, F. Cricchio, L. Nordstrom)
Modified, February 2011 (DIN)
Fixed a bug in gengrid, February 2014 (DIN)
```

1.0.113 rhoinit (Source File: rhoinit.f90)

INTERFACE:

```
Subroutine rhoinit
```

USES:

```
      Use modinput
      Use modmain
#ifdef USEOMP
      use omp_lib
#endif
```

DESCRIPTION:

Initialises the crystal charge density. Inside the muffin-tins it is set to the spherical atomic density. In the interstitial region it is taken to be constant such that the total charge is correct. Requires that the atomic densities have already been calculated.

REVISION HISTORY:

```
Created January 2003 (JKD)
```

1.0.114 writepchgs (Source File: writepchgs.F90)

INTERFACE:

```
subroutine writepchgs(fnum,lmax)
```

USES:

```
use modinput
use modmain
```

DESCRIPTION:

Write partial charges to file.

REVISION HISTORY:

```
Created 2010 (Sagmeister)
```

1.0.115 rhovalk (Source File: rhovalk.f90)

INTERFACE:

```
Subroutine rhovalk (ik, evecfv, evecsv)
```

USES:

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
ik     : k-point number (in,integer)
evecfv : first-variational eigenvectors (in,complex(nmatmax,nstfv,nspnfv))
evecsv : second-variational eigenvectors (in,complex(nstsv,nstsv))
```

DESCRIPTION:

Generates the partial valence charge density from the eigenvectors at k-point `ik`. In the muffin-tin region, the wavefunction is obtained in terms of its (l, m)-components from both the APW and local-orbital functions. Using a backward spherical harmonic transform (SHT), the wavefunction is converted to real-space and the density obtained from its modulus squared. This density is then transformed with a forward SHT and accumulated in the global variable `rhomt`. A similar proccess is used for the intersitial density in which the wavefunction in real-space is obtained from a Fourier transform of the sum of APW functions. The interstitial density is added to the global array `rhoir`. See routines `wavefmt`, `genshtmat` and `seceqn`.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

1.0.116 genapwfr (Source File: genapwfr.f90)

INTERFACE:

`Subroutine genapwfr`

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Generates the APW radial functions. This is done by integrating the scalar relativistic Schrödinger equation (or its energy deriatives) at the current linearisation energies using the spherical part of the effective potential. The number of radial functions at each l-value is given by the variable `apword` (at the muffin-tin boundary, the APW functions have continuous derivatives up to order `apword` $-$ 1). Within each l, these functions are orthonormalised with the Gram-Schmidt method. The radial Hamiltonian is applied to the orthonormalised functions and the results are stored in the global array `apwfr`.

REVISION HISTORY:

```
Created March 2003 (JKD)
```

1.0.117 getevalsv (Source File: getevalsv.f90)

INTERFACE:

```
subroutine getevalsv(vpl, evalsvp)
```

USES:

```
use modmain
use modinput
use modmpi
```

DESCRIPTION:

The file where the (second-variational) eigenvalues are stored is `EVALSV.OUT`. It is a direct-access binary file, the record length of which can be determined with the help of the array sizes and data type information. One record of this file has the following structure

k_{lat}	N_{stsv}	E

The following table explains the parts of the record in more detail

name	type	shape	description
k_{lat}	real(8)	3	k-point in lattice coordinates
N_{stsv}	integer	1	number of (second-variational) states (without core states)
E	real(8)	N_{stsv}	(second-variational) eigenvalue array

REVISION HISTORY:

```
Documentation added, Dec 2009 (S. Sagmeister)
Consmetic changes, added more comments. 2016 (Aurich)
```

1.0.118 scf_cycle (Source File: scf_cycle.f90)

INTERFACE:

```
subroutine scf_cycle(verbosity)
```

USES:

```
Use modinput
Use modmain
Use modmpi
Use scl_xml_out_Module
Use TS_vdW_module, Only: C6ab, R0_eff_ab
Use mod_hybrids, only: ihyb
```

DESCRIPTION:

REVISION HISTORY:

```
Created February 2013 (DIN)
```

1.0.119 xasinit (Source File: xasinit.f90)

INTERFACE:

```
subroutine xasinit
```

DESCRIPTION:

This is the main initialization subroutine of the BSE-XAS program.
USES:

```
Use modmain
Use modmpi
Use modinput
Use modxs
Use modxas
Implicit none
Integer :: is, ist, m, ic, ias, l, ir, ia, k
```

REVISION HISTORY:

```
Created June 2015 by C. Vorwerk
```

1.0.120 setup_pwmat (Source File: m_setup_pwmat.f90)

INTERFACE:

```
subroutine setup_pwmat(pwmat, iqmt, igqmt)
```

INPUT/OUTPUT PARAMETERS:

```
In:
  integer(4) :: iqmt            ! Index of momentum transfer
  integer(4) :: igqmt           ! Index of G+qmt
Out:
  complex(8) :: pwmat(hamsize)  ! Plane wave matrix elemens
```

DESCRIPTION:

The routine generates the plane wave matrix elements

$$\tilde{M}_\alpha(\vec{G},\vec{q}_{\rm mt}) = \sqrt{f_{o_\alpha,\vec{k}_{\alpha+\vec{q}_{\rm mt}/2}} - f_{u_\alpha,\vec{k}_{\alpha-\vec{q}_{\rm mt}/2}}}\langle u_\alpha \vec{k}_{\alpha-\vec{q}_{\rm mt}/2}|e^{-i(\vec{G}_{\rm mt}+\vec{q}_{\rm mt})r}|o_\alpha\vec{k}_{\alpha+\vec{q}_{\rm mt}}\rangle$$

Alpha is the combined index used in the BSE Hamiltonian.

REVISION HISTORY:

```
Created. (Aurich)
```

1.0.121 setup_pwmat_dist (Source File: m_setup_pwmat.f90)

INTERFACE:

```
subroutine setup_pwmat_dist(dpwmat, iqmt, igqmt, binfo)
```

INPUT/OUTPUT PARAMETERS:

```
In:
  integer(4) :: iqmt            ! Index of momentum transfer
  integer(4) :: igqmt           ! Index of G+qmt
  type(blacsinfo) :: binfo      ! Info type for BLACS grid
Out:
  type(dzmat) :: dpwma          ! 2D block cyclic distributed plane wave
                                ! matrix elements
```

DESCRIPTION:

The routine generates the plane wave matrix elements

$$\tilde{M}_\alpha(G,qmt) = \sqrt{f_{v_\alpha,\vec{k}_\alpha} - f_{c_\alpha,\vec{k}_\alpha+\vec{q}_{\rm mt}}}\langle v_\alpha\vec{k}_\alpha|e^{-i(G+qmt)r}|c_\alpha\vec{k}_\alpha+qmt\rangle$$

Alpha is the combined index used in the BSE Hamiltonian.

REVISION HISTORY:

```
Created. (Aurich)
```

1.0.122 xsgeneigveclauncher (Source File: xsgeneigveclauncher.f90)

INTERFACE:

```
subroutine xsgeneigveclauncher()
```

USES:

```
use modmpi
use modinput, only: input
use mod_qpoint, only: nqpt, vql
use modxs, only: unitout, totalqlmt, tscreen, qvkloff
use m_genfilname, only: genfilname
use m_writegqpts, only: writegqpts
use mod_misc, only: filext
use mod_xsgrids
use mod_Gkvector, only: gkmax
```

DESCRIPTION:

Wrapper routine for `b_xsgeneigvec`. Launches one-shot ground state calculations needed for Q-dependent BSE. Note: First Q-point in the Q-point list needs to be the Gamma point.
REVISION HISTORY:

```
Created. 2017 (Aurich)
```

1.0.123 writeqmtpts (Source File: writeqmtpts.f90)

INTERFACE:

```
subroutine writeqmtpts
```

USES:

```
use modxs, only: nqmt, vqlmt, totalqcmt, vqcmt, ivgmt, vgcmt
use m_getunit
use m_genfilname
```

DESCRIPTION:

Writes the momentum transfer **Q**-points
REVISION HISTORY:

```
Created 2017 (BA)
```

1.0.124 angavsc0 (Source File: angavsc0.F90)

INTERFACE:

```
subroutine angavsc0(n, nmax, scrnh, scrnw, scrn, scieff)
```

USES:

```
use modmpi
use mod_constants, only: zzero, pi, twopi, fourpi
use mod_qpoint, only: ngridq
use mod_lattice, only: omega, binv
use modinput, only: input
use modxs, only: dielten0, dielten, symt2, tleblaik,&
               & lmmaxdielt, sptclg
use invert
```

INPUT/OUTPUT PARAMETERS:

```
IN:
n, integer               : Number of G+q vectors for current q-point
nmax, integer            : Maximum number of G+q vectors over all q-points
scrnh(3,3), complex(8)   : Head of dielectric matrix
scrnw(n,2,3), complex(8) : Wings of dielectric matrix
scrn(n,n), complex(8)    : Body of dielectric matrix
OUT:
scieff(nmax,nmax), complex(8) : Screened coulomb potential
```

DESCRIPTION:

This routine deals with the divergences of $\frac{\tilde{\varepsilon}_{G,G'}(q,\omega=0)}{|G+q||G'+q|}$ for $G = G' = q = 0$ by averaging around the Gamma point.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1 Fortran: Module Interface modxas (Source File: mod_variation.f90)

Contains additional global variables required for XAS calculations within the BSE EXCITING code.

REVISION HISTORY:

```
Created JUNE 2015 by Christian Vorwerk
```

1.1.1 acscq (Source File: avscq.F90)

INTERFACE:

```
subroutine avscq(iqr, n, nmax, scrn, scieff)
```

USES:

```
use modinput, only: input
use mod_qpoint, only: iqmap
use modxs, only: ivqr
use modmpi
use invert
```

INPUT/OUTPUT PARAMETERS:

```
IN:
iqr, integer             : Index of the reduced q-point to be considered
n, integer               : Number of G+q vectors for current q-point
nmax, integer            : Maximum number of G+q vectors over all q-points
scrn(n,n), complex(8)    : Body of dielectric matrix
OUT:
scieff(nmax,nmax), complex(8) : Screened coulomb potential
```

DESCRIPTION:

By default this just calculates $\frac{\tilde{\varepsilon}_{G,G'}(q,\omega=0)}{|G+q||G'+q|}$. There are options controlling how the dielectric matrix should be inverted. There are also possible averaging and extrapolation schemes.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1.2 dfq (Source File: dfq.f90)

INTERFACE:

```
subroutine dfq(iq)
```

USES:

```
use mod_misc, only: filext
use modinput, only: input
use modmpi, only: procs, rank, barrier
use mod_misc, only: task
use mod_constants, only: zzero, zone, zi, krondelta
use mod_kpoint, only: nkpt, wkpt
use mod_qpoint, only: nqpt, vql
use mod_lattice, only: omega
use modxs, only: tfxcbse, tscreen, bzsampl, wpari,&
             & wparf, ngq, fnpmat,&
             & fnetim, fnxtim, fnemat, fnchi0,&
             & fnchi0_t, qvkloff, istocc0, istocc,&
             & istunocc0, istunocc, isto0, isto,&
             & istu0, istu, unitout, nst1,&
             & nst2, nst3, nst4, istl1,&
```

```
                & istu1, istl2, istu2, istl3,&
                & istu3, istl4, istu4, deou,&
                & deuo, docc12, docc21, xiou,&
                & xiuo, pmou, pmuo, nwdf,&
                & bsed, ikmapikq, tordf, symt2,&
                & bcbs, filexteps,&
                & eps0dirname, scrdirname, timingdirname
#ifdef TETRA
  use modxs, only: fnwtet
  use mod_eigenvalue_occupancy, only: nstsv, evalsv, efermi
  use modtetra
#endif
  use m_genwgrid
  use m_getpemat
  use m_dftim
  use m_gettetcw
  use m_putx0
  use m_getunit
  use m_writevars
  use m_filedel
  use m_genfilname
  use m_ematqk
  use m_writecmplxparts
  use m_putgeteps0
  use mod_variation, only: ematqk_sv
```

INPUT/OUTPUT PARAMETERS:

```
  In:
  integer(4) :: iq  ! q-point index
```

DESCRIPTION:

Calculates the symmetrized Kohn-Sham response function $\tilde{\chi}^0_{\mathbf{GG'}}(\mathbf{q},\omega)$ for one $\mathbf{q}$-point according to

$$\tilde{\chi}^0_{\mathbf{GG'}}(\mathbf{q},\omega) = \sum_{ou\mathbf{k}} \left[\tilde{M}^{\mathbf{G}}_{ou\mathbf{k}}(\mathbf{q}) \tilde{M}^{\mathbf{G'}}_{ou\mathbf{k}}(\mathbf{q})^* w_{ou\mathbf{k}}(\mathbf{q},\omega) + \tilde{M}^{\mathbf{G}}_{uo\mathbf{k}}(\mathbf{q}) \tilde{M}^{\mathbf{G'}}_{uo\mathbf{k}}(\mathbf{q})^* w_{uo\mathbf{k}}(\mathbf{q},\omega) \right]$$

It is related to the Kohn-Sham response function $\chi^0_{\mathbf{GG'}}(\mathbf{q},\omega)$ by

$$\tilde{\chi}^0_{\mathbf{GG'}}(\mathbf{q},\omega) = v^{\frac{1}{2}}_{\mathbf{G}}(\mathbf{q}) \chi^0_{\mathbf{GG'}}(\mathbf{q},\omega) v^{\frac{1}{2}}_{\mathbf{G'}}(\mathbf{q})$$

and is well defined in the limit as $\mathbf{q}$ goes to zero. The symmetrized matrix elements are defined as

$$\tilde{M}^{\mathbf{G}}_{ou\mathbf{k}}(\mathbf{q}) = v^{\frac{1}{2}}_{\mathbf{G}}(\mathbf{q}) M^{\mathbf{G}}_{ou\mathbf{k}}(\mathbf{q}),$$

where

$$M^{\mathbf{G}}_{ou\mathbf{k}}(\mathbf{q}) = \langle o\mathbf{k} | e^{-i(\mathbf{G}+\mathbf{q})\mathbf{r}} | u\mathbf{k}+\mathbf{q} \rangle.$$

For $\mathbf{G} = 0$ we have to consider three vectors stemming from the limits as $\mathbf{q} \to 0$ along the Cartesian basis vectors $\mathbf{e_i}$, i.e., we can think of $\mathbf{0}_1, \mathbf{0}_2, \mathbf{0}_3$ in place of $\mathbf{0}$. The weights $w_{\mathrm{ou}\mathbf{k}}(\mathbf{q}, \omega)$ are defined as

$$w_{nm\mathbf{k}}(\mathbf{q}, \omega) = \lambda_{\mathbf{k}} \frac{f_{n\mathbf{k}} - f_{m\mathbf{k}+\mathbf{q}}}{\varepsilon_{n\mathbf{k}} - \varepsilon_{m\mathbf{k}+\mathbf{q}} + \Delta_{n\mathbf{k}} - \Delta_{m\mathbf{k}+\mathbf{q}} + \omega + i\eta}$$

in the case where we use a Lorentzian broadening η. In the above expression $\lambda_{\mathbf{k}}$ is the weight of the $\mathbf{k}$-point, $\varepsilon_{n\mathbf{k}}$ and $\varepsilon_{m\mathbf{k}+\mathbf{q}}$ are the DFT Kohn-Sham energies, $\Delta_{n\mathbf{k}}$ and $\Delta_{m\mathbf{k}+\mathbf{q}}$ are the scissors corrections that are non-zero in the case where $m\mathbf{k} + \mathbf{q}$ corresponds to a conduction state. The indices o and u denote *at least partially occupied* and *at least partially unoccupied* states, respectively. The symmetrized Kohn-Sham response function can also be calculated for imaginary frequencies $i\omega$ without broadening η. In this case the replacement

$$\omega + i\eta \mapsto i\omega$$

is applied to the expressions for the weights. Optionally, the weights can be calculated with the help of the linear tetrahedron method (including Bloechl's correction). This routine can be run with MPI parallelization for energies.

REVISION HISTORY:

```
Created March 2005 (Sagmeister)
Added band and k-point analysis, 2007-2008 (Sagmeister)
Changed parts that are unique to execution of dfq with tscreen = true (Aurich)
  Changed Plane wave matrix elements construction call
  Changed write out of EPS
  Added possibility to fully calculate anti-resonant part
```

1.1.3 genphasedm (Source File: genphasedm.F90)

INTERFACE:

```
Subroutine genphasedm(iq, jsym, nmax, n, phfdm, tphf)
```

USES:

```
use modinput
use modxs, only: igqig
use mod_Gvector, only: ivg
use mod_symmetry, only: vtlsymc
use mod_constants, only: twopi
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  integer(4), iq : Index of non-reduced q-point
  integer(4), jsym : Index of crystal symmetry operation that maps qr onto q
  integer(4), nmax : Maximal number of G+q vectors over all q
  integer(4), n    : Number of G+q vectors for current q
OUT:
```

```
complex(8), phfdm(nmax, nmax) : Phase factor for the dielectric matrix
                                due to translations
logical, tphf : True if non-trivial phase appears at least for one (G,Gp) component
```

DESCRIPTION:

The routine computed the phase factors needed to reduce the dielectric matrix at any q-point to corresponding matrix at a reduces q-point. The phases occur due to the translational vector $\vec{t}$ of the crystal symmetry operation chosen to reduce $\vec{q}$ to $\vec{q}_{\rm r}$. The rotaional effects of the symmetry operation are accounted for differently by remapping the set of $\left\{\vec{G}+\vec{q}\right\}$ vectors to the set of $\left\{\vec{G}+\vec{q}_{\rm r}\right\}$ vectors. It is $\epsilon_{\vec{G},\vec{G}'}(\vec{q}) = e^{i2*pi(\vec{G}-\vec{G}')\cdot\vec{t}}\epsilon_{\tilde{\vec{G}},\tilde{\vec{G}}'}(\vec{q}_r)$

REVISION HISTORY:

```
Added to documentation scheme. Note: Description may be buggy. (Aurich 2016)
```

1.1.4 xcd_pwca (Source File: xcd_pwca.F90)

INTERFACE:

```
Subroutine xcd_pwca (n, rho, dvx, dvc)
```

INPUT/OUTPUT PARAMETERS:

```
n     : number of density points (in,integer)
rho   : charge density (in,real(n))
dvx   : exchange potential derivative (out,real(n))
dvc   : correlation potential derivative (out,real(n))
```

DESCRIPTION:

Spin-unpolarised exchange-correlation potential derivative of the Perdew-Wang parameterisation of the Ceperley-Alder electron gas, Phys. Rev. B 45, 13244 (1992) and Phys. Rev. Lett. 45, 566 (1980). Based upon the routine `xc_pwca`.

REVISION HISTORY:

```
Created February 2007 (Sagmeister)
```

1.1.5 putscreen (Source File: putscreen.F90)

INTERFACE:

```
subroutine putscreen(un, tq0, n, chi0, chi0h, chi0w)
```

USES:

```
use mod_constants, only: krondelta
```

INPUT/OUTPUT PARAMETERS:

```
In:
integer :: un  ! Unit to wirte to
logical :: tp0 ! Flag if iq is the q=0 q-point
integer :: n   ! Number of G+q vectros
complex(8) :: ch0(n,n)    ! Body of RPA density-density response matrix
complex(8) :: ch0h(3,3)   ! Body of RPA density-density response matrix
complex(8) :: ch0w(n,2,3) ! Wings of RPA density-density response matrix
```

DESCRIPTION:

Writes the microscopic V-symmetrized Kohn-Sham dielectric function/tensor $\tilde{\epsilon}^0_{\mathbf{GG'}}(\mathbf{q},\omega) = \delta_{\mathbf{GG'}} - \tilde{\chi}^0_{\mathbf{GG'}}(\mathbf{q},\omega)$ for fixed frequency and q point to a human readable text file. Is used only for $\omega = 0$.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1.6 scrcoulintlauncher (Source File: scrcoulintlauncher.f90)

INTERFACE:

```
subroutine scrcoulintlauncher
```

USES:

```
use modmpi
use modxs, only: unitout
use modinput, only: input
use modbse
```

DESCRIPTION:

Launches the calculation of the direct term of the Bethe-Salpeter Hamiltonian for each specified momentum transfer vector.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.1.7 writeqpts (Source File: writeqpts.F90)

INTERFACE:

```
subroutine writeqpts
```

USES:

```
use mod_qpoint, only: nqpt, vql, vqc
use mod_misc, only: task
use modxs, only: ngq, nqptr, vqlr, vqcr, wqptr
use m_getunit
use m_genfilname
```

DESCRIPTION:

Writes the **q**-points in lattice coordinates, weights and number of $\mathbf{G}+\mathbf{q}$-vectors to the file `QPOINTS.OUT`. Based on the routine `writekpts`.

REVISION HISTORY:

Created October 2006 (Sagmeister)

1.1.8 findgroupq (Source File: findgroupq.F90)

INTERFACE:

```
Subroutine findgroupq (tfbz, vql, epslat, bvec, symlat, nsymcrys, &
& lsplsymc, nsymcrysq, scqmap, ivscwrapq)
  use modmpi
```

DESCRIPTION:

Find the (little) group of **q** (which includes finding the small group of **q**). All symmetries, where the rotational part transforms **q** into an equivalent vector are collected for the small group of **q**. Inclusion of non-primitive translations yields the little group of **q**.

REVISION HISTORY:

Created March 2006 (Sagmeister)

1.1.9 genkcpts (Source File: bsedgridinit.F90)

Subroutine bsedgridinit () **USES:**

```
Use modmain
Use modxs
Use modinput
Use m_gndstateq
Use modmpi
Use m_genfilname
Use m_getunit
Implicit None
Character (77) :: string
```

DESCRIPTION:

Initializes for one BSE calculation within a double grid loop.

REVISION HISTORY:

```
Created January 2014, S. Kontur
```

1.1.10 setup_bse_tr (Source File: m_setup_bse.f90)

INTERFACE:

```
subroutine setup_bse_tr(iqmt, smat, cmat, cpmat)
```

USES:

```
use modmpi
use modinput, only: input
use mod_constants, only: zzero, zone
use modbse, only: hamsize
use modxs, only: unitout
use m_hesolver
use m_sqrtzmat
use m_invertzmat
```

INPUT/OUTPUT PARAMETERS:

```
In:
  integer(4) :: iqmt  ! Index of momentum transfer Q

Out:
  complex(8) :: smat(:,:)  ! Aux. EVP matrix S=(A-B)^{1/2}(A+B)(A-B)^{1/2}
  complex(8) :: cmat(:,:)  ! C = (A-B)^{1/2}
  complex(8) :: cpmat(:,:) ! C'= (A-B)^{-1/2}
```

DESCRIPTION:

The routine sets up the auxilliary EVP matrix.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.1.11 setup_bse_block (Source File: m_setup_bse.f90)

INTERFACE:

```
subroutine setup_bse_block(ham, iqmt, fcoup)
```

USES:

```
use modmpi
use modinput, only: input
use mod_constants, only: zzero, zone
use modxs, only: unitout
```

```
use modbse, only: de, nou_bse_max, hamsize, kmap_bse_rg,&
                & kousize, nkkp_bse, nk_bse, ofac,&
                & scclifbasename, exclifbasename,&
                & scclicfbasename,&
                & infofbasename,&
                & vwdiffrr, vwdiffar
use m_getunit
use m_genfilname
use m_putgetbsemat
use m_writecmplxparts
```

INPUT/OUTPUT PARAMETERS:

```
In:
  integer(4) :: iqmt  ! Index of momentum transfer Q
  logical    :: fcoup ! If true, builds RA instead of RR block of BSE matrix
In/Out:
  complex(8) :: ham(:,:) ! RR or RA block of BSE-Hamiltonian matrix
```

DESCRIPTION:

The routine sets up the resonant-resonant or resonant-antiresonant block of the BSE-Hamiltonian matrix. The routine reads `EXCLI.OUT` and `SCCLI.OUT` (`EXCLIC.OUT` and `SCCLIC.OUT`).

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.1.12 setup_bse_tr_dist (Source File: m_setup_bse.f90)

INTERFACE:

```
subroutine setup_bse_tr_dist(iqmt, binfo, smat, cmat, cpmat)
```

USES:

```
use modmpi
use modscl
use modinput, only: input
use mod_constants, only: zzero, zone
use modxs, only: unitout
use modbse, only: hamsize
use m_dhesolver
use m_sqrtzmat
use m_invertzmat
use m_writecmplxparts
use m_dzmatmult
```

INPUT/OUTPUT PARAMETERS:

```
In:
  integer(4) :: iqmt  ! Index of momentum transfer Q
  type(blacsinfo) :: binfo  ! Info struct for BLACS grid on which the
  matrices are distributed

In/Out:
  type(dzmat), optional :: smat(:,:)  ! Aux. EVP matrix S=(A-B)^{1/2}(A+B)(A-B)^{1/2}
  type(dzmat), optional :: cmat(:,:)  ! C = (A-B)^{1/2}
  type(dzmat), optional :: cpmat(:,:) ! C'= (A-B)^{-1/2}
```

DESCRIPTION:

The routine sets up the auxilliary EVP matrix using block cyclic distributed matrices for ScaLapack.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.1.13 setup_bse_block_dist (Source File: m_setup_bse.f90)

INTERFACE:

```
subroutine setup_bse_block_dist(ham, iqmt, fcoup, binfo)
```

USES:

```
  use modmpi
  use modscl
  use modinput, only: input
  use mod_constants, only: zzero, zone
  use modxs, only: unitout
  use modbse, only: nou_bse_max, kmap_bse_rg,&
                 & kousize, nkkp_bse, nk_bse, ofac,&
                 & scclifbasename, exclifbasename,&
                 & scclicfbasename,&
                 & infofbasename, &
                 & vwdiffrr, vwdiffar, hamsize
  use m_getunit
  use m_genfilname
  use m_putgetbsemat
  use m_writecmplxparts
```

INPUT/OUTPUT PARAMETERS:

```
In:
  integer(4) :: iqmt ! Index of momentum transfer Q
  logical :: fcoup   ! If true, builds RA instead of RR block of BSE matrix
  type(blacsinfo) :: binfo ! Info type of the BLACS grid
```

```
In/Out:
  type(dzmat) :: ham ! 2D block cyclic distributed RR or RA
                     ! block of BSE-Hamiltonian matrix
```

DESCRIPTION:

The routine sets up the content of the local array of the 2d block cyclic distributed resonant-resonant or resonant-antiresonant block of the BSE-Hamiltonian matrix. Process 0 reads `EXCLI.OUT` and `SCCLI.OUT` (`EXCLIC.OUT` and `SCCLIC.OUT` or `SCCLICTI.OUT`) for each `ikkp` record and send the data block-wise to the responsible processes. If the matrix is not to be distributed the routine calls `setup_bse` instead.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.1.14 buildham (Source File: m_setup_bse.f90)

INTERFACE:

```
subroutine buildham(fc, hamblck, ig, jg, ib, jb, occ1, occ2, scc, exc)
```

USES:

```
use modbse, only: de
use modinput, only: input
```

INPUT/OUTPUT PARAMETERS:

```
In:
logical :: fc ! Build RA instead of RR
integer(4) :: ig, jg ! Position of sub block in global matrix
integer(4) :: ib, jb ! Sub block size
real(8)    :: occ1(ib), occ2(jb)    ! Occupation factors
complex(8), optional :: scc(ib, jb) ! Screened Coulomb interaction
complex(8), optional :: exc(ib, jb) ! Exchange interaction
In/Out:
complex(8) :: hamblck(ib,jb)       ! Sub block of BSE-Hamiltonian
```

DESCRIPTION:

The routine returns a sub block of the distributed BSE-Hamiltonian matrix:
$H(i_g : ig + ib - 1, j_g : j_g + jb - 1)$ where each entry is computed according to
$H(i,j) = E(i,j) + F(i)\left(-W(i,j) + 2 * V(i,j)\right) F(j)$
Only if the sub block contains diagonal elements of the matrix the kohn sham transition energies E will be added (RR case only). From the transition energies the gap energy is subtracted and the scissor is added. The exchange term V is added optionally.

The matrix indices correspond to combined indices α:
Where $\alpha = \{\vec{k}_\alpha, o_\alpha, u_\alpha\}$, so that:
$F_\alpha = \sqrt{\left| f_{\vec{k}_{\alpha_1} o_{\alpha_1}} - f_{\vec{k}_{\alpha_1} u_{\alpha_1}} \right|}$
$V_{\alpha_1,\alpha_2} = V_{\vec{k}_{\alpha_1} o_{\alpha_1} u_{\alpha_1}, \vec{k}_{\alpha_2} o_{\alpha_2} u_{\alpha_2}}$
$W_{\alpha_1,\alpha_2} = W_{\vec{k}_{\alpha_1} o_{\alpha_1} u_{\alpha_1}, \vec{k}_{\alpha_2} o_{\alpha_2} u_{\alpha_2}}$
REVISION HISTORY:

Created 2016 (Aurich)

1.1.15 getngqmax (Source File: getngqmax.F90)

INTERFACE:

```
Subroutine getngqmax
```

USES:

```
Use modinput
Use modmain
use modmpi
Use modxs
```

DESCRIPTION:

Determines the largest number of $\mathbf{G}+\mathbf{q}$-vectors with length less than `gqmax` over all the $\mathbf{q}$-points and stores it in the global variable `ngqmax`. This variable is used for allocating arrays. Based upon the routine `getngkmax`.
REVISION HISTORY:

Created October 2006 (Sagmeister)

1.1.16 exccoulint (Source File: exccoulint.f90)

INTERFACE:

```
subroutine exccoulint(iqmt)
```

USES:

```
use mod_misc, only: filext
use mod_constants, only: zone, zzero
use mod_APW_LO, only: lolmax
use mod_lattice, only: omega
use modinput, only: input
use modmpi, only: rank, barrier, mpi_allgatherv_ifc
use modxs, only: xsgnt, unitout,&
               & ngq, nqmt,&
```

```
                & totalqlmt, ivgmt,&
                & kpari, kparf,&
                & ppari, pparf,&
                & bcbs, iqmtgamma,&
                & filext0, usefilext0, iqmt0, iqmt1, ivgigq
  use m_xsgauntgen
  use m_findgntn0
  use m_writegqpts
  use m_genfilname
  use m_getunit
  use modbse
  use m_ematqk
  use m_putgetbsemat
  use mod_xsgrids
  use mod_Gkvector, only: gkmax
```

DESCRIPTION:

Calculates the exchange term of the Bethe-Salpeter Hamiltonian.
REVISION HISTORY:

```
Forked from exccoulint.F90 and adapted for non-TDA and Q-dependent BSE. (Aurich)
```

1.1.17 pade (Source File: pade.F90)

INTERFACE:

```
Subroutine pade (m, z, n, iw, ih, h)
```

INPUT/OUTPUT PARAMETERS:

```
m     : number of values in array below (in,integer)
z     : values for which analytic continuation is to be performed
        (in,complex(m))
n     : number of values in arrays below (in,integer)
iw    : values for which function is evaluated
        (in,complex(n))
ih    : function evaluated at values "iw" (in,complex(n))
h     : function evaluated at values "z" (out,complex(n))
```

USES:

```
Use m_ctdfrac
```

DESCRIPTION:

Implementation of a Padé approximant using Thiele's reciprocal-difference method. This routine takes a complex function ($f_n = f(z_n)$) evaluated at an initial set of arguments,

(z_n) approximates the function with the help of Padé approximants, and evaluates (extrapolates/rotates) this approximation at a given set of arguments (z). The N-point Padé approximant then reads

$$P_N(z) = \cfrac{a_1}{1 + \cfrac{a_2(z - z_1)}{\cdots + \cfrac{a_n(z - z_{N-1})}{1 + (z - z_N)g_{N+1}(z)}}}$$

where

$$g_n(z) = \frac{g_{n-1}(z_{n-1}) - g_{n-1}(z)}{(z - z_{n-1})g_{n-1}(z)},\ n \geq 2$$

and

$$a_n = g_n(z_n),\ g_1(z_n) = f_n,\ n = 1, \ldots, N.$$

For simplicity, the expression $g_{N+1}(z)$ is set to zero in the continued fraction expression of the approximant. Expressions are taken from K. Lee and K. Chang, Phys. Rev. B 54, R8285 (1996). See also H. Vidberg and J. Serene, J. Low Temp. Phys., 29, 179 (1977).
REVISION HISTORY:

```
Created December 2006 (S. Sagmeister)
Documentation added, July 2011 (S. Sagmeister)
```

1.1.18 putbseinfo (Source File: m_putgetbsemat.f90)

INTERFACE:

```
subroutine putbseinfo(fname, iqmt)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
IN:
character(*) :: fname    ! Output file name
```

DESCRIPTION:

The routine write supporting information about a BSE calculation to file.
REVISION HISTORY:

```
Created. (Aurich)
```

1.1.19 getbseinfo (Source File: m_putgetbsemat.f90)

INTERFACE:

```
subroutine getbseinfo(fname, iqmt, fcmpt, fid)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
IN:
integer(4) :: iqmt ! index of momentum transfer vector
```

DESCRIPTION:

The routine reads supporting information about a saved BSE calculation. And compared it to the requested inforamtion.

REVISION HISTORY:

```
Created. (Aurich)
```

1.1.20 putbsereset (Source File: m_putgetbsemat.f90)

INTERFACE:

```
subroutine putbsereset
```

DESCRIPTION:

Resets module vars.

REVISION HISTORY:

```
Created. (Aurich)
```

1.1.21 putbsemat (Source File: m_putgetbsemat.f90)

INTERFACE:

```
subroutine putbsemat(fname, tag, ikkp, iqmt, zmat)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
IN:
character(*) :: fname    ! Output file name
integer(4) :: tag        ! MPI communication tag
integer(4) :: ikkp       ! Index of ik jk combination
integer(4) :: iqmt       ! Index of momentum transfer q
complex(8) :: zmat(:,:)  ! Complex 2d-array
```

DESCRIPTION:

The routine writes complex 2d-array to a direct access file and is intended for the use in be BSE part of the code. It is used to write the screened coulomb interaction `SCCLI.OUT` and the exchange interaction `EXCLI.OUT` to file.

REVISION HISTORY:

```
Forked from {\tt putbsemat}. (Aurich)
```

1.1.22 getbsemat (Source File: m_putgetbsemat.f90)

INTERFACE:

```
subroutine getbsemat(fname, iqmt, ikkp, zmat, check, fcmpt, fid)
```

DESCRIPTION:

This routine is used for reading the screened coulomb interaction and exchange interaction from file. It works for ou,ou combinations.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1.23 writeangmom (Source File: writeangmom.F90)

INTERFACE:

```
Subroutine writeangmom (un)
```

USES:

```
Use modinput
Use modmain
Use modxs
```

INPUT/OUTPUT PARAMETERS:

DESCRIPTION:

Outputs information about the angular momentum cutoffs to the file `INFOXS.OUT`.

REVISION HISTORY:

```
Created October 2006 (Sagmeister)
```

1.1.24 transijst (Source File: transijst.F90)

INTERFACE:

```
logical function transijst(ik,ist,jst)
```

USES:

```
use modinput
```

DESCRIPTION:

This function returns "true" if the transition specified by the input k-point initial and final state matches with the ones specified in the `transitions` element. Wether a state is included or excluded depends on the sequence of the transitions specified (using the "include" and "exclude" attribute).

REVISION HISTORY:

```
Created July 2010 (Sagmeister)
```

1.1.25 gradzfmtr (Source File: gradzfmtr.F90)

INTERFACE:

```
Subroutine gradzfmtr (lmax, nr, r, l1, m1, ld1, ld2, fmt, gfmt)
```

INPUT/OUTPUT PARAMETERS:

```
lmax  : maximum angular momentum (in,integer)
nr    : number of radial mesh points (in,integer)
r     : radial mesh (in,real(nr))
ld1   : leading dimension 1 (in,integer)
ld2   : leading dimension 2 (in,integer)
fmt   : real muffin-tin function (in,real(nr))
gfmt : gradient of zfmt (out,real(ld1,ld2,3))
```

DESCRIPTION:

Calculates the gradient of a muffin-tin function with real spherical harmonics expansion coefficients, $f(r)$, corresponding to a specific lm-combination. The gradient is given in a spherical harmonics representation. The y-component is divided by i to be expressed as a real number. See routine `gradzfmt`.

REVISION HISTORY:

```
Created April 2008 (Sagmeister)
```

1.1.26 getpmat (Source File: getpmat.F90)

INTERFACE:

```
subroutine getpmat(ik, vklt, i1, f1, i2, f2, tarec, filnam, pm)
```

USES:

```
use modmain
use modinput
use modxs
use modmpi
use m_getunit
```

INPUT/OUTPUT PARAMETERS:

```
In:
integer :: ik           ! k point index
integer :: i1,f1        ! Range of bands (final states)
integer :: i2,f2        ! Range of bands (initial states)
logical :: tarec        ! Use absolute ik as record index or
                        ! relative ik w.r.t k set of currrent MPI rank
character(*) :: filname ! File name of direct access momentum matrix file
Out:
complex(8) :: pm(3, i1:f1, i2:f2) ! The requested slice of the saved momentum
                                  ! matrix elements
```

DESCRIPTION:

Reads selectable portions of the momentum matrix form file.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1.27 findsymeqiv (Source File: findsymeqiv.F90)

INTERFACE:

```
subroutine findsymeqiv(tfbz, vpl, vplr, nsc, sc, ivgsc)
```

USES:

```
use modmpi, only: terminate
use modinput, only: input
use mod_lattice, only: bvec
use mod_symmetry, only: nsymcrys, lsplsymc, symlat, maxsymcrys
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  logical, tfbz    :  Use 1st Bz of [0,1) as unit cell
  real(8), vpl(3)  :  Lattice coordinates of non-reduced k point
  real(8), vplr(3) :  Lattice coordinates of reduced k point
OUT:
  integer(4), nsc  :  Number of symmetry operations that transform vplr into vpl
  integer(4), sc(maxsymcrys) : The first nsc entries contain the
                               corresponding crystal symmetry indices
  integer(4), ivgsc(3, maxsymcrys) : G vectors that shift the rotated vplr
                                     back to the unit cell
```

DESCRIPTION:

Given one non-reduced k-point vector and a reduces one, the routine determines the symmetry operations that rotate the reduced k point to the non-reduced one. Note: The routine terminates the program, if no symmetry operation is found.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich 2016)
Changed formatting and added comments. (Aurich 2016)
```

1.1.28 diagfull (Source File: m_diagfull.f90)

INTERFACE:

```
subroutine diagfull(n, ham, evalre, evalim, evecr, evecl, fbalance, frcond, fsort)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  integer(4) :: n
  complex(8) :: ham(n,n)          ! Complex square matrix
  logical, optional :: fbalance   ! Balance matrix
  logical, optional :: frcond     ! Calculate reciprocal conditioning numbers
  logical, optional :: fsort      ! Sort solutions by real part of eigenvalues
OUT:
  real(8) :: evalre(n)                        ! Real part of eigenvalues
  real(8), optional :: evalim(n)              ! Imaginary part of eigenvalues
  complex(8), target, optional :: evecr(n, n) ! Right eigenvectors as columns
  complex(8), target, optional :: evecl(n, n) ! Left eigenvectors as columns
```

DESCRIPTION:

This is a wrapper routine around the lapack routine `ZGEEVX` for the digitalization of a general complex double precision matrix. It returns real, and optionally imaginary, part of the eigenvalues and optionally right and/or left eigenvectors. There are flags for balancing the

matrix and calculating the reciprocal conditioning numbers of eigenvalues and eigenvectors. Note: If frcond = .ture. the left and right eigenvectors are still computed internally.
REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.1.29 fxc_lrc (Source File: fxc_lrc.F90)

INTERFACE:

```
Subroutine fxc_lrc (msiz, sw, alpha, fxc)
```

USES:

```
Use mod_constants, Only: fourpi
Use modxs, Only: unitout
```

INPUT/OUTPUT PARAMETERS:

```
msiz  : matrix size of local field effects (in,integer)
sw    : true for inclusion of local field effects (in,logical)
alpha : real constant (in,real)
fxc   : xc-kernel Fourier coefficients (out,complex(:,:))
```

DESCRIPTION:

Static long range xc-kernel; S. Botti, PRB 70, 045301 (2004). Calculates the symmetrized xc-kernel for the static long range model. According to the switch `sw` either

$$f_{\rm xc}(\mathbf{G},\mathbf{G}') = -\frac{\alpha}{4\pi}\delta(\mathbf{G},\mathbf{G}'),$$

if the switch is true, or

$$f_{\rm xc}(\mathbf{G},\mathbf{G}') = -\frac{\alpha}{4\pi}\delta(\mathbf{G},\mathbf{G}')\delta(\mathbf{G},\mathbf{0}),$$

otherwise.
REVISION HISTORY:

```
Created March 2006 (Sagmeister)
```

1.1.30 xszoutpr (Source File: xszoutpr.F90)

INTERFACE:

```
Subroutine xszoutpr (n1, n2, alpha, x, y, a)
```

INPUT/OUTPUT PARAMETERS:

```
n1,n2 : size of vectors and matrix, respectively (in,integer)
alpha : complex constant (in,complex)
x     : first input vector (in,complex(n1))
y     : second input vector (in,complex(n2))
a     : output matrix (out,complex(n1,n2))
```

DESCRIPTION:

Performs the rank-2 operation

$$A_{ij} \rightarrow \alpha \mathbf{x}_i^* \mathbf{y}_j + A_{ij}.$$

REVISION HISTORY:

```
Created March 2008 (Sagmeister)
```

1.1.31 getevecfv0 (Source File: m_getgrst.f90)

INTERFACE:

```
subroutine getevecfv0(vpl, vgpl, evecfvt)
```

USES:

```
use mod_kpoint, only: vkl_ptr
use mod_Gkvector, only: ngkmax_ptr, vgkl_ptr, ngk_ptr
use mod_eigensystem, only: nmatmax_ptr
use mod_misc, only: filext
use modxs, only: filext0
use mod_spin, only: nspnfv
use mod_eigenvalue_occupancy, only: nstfv
use mod_ematptr
use mod_constants, only: zzero
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  real(8) :: vpl(3)          ! k-point vector in lattice coordinates
  real(8) :: vgpl(3, ngkmax) ! G+k-vectors in lattice coordinates
OUT:
  complex(8) :: evecfvt(nmatmax, nstfv, nspnfv) ! Eigenvectors at that k-point
```

DESCRIPTION:

This routine is a wrapper for `getevecfv` that changes the k and $G + k$ quantities in `mod_kpoint` and `mod_Gkvector` to the corresponding quantities saved in `modxs` (nmatmax0, vkl0, ngk0, etc.), changes the file extension in `mod_misc` accordingly, reads in the Eigenvector and finally restores the original state.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1.32 getevecfv1 (Source File: m_getgrst.f90)

INTERFACE:

```
subroutine getevecfv1(vpl, vgpl, evecfvt)
```

USES:

```
use mod_kpoint, only: vkl_ptr
use mod_Gkvector, only: ngkmax_ptr, vgkl_ptr, ngk_ptr
use mod_eigensystem, only: nmatmax_ptr
use mod_spin, only: nspnfv
use mod_eigenvalue_occupancy, only: nstfv
use mod_ematptr
use mod_constants, only: zzero
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  real(8) :: vpl(3)          ! k-point vector in lattice coordinates
  real(8) :: vgpl(3, ngkmax) ! G+k-vectors in lattice coordinates
OUT:
  complex(8) :: evecfvt(nmatmax, nstfv, nspnfv) ! Eigenvectors at that k-point
```

DESCRIPTION:

This routine is a wrapper for `getevecfv` that changes the k and $G + k$ quantities in `mod_kpoint` and `mod_Gkvector` to the corresponding quantities saved in `modxs` (nmatmax0, vkl0, ngk0, etc.), changes the file extension in `mod_misc` accordingly, reads in the Eigenvector and finally restores the original state.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1.33 getevecfv0 (Source File: m_getgrst.f90)

INTERFACE:

```
subroutine getevecsv0(ik, evecsvt)
```

USES:

```
use mod_kpoint, only: vkl_ptr
use mod_Gkvector, only: ngkmax_ptr, vgkl_ptr, ngk_ptr
use mod_eigensystem, only: nmatmax_ptr
use mod_misc, only: filext
use modxs, only: filext0
use mod_eigenvalue_occupancy, only: nstsv
use mod_ematptr
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  integer :: ik        ! k-point index
OUT:
  complex(8) :: evecsvt(nstsv, nstsv) ! Eigenvectors at that k-point
```

DESCRIPTION:

This routine is a wrapper for `getevecfv` that changes the k and $G + k$ quantities in `mod_kpoint` and `mod_Gkvector` to the corresponding quantities saved in `modxs` (nmatmax0, vkl0, ngk0, etc.), changes the file extension in `mod_misc` accordingly, reads in the Eigenvector and finally restores the original state.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
Extended to 2nd variation. (Vorwerk)
```

1.1.34 getevecsv1 (Source File: m_getgrst.f90)

INTERFACE:

```
subroutine getevecsv1(ik,evecsvt)
```

USES:

```
use mod_kpoint, only: vkl_ptr
use mod_Gkvector, only: ngkmax_ptr, vgkl_ptr, ngk_ptr
use mod_eigensystem, only: nmatmax_ptr
use mod_eigenvalue_occupancy, only: nstsv
use mod_ematptr
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  integer :: ik          ! k-point index
OUT:
  complex(8) :: evecsvt(nstsv, nstsv) ! Eigenvectors at that k-point
```

DESCRIPTION:

This routine is a wrapper for `getevecfv` that changes the k and $G + k$ quantities in `mod_kpoint` and `mod_Gkvector` to the corresponding quantities saved in `modxs` (nmatmax0, vkl0, ngk0, etc.), changes the file extension in `mod_misc` accordingly, reads in the Eigenvector and finally restores the original state.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
Extended to 2nd variation (Vorwerk)
```

1.1.35 findocclims (Source File: findocclims.F90)

INTERFACE:

```
subroutine findocclims(iq, ikiq2ikp, iocc_common, iunocc_common, io0, io, iu0, iu)
```

USES:

```
use mod_constants, only: h2ev
use mod_kpoint, only: nkpt, vkl
use mod_eigenvalue_occupancy, only: nstsv, occsv, evalsv, occmax,&
                                  & efermi
use modinput, only: input
use modxs, only: evalsv0, occsv0, evlmin,&
               & evlmax, evlmincut, evlmaxcut, evlhpo,&
               & evllpu, ksgap, ksgapval, qgap, nstocc0,&
               & vkl0, nstunocc0, unitout
use m_genfilname
```

INPUT/OUTPUT PARAMETERS:

```
IN:
integer :: iq ! q-point index
integer :: ikiq2ikp(nkpt) ! Index mapping from (ik,iq)-->ikp
OUT:
integer :: iocc_common   ! Highest (partially) occupied state over all k-points and k+q-poi
integer :: iunocc_common ! Lowest (partially) unoccupied state over all k-points and k+q-po
integer :: io0(nkpt) ! Highest (partially) occupied state for each k-point
integer :: iu0(nkpt) ! Lowest (partially) unoccupied state for each k-point
integer :: io(nkpt), iu(nkpt) ! Same as above, but for k+q
INDIRECT OUT:
integer :: modxs:nstocc0 = iocc0
integer :: modxs:nstunocc0 = nstsv - iocc0
```

DESCRIPTION:

For a given q-point index the routine inspects the occupancies and second variational eigenvalues of the corresponding $k+q$ and k set to determin the highest/lowest (at least partially) occupied/unoccupied state index for all $k+q$ and k points and the highest/lowest occupied/unoccupied state index over all $k+q$ and k points. For the k set the files `EVECSV_QMT000.OUT` and `OCCSV_QMT000.OUT` is used, while for the k+q set the files `EVECSV_QMTXXX.OUT` and `OCCSV_QMTXXX.OUT` is used, where `XXX` is the XXX'th element of the q-point list specified in the input file. For the input of iq = 0 the k and $k+q$ sets are identical and the `_QMT000` files are used.

REVISION HISTORY:

```
Added description schema. And rudimentary description. (Aurich)
Added some more description. (Aurich)
Added calculation of minimal (indirect) gap. (Aurich)
Added calculation of minimal gap for iq. (Aurich)
Change input output. (Aurich)
```

1.1.36 sorteval (Source File: sorteval.F90)

INTERFACE:

```
Module m_sorteval
      Implicit None
Contains
Subroutine sorteval (vpl, ra, ca)
```

USES:

```
      Use modmain
```

DESCRIPTION:

Rearrange real array `ra` or complex array `ca` through the same permutations that would sort the energy eigenvalues into ascending order for a given k point `vpl`. No sorting of the eigenvalues itself is performed. Uses the sortidx subroutine which in turn uses the Heapsort algorithm. **REVISION HISTORY:**

```
Created Jul 2013 SR
```

1.1.37 getdocc (Source File: getdocc.F90)

INTERFACE:

```
subroutine getdocc(iq, ik, ikq, l1, u1, l2, u2, docc)
```

USES:

```
use mod_eigenvalue_occupancy, only: nstsv
use mod_kpoint, only: vkl
use modxs, only: vkl0
```

DESCRIPTION:

Calculates occupation number differences for $f_{o_l k_i} - f_{u_m k_j}$. Currently the `iq` input argument does nothing. WARNING: `xssave0` has to be called in advance.
REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1.38 fxc_spk (Source File: fxc_spk.F90)

INTERFACE:

```
Subroutine fxc_spk (fxctype,msiz,ngtot,nw, chim, chim_w, fxc, fxc_w)
```

USES:

```
use modmpi
Use mod_constants, Only: fourpi, zzero, zone
Use modxs, Only: unitout
Use invert
Use m_dyson
Use modinput
```

INPUT/OUTPUT PARAMETERS:

```
msiz  : matrix size of local field effects (in,integer)
chim : model static density response function (e.g., scissors corrected) (in,complex(:,:)
fxc   : xc-kernel  (out,complex(:,:))
```

DESCRIPTION:

Static spk xc-kernel; PRL 107, 186401 (2011), PRL 114, 146402 (2015) Calculates the symmetrized BO and RBO xc-kernels.

REVISION HISTORY:

```
Created October 2015 (SR)
```

1.1.39 genpwmat (Source File: genpwmat.F90)

INTERFACE:

```
Subroutine genpwmat (vpl, ngpmax, ngp, vgpc, gpc, igpig, ylmgp, sfacgp, &
& vklk, ngkk, igkigk, apwalmk, evecfvk, evecsvk, vklkp, ngkkp, igkigkp, &
& apwalmkp, evecfvkp, evecsvkp, pwmat)
```

USES:

```
Use modmain
Use modxs
Use modinput
```

INPUT/OUTPUT PARAMETERS:

DESCRIPTION:

Calculates the matrix elements of the plane wave

$$M_{ij\mathbf{k}} = \langle \Psi_{i,\mathbf{k}} | e^{-i(\mathbf{G}+\mathbf{q})\mathbf{r}} | \Psi_{j,\mathbf{k}} \rangle.$$

Straightforward implementation for checking.

REVISION HISTORY:

```
Created November 2007 (Sagmeister)
```

1.1.40 fxc_bse_ma03 (Source File: fxc_bse_ma03.F90)

INTERFACE:

```
Subroutine fxc_bse_ma03 (msiz, oct, sw, iw, fxc)
```

USES:

```
Use modinput
Use mod_constants, Only: zzero
Use modmpi, Only:
Use modxs, Only: unitout, bzsampl
Use invert
Use m_xsgauntgen
Use m_findgntn0
Use m_writegqpts
Use m_genfilname
Use m_getunit
```

INPUT/OUTPUT PARAMETERS:

```
msiz  : matrix size of local field effects (in,integer)
sw    : true for inclusion of local field effects (in,logical)
alpha : real constant (in,real)
fxc   : xc-kernel Fourier coefficients (out,complex(:,:))
```

DESCRIPTION:

BSE-kernel of A. Marini, Phys. Rev. Lett. 91, 256402 (2003). Interface function.

REVISION HISTORY:

```
Created March 2008 (Sagmeister)
```

1.1.41 wavefmt_lo (Source File: wavefmt_lo.F90)

INTERFACE:

```
Subroutine wavefmt_lo (lrstp, lmax, is, ia, ngp, evecfv, ld, wfmt)
```

USES:

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
lrstp  : radial step length (in,integer)
lmax   : maximum angular momentum required (in,integer)
is     : species number (in,integer)
ia     : atom number (in,integer)
```

```
ngp    : number of G+p-vectors (in,integer)
evecfv : first-variational eigenvector (in,complex(nmatmax))
ld     : leading dimension (in,integer)
wfmt   : muffin-tin wavefunction (out,complex(ld,*))
```

DESCRIPTION:

Muffin-tin wavefunction built up by local orbital contribution only. Based upon the routine `wavefmt`.

REVISION HISTORY:

```
Created May 2008 (Sagmeister)
```

1.1.42 screenlauncher (Source File: screenlauncher.f90)

INTERFACE:

```
subroutine screenlauncher
```

USES:

```
use modmpi
use modinput, only: input
use mod_APW_LO, only: lolmax
use mod_kpoint, only: nkpt
use mod_qpoint, only: nqpt
use modxs, only: xsgnt, nwdf, qpari,&
                 & qparf, unitout, totalqlmt, qvkloff,&
                 & gqdirname, eps0dirname, scrdirname, timingdirname,&
                 & ikmapikq, nkpt0, vkl0, usefilext0, filext0, filexteps,&
                 & iqmt0, iqmt1, evalsv0, istocc0, istunocc0, isto0, isto,&
                 & istu0, istu, nst1, nst2, ngq
use mod_xsgrids
use m_genfilname
use m_filedel
use m_writegqpts
use m_xsgauntgen
use m_findgntn0
use mod_Gkvector, only: gkmax
use m_ematqk
```

DESCRIPTION:

This is a wrapper routine for the call of `dfq.f90` in the screen task of the BSE calculation. It stats the calculation of the Kohn-Sham dielectric matrix (in RPA) for the q-points formed from the k-point differences $\vec{q} = \vec{k}' - \vec{k}$. If the BSE coupling blocks are included, the dielectric matrix is also calculated for a shifted set of q-point which are formed form $q = -\vec{k}' - \vec{k}$.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1.43 rewritesorted (Source File: rewritesorted.F90)

INTERFACE:

```
Subroutine rewritesorted
```

USES:

```
Use m_sorteval
Use modmain
Use modxs
```

DESCRIPTION:

In a collinear spin calculation, the second variational hamiltonian is block diagonalized in subroutine seceqnsv. This leads to block-sorted eigenvalues, eigenvectors and occupations files into increasing energy for each spin chanel. The XS part needs fully sorted arrays intro incresing energy in order to correctly find bandlimits. Accordingly, this subroutine rewrites the corresponding files sorted into increasing energy when the calculation is collinear.

REVISION HISTORY:

```
Created July 2013 SR
```

1.1.44 zfinp2 (Source File: zfinp2.F90)

INTERFACE:

```
Complex (8) Function zfinp2 (ngp1, ngp2, igpig, zfmt1, zfmt2, zfir1, &
& zfir2)
```

USES:

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
zfmt1 : first complex function in spherical harmonics for all muffin-tins
        (in,complex(lmmaxvr,nrcmtmax,natmtot))
zfmt2 : second complex function in spherical harmonics for all muffin-tins
        (in,complex(lmmaxvr,nrcmtmax,natmtot))
zfir1 : first complex interstitial function in real-space
        (in,complex(ngrtot))
zfir2 : second complex interstitial function in real-space
        (in,complex(ngrtot))
```

DESCRIPTION:

Calculates the inner product of two complex fuctions over the entire unit cell. The muffin-tin functions should be stored on the coarse radial grid and have angular momentum cut-off `lmaxvr`. In the intersitial region, the integrand is multiplied with the smooth characteristic function, $\tilde{\Theta}(\mathbf{r})$, to remove the contribution from the muffin-tin. See routines `zfmtinp` and `gencfun`. Based upon the routine `zfinp`.

REVISION HISTORY:

```
Created January 2007 (Sagmeister)
```

1.1.45 bselauncher (Source File: bselauncher.f90)

INTERFACE:

```
subroutine bselauncher
```

USES:

```
use modmpi
use modscl
use modxs, only: unitout
use modinput, only: input
```

DESCRIPTION:

Launches the construction and solving of the Bethe-Salpeter Hamiltonian for the specified $\vec{Q}_{\mathrm{mt}}$ momentum transfer and approximation (TDA or non-TDA).

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.1.46 writeexcevec (Source File: writeexcevec.f90)

INTERFACE:

```
subroutine writeexcevec()
```

USES:

```
use modxs, only: unitout
use modinput
use modmpi
use mod_constants, only: h2ev
use m_getunit
use m_genfilname
use m_putgetexcitons
```

DESCRIPTION:

Reads the binary file `EXCCOEFF` containing the eigenvector coefficients of the BSE calculation and prints selected coefficients to human readable files. Writes to the folders `EXCITON_EVEC`, `BEVEC` and `BEVEC_KSUM`.

REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.1.47 genparidxran (Source File: genparidxran.F90)

INTERFACE:

```
subroutine genparidxran(typ, n)
```

USES:

```
use modmpi
use mod_qpoint, only: nqpt
use mod_kpoint, only: nkpt
use modxs, only: wpari, wparf, qpari, qparf,&
               & kpari, kparf, nwdf, ppari,&
               & pparf, partype, unitout
!INPUT/OUTPUT PARAMETER:
  IN:
  character(1) :: typ  ! Physical meaning of index
  integer(4) :: n      ! Number of elements to distribute
  Module IN:
  integer(4) :: nwdf   ! Number of frequencies for the construction of
                       ! the dielectric matrix in RPA
  integer(4) :: nkpt   ! Number of k-points
  integer(4) :: nqpt   ! Number of q-points
```

DESCRIPTION:

The routine sets loop indices for each calling `MPI` rank for k-points, q-points, p-point and ω-points in modxs.

REVISION HISTORY:

```
Added to documentation scheme. 2016 (Aurich)
Changed to also work for the case that there
are more processes than elements. (Aurich)
```

1.1.48 kkpmap (Source File: kkpmap.F90)

INTERFACE:

```
subroutine kkpmap(ikkp, nkp, ik, ikp)
```

INPUT/OUTPUT PARAMETERS:

```
IN
  ikkp, integer: Index of unique k-point combination
  nkp, integer:  Number of k-points
OUT
  ik, integer:  Index of first k-point
  ikp, integer: Index of second k-point
```

DESCRIPTION:

Calculates individual k-point indices for a combined k-k' index.
Example: nkp=3
Number of unique combinations 6
ikkp ik ikp
1 1 1
2 1 2
3 1 3
4 2 2
5 2 3
6 3 3
REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1.49 kkpmap_back (Source File: kkpmap.F90)

INTERFACE:

```
subroutine kkpmap_back(ikkp, nkp, ik, ikp)
```

INPUT/OUTPUT PARAMETERS:

```
IN
  nkp, integer:  Number of k-points
  ik, integer:  Index of first k-point
  ikp, integer: Index of second k-point
OUT
  ikkp, integer: Index of unique k-point combination
```

DESCRIPTION:

Calculates the combined index given to k indices with ikp¿=ik. Example: nkp=3
Number of unique combinations 6
ikkp ik ikp
1 1 1
2 1 2
3 1 3
4 2 2
5 2 3
6 3 3
REVISION HISTORY:

```
Created as complement to kkpmap. (Aurich)
```

1.1.50 rotematrad (Source File: rotematrad.F90)

INTERFACE:

```
subroutine rotematrad(ngp, igpmap)
```

USES:

```
use modxs, only: riaa, riloa, rilolo
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  integer(4), ngp : number of G+q vectors
  integer(4), igpmap(ngp) : {G+q} --> {G+q_r} index map
```

DESCRIPTION:

The routine applies the $\left\{\vec{G}+\vec{q}\right\}$ index map created in `findgqmap` to the radial integral arrays computed for the reduced q set.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich 2016)
```

1.1.51 scrcoulint (Source File: scrcoulint.f90)

INTERFACE:

```
subroutine scrcoulint(iqmt, fra)
```

USES:

```
use mod_misc, only: filext
use modinput, only: input
use modmpi
use mod_constants, only: zzero, zone, fourpi
use mod_APW_LO, only: lolmax
use mod_qpoint, only: iqmap, vql, vqc, nqpt, ivq, wqpt
use mod_lattice, only: omega
use mod_symmetry, only: maxsymcrys
use modxs, only: xsgnt, unitout,&
              & ngqmax,&
              & nqptr, qpari, qparf, ivqr,&
              & ngq, ppari, pparf, iqmapr,&
              & vqlr, vqcr, wqptr, ngqr,&
              & bcbs, ematraddir, eps0dirname,&
              & filext0, usefilext0, iqmt0, iqmt1,&
              & iqmtgamma,&
              & bsedl, bsedu, bsedd, bsed
use m_xsgauntgen
```

```
use m_findgntn0
use m_writevars
use m_genfilname
use m_getunit
use m_ematqk
use m_putgetbsemat
use modbse
use mod_xsgrids
use mod_Gkvector, only: gkmax
```

DESCRIPTION:

Calculates the resonant-resonant or resonant-anit-resonant block of the direct term of the Bethe-Salpeter Hamiltonian for a momentum transfer $\vec{Q}_{mt}$.
REVISION HISTORY:

```
Forked from scrcoulint.F90 and adapted for non-TDA BSE and finite Q. (Aurich)
```

1.1.52 genkcpts (Source File: genksubpts.F90)

Subroutine genksubpts () **USES:**

```
Use modmain
Use modxs
Use modinput
Use m_getunit
Use m_genfilname
Use modmpi
```

DESCRIPTION:

Sets up a double k-grid for BSE calculations.
REVISION HISTORY:

```
Created January 2014, S. Kontur
```

1.1.53 dhesolver (Source File: m_dhesolver.f90)

INTERFACE:

```
subroutine dhesolver(ham, eval, binfo, evec, i1, i2, v1, v2, found, eecs)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  type(blacsinfo) :: binfo       ! Info type describing the BLACS grid
  integer(4), optional :: i1, i2 ! Index range of eigen-solutions
  real(8), optional :: v1, v2    ! Range of eigenvalues to search for
```

```
  integer(4), optional :: eecs   ! Estimate for eigenvalue clustering.
                                 ! Apriori not known but needed for propper
                                 ! orthogonalization of eigenvectors (default = 3)
IN/OUT:
  type(dzmat) :: ham         ! 2D block cyclic distributed hermitian matrix
  real(8) :: eval(ham%nrows) ! Real valued eigenvalues in ascending order
  type(dzmat), optional :: evec   ! 2D block cyclic distributed eigenvector matrix
OUT:
  integer(4), optional :: found ! How many solutions were found
```

DESCRIPTION:

Takes the upper triangular part of an distributed complex matrix matrix, assumed to be hermitian and finds eigenvalues and eigenvectors using the scalapack routine `pzheevx`.

REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.1.54 genwgrid (Source File: genwgrid.F90)

INTERFACE:

```
subroutine genwgrid(n, intv, timag, brd, w_real, w_cmplx)
```

INPUT/OUTPUT PARAMETERS:

```
In:
integer :: n       ! Number of grid points
real(8) :: intv(2) ! Intervall limits
logical, optional :: timag ! Flag for imaginary fequencies
real(8), optional :: brd   ! Broadening parameter
Out:
real(8), optional :: w_real(:)     ! Real valued grid
complex(8), optional :: w_cmplx(:) ! Complex valued grid
```

DESCRIPTION:

Generates an evenly spaced "frequency" grid. The gird is either reals OR complex valued. If timag=true then all frequencies are multiplied by i, otherwise the complex grid is mathematically identical to the real valued one. If brd is non zero the frequencies of the complex grid are shifted by i*brd.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1.55 genpmatcorxs (Source File: genpmatcorxs.f90)

INTERFACE:

```
subroutine genpmatcorxs(ik,ngp,apwalm,evecfv,evecsv,pmatc)
```

USES:

```
    use modinput
    use mod_constants, only: zzero, zi
    use mod_kpoint, only: vkl
    use mod_atoms, only: idxas, natmtot
    use mod_muffin_tin, only: nrcmtmax, nrcmt, rcmt, nrmtmax, nrmt, &
      & idxlm, lmmaxapw
    use mod_atoms, only: spl, spk, spr
    use mod_spin, only: nspinor
    use mod_Gkvector, only: ngkmax
    use mod_APW_LO, only: apwordmax
    use mod_eigensystem, only: nmatmax
    use mod_eigenvalue_occupancy, only: nstfv, nstsv
    use m_getgrst, only: wavefmtsv
 use modmain
    use modxas
```

INPUT/OUTPUT PARAMETERS:

```
    ik     : k-point position (in,integer)
    ngp    : number of G+p-vectors (in,integer)
    apwalm : APW matching coefficients
             (in,complex(ngkmax,apwordmax,lmmaxapw,natmtot))
    evecfv : first-variational eigenvector (in,complex(nmatmax,nstfv))
    evecsv : second-variational eigenvectors (in,complex(nstsv,nstsv))
    pmatc   : momentum matrix elements (out,complex(3,ncg,nstsv))
```

DESCRIPTION:

Calculates the momentum matrix elements between a core state and a conduction state.

REVISION HISTORY:

```
    Created August 2006 (RGA)
    Revisited June 2011 (DIN)
  Adjusted to core states November 2015 (Christian Vorwerk)
```

1.1.56 wavefmt_apw (Source File: wavefmt_apw.F90)

INTERFACE:

```
Subroutine wavefmt_apw (lrstp, lmax, is, ia, ngp, apwalm, evecfv, ld, &
& wfmt)
```

USES:

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
lrstp  : radial step length (in,integer)
lmax   : maximum angular momentum required (in,integer)
is     : species number (in,integer)
ia     : atom number (in,integer)
ngp    : number of G+p-vectors (in,integer)
apwalm : APW matching coefficients
         (in,complex(ngkmax,apwordmax,lmmaxapw,natmtot))
evecfv : first-variational eigenvector (in,complex(nmatmax))
ld     : leading dimension (in,integer)
wfmt   : muffin-tin wavefunction (out,complex(ld,*))
```

DESCRIPTION:

Muffin-tin wavefunction built up by APW contribution only. Based upon the routine `wavefmt`.

REVISION HISTORY:

```
Created May 2008 (Sagmeister)
```

1.1.57 xsgeneigvec (Source File: xsgeneigvec.f90)

INTERFACE:

```
subroutine xsgeneigvec(qi, qf, nqpts, vql, qvkloff, tscr, tmqmt)
```

USES:

```
use modmpi
use modinput, only: input
use modxs, only: unitout, vqlmt, vqcmt
use mod_misc, only: filext
use m_filedel, only: filedel
use m_genfilname, only: genfilname
```

INPUT/OUTPUT PARAMETERS:

```
In:
  integer(4) :: qi, qf ! Range qi:qf of then nqpts momentum transfer Qs form Q-point list
  integer(4) :: nqpts  ! for which to do one-shot GS runs
  real(8)    :: vql(3,nqpts)     ! Q-point vectors form list
  real(8)    :: qvkloff(3,nqpts) ! Offsets of corresponding k-grids
  logical    :: tscr  ! If true use screening file extension
  logical    :: tmqmt ! If true use file extension indicating that -Q/2 was used
```

DESCRIPTION:

The routine generates the one-shot GS quantities used in the BSE. The routine writes the files `APWCMT_*.OUT`, `EIGVAL_*.OUT`, `EVALSV_*.OUT`, `EVECFV_*.OUT`, `EVECSV_*.OUT`, `BONDLENGTH_*.OUT`, `EFERMI_*.OUT`, `LOCMT_*.OUT`, `OCCSV_*.OUT`, `geometry_*.xml`. The file extension contains the number of the considered Q-point `_QMTxyz`, the information whether the +Q/2 or -Q/2 shift was used `_m`. If the screening parameter were used the extension `_SCR` is added.

REVISION HISTORY:

```
Based on xsgeneigvec
Created. 2016 (Aurich)
```

1.1.58 bsesoldiag (Source File: bsesoldiag.F90)

INTERFACE:

```
subroutine bsesoldiag(solsize, hamsize, ham, eval, evec)
```

USES:

```
use modmpi
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  integer :: solsize ! Number of solutions from lowest EV
  integer :: hamsize ! Dimension of the hermitian matrix
  complex(8) :: ham(hamsize,hamsize) ! Upper triangular part of an hermitian matrix
OUT:
  real(8) :: eval(hamsize) ! Real valued eigenvalues in ascending order
                           ! (the first solsize elements are set)
  complex(8) :: evec(hamsize, hamsize) ! Corresponding eigenvectors as columns
```

DESCRIPTION:

Takes a upper triangular part of an hermitian matrix and finds eigenvalues and eigenvectors using the lapack routine `zheevr`.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1.59 zinvert (Source File: m_invertzmat.f90)

INTERFACE:

```
subroutine zinvert(zmat)
```

INPUT/OUTPUT PARAMETERS:

```
IN/OUT:
  complex(8) :: zmat  ! On entry: Matrix.
                      ! On exit : Inverted matrix.
```

DESCRIPTION:

Inverts complex matrix.
REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.1.60 dzinvert (Source File: m_invertzmat.f90)

INTERFACE:

```
subroutine dzinvert(zmat)
```

INPUT/OUTPUT PARAMETERS:

```
IN/OUT:
  type(dzmat) :: zmat ! On entry: Distributed matrix to invert.
                      ! On exit : Inverted matrix.
```

DESCRIPTION:

Takes a distributed general complex matrix and inverts it.
REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.1.61 kernxc_bse (Source File: kernxc_bse.F90)

INTERFACE:

```
subroutine kernxc_bse
```

USES:

```
use modmpi
use modinput
use mod_APW_LO, only: lolmax
use mod_qpoint, only: nqpt
use mod_kpoint, only: nkptnr
use mod_misc, only: task
use mod_lattice, only: omega
```

```
  use mod_constants, only: zi, zone, zzero
#ifdef TETRA
  use modtetra
#endif
  use modxs, only: xsgnt, unitout, ikmapikq,&
    & istocc0, istunocc0, istocc, istunocc,&
    & isto0, istu0, isto, istu,&
    & nst1, nst2, nst3, nst4,&
    & istl1, istl2, istl3, istl4,&
    & istu1, istu2, istu3, istu4,&
    & ksgap, wpari, wparf, nwdf,&
    & bzsampl, ngq, xiou, xiuo,&
    & pmou, pmuo, deou, deuo,&
    & docc12, docc21, bsed, torfxc,&
    & iqmt1, iqmt0, iqmtgamma, bcbs,&
    & sta1, sto1, sta2, sto2,&
    & usefilext0, filext0
  use m_xsgauntgen
  use m_findgntn0
  use m_writegqpts
  use m_genwgrid
  use m_xszoutpr3
  use m_getpemat
  use m_getunit
  use m_genfilname
  use m_putgetbsemat
  use m_ematqk
  use modbse
```

INPUT/OUTPUT PARAMETERS:

```
    oct   : optical diagonal tensor component (in,integer)
```

DESCRIPTION:

BSE-kernel of A. Marini, Phys. Rev. Lett. 91, 256402 (2003)

REVISION HISTORY:

```
    Created March 2008 (Sagmeister)
```

1.1.62 xszoutpr2 (Source File: xszoutpr2.F90)

INTERFACE:

```
      Subroutine xszoutpr2 (n1, n2, alpha, x, y, a)
```

INPUT/OUTPUT PARAMETERS:

```
n1,n2 : size of vectors and matrix, respectively (in,integer)
alpha : complex constant (in,complex)
x     : first input vector (in,complex(n1))
y     : second input vector (in,complex(n2))
a     : output matrix (out,complex(n1,n2))
```

DESCRIPTION:

Performs the rank-2 operation

$$A_{ij} \rightarrow \alpha \mathbf{x}_i \mathbf{y}_j^* + A_{ij}.$$

REVISION HISTORY:

```
Created March 2008 (Sagmeister)
```

1.1.63 xszoutpr3 (Source File: xszoutpr3.F90)

INTERFACE:

```
Subroutine xszoutpr3 (n1, n2, alpha, x, y, a)
```

INPUT/OUTPUT PARAMETERS:

```
n1,n2 : size of vectors and matrix, respectively (in,integer)
alpha : complex constant (in,complex)
x     : first input vector (in,complex(n1))
y     : second input vector (in,complex(n2))
a     : output matrix (out,complex(n1,n2))
```

DESCRIPTION:

Performs the rank-2 operation

$$A_{ij} \rightarrow \alpha \mathbf{x}_i \mathbf{y}_j + A_{ij}.$$

REVISION HISTORY:

```
Created March 2008 (Sagmeister)
```

1.1.64 putx0 (Source File: putx0.F90)

INTERFACE:

```
subroutine putx0(tp0, iq, iw, filnam, filxt, ch0, ch0wg, ch0hd)
```

USES:

```
use modmpi
use modmain
use modxs
use m_getunit
```

INPUT/OUTPUT PARAMETERS:

```
In:
logical :: tp0 ! Flag if iq is the q=0 q-point
integer :: iq  ! q-point index
integer :: iw  ! iw'th freqency
character(*) :: filnam, filxt ! File name and file extension
complex(8) :: ch0(:,:)     ! Body of RPA density-density response matrix
complex(8) :: ch0wg(:,:,:) ! Wings of RPA density-density response matrix
complex(8) :: ch0hd(:,:)   ! Body of RPA density-density response matrix
```

DESCRIPTION:

Writes the V-symmetrized Kohn-Sham response function $\tilde{\chi}^0_{\mathbf{GG}'}(\mathbf{q},\omega)$ for one q-point and multiple frequencies into a direct access file. The records are numbered by the frequency index.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

1.1.65 df (Source File: df.F90)

INTERFACE:

```
subroutine df
```

USES:

```
use modinput, only: input
use mod_APW_LO, only: lolmax
use mod_qpoint, only: nqpt
use modxs, only: tscreen, xsgnt, nwdf, qpari,&
                 & qparf, unitout
use modmpi
use m_writegqpts
use m_xsgauntgen
use m_findgntn0
use m_genfilname
```

DESCRIPTION:

Control routine for setting up the Kohn-Sham response function or the microscopic dielectric function/matrix for all specified **q**-points. Can be run with MPI parallelization for **q**-points.

REVISION HISTORY:

```
Created March 2006 (Sagmeister)
```

1.2 Fortran: Module Interface modxas (Source File: modxas.f90)

Contains additional global variables required for XAS calculations within the BSE EXCITING code.

REVISION HISTORY:

```
Created JUNE 2015 by Christian Vorwerk
```

1.2.1 hesolver (Source File: m_hesolver.f90)

INTERFACE:

```
subroutine hesolver(hemat, eval, evec, i1, i2, v1, v2, found)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  complex(8) :: hemat(:,:)       ! Upper triangular part of a hermitian matrix
  integer(4), optional :: i1, i2 ! Index range of eigen solutions
  real(8), optional :: v1, v2    ! Range of eigenvalues to search for
OUT:
  real(8) :: eval(:)                ! Real valued eigenvalues in ascending order
  complex(8), optional :: evec(:,:) ! Corresponding eigenvectors as columns
  integer(4), optional :: found     ! Number of found eigen-solutions
```

DESCRIPTION:

Wrapper for Lapack's zheevr routine. Takes a upper triangular part of an hermitian matrix and finds eigenvalues and/or eigenvectors using the lapack routine `zheevr`.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.2.2 bsegenspec (Source File: bsegenspec.f90)

INTERFACE:

```
subroutine bsegenspec()
```

USES:

```
use modmpi
use modinput
use m_readoscillator
use modbse, only: nk_bse
use m_makespectrum
use mod_constants, only: zzero, h2ev
use mod_kpoint, only: nkptnr
use mod_lattice, only: omega
use m_genwgrid
```

DESCRIPTION:

Collects BSE results from files EXCITON*.OUT and computes spectra anew.
REVISION HISTORY:

```
Created March, 2017, BA
```

1.2.3 fxc_lrcd (Source File: fxc_lrcd.F90)

INTERFACE:

```
Subroutine fxc_lrcd (msiz, sw, alpha, beta, w, fxc)
```

USES:

```
Use mod_constants, Only: fourpi
Use modxs, Only: unitout
```

INPUT/OUTPUT PARAMETERS:

```
msiz  : matrix size of local field effects (in,integer)
sw    : true for inclusion of local field effects (in,logical)
alpha : real constant (in,real)
w     : frequency grid (in,complex(:))
fxc   : xc-kernel Fourier coefficients (out,complex(:,:))
```

DESCRIPTION:

Dynamical long range xc-kernel; S. Botti, PRB 72, 125203 (2005). Calculates the symmetrized xc-kernel for the static long range model. According to the switch `sw` either

$$f_{\rm xc}(\mathbf{G},\mathbf{G}') = -\frac{1}{4\pi}(\alpha+\beta\omega^2)\delta(\mathbf{G},\mathbf{G}'),$$

if the switch is true, or

$$f_{\rm xc}(\mathbf{G},\mathbf{G}') = -\frac{1}{4\pi}(\alpha+\beta\omega^2)\delta(\mathbf{G},\mathbf{G}')\delta(\mathbf{G},\mathbf{0}),$$

otherwise.
REVISION HISTORY:

```
Created March 2006 (Sagmeister)
```

1.3 Fortran: Module Interface modbse (Source File: modbse.f90)

Supporting global variables and routines for the BSE scope.
REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.3.1 setranges_modxs (Source File: modbse.f90)

INTERFACE:

```
subroutine setranges_modxs(iqmt)
  use modinput
  use mod_xsgrids
  use mod_Gkvector, only: gkmax
  use modxs, only: totalqlmt, evalsv0, usefilext0, filext0,&
                & ksgap, ksgapval, qmtpgap, qmtmgap, unitout
  use m_genfilname
```

INPUT/OUTPUT PARAMETERS:

```
In:
integer(4) :: iqmt ! Considered q-point index (must be on the not shifted k-grid)
```

DESCRIPTION:

A small wrapper for the routine `findocclims`. Used to initialize modxs module variables for occupation limits for $\vec{k}$ and $\vec{k}+\vec{q}$ in some BSE related routines.

REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.3.2 select_transitions (Source File: modbse.f90)

INTERFACE:

```
subroutine select_transitions(iqmt, serial, dirname)
  use mod_kpoint, only: vkl
  use modxs, only: usefilext0, filext0, vkl0
  use modxas, only: xasstart, xasstop, ecore
  use m_genfilname
  use mod_symmetry, only: nsymcrys
```

INPUT/OUTPUT PARAMETERS:

```
In:
integer(4) :: iqmt  ! q-point index (on unshifted k-mesh)
Module out:
integer(4) :: hamsize            ! Dimension of the BSE Hamiltonian matrix
real(8)    :: ofac(hamsize)      ! Occupation factors need for
                                 ! the construction of the BSE matrix
integer(4) :: smap(hamsize, 3)   ! Map between BSE matrix index and u,o,k indices
integer(4) :: kousize(nk)        ! How many u o combinations allowed for each k
```

DESCRIPTION:

Given a selected energy range for the spectrum, this routine will select relevant transitions for each k point. Appart from the KS transition energies the routine checks whether the transition contain problematic occupancy differences and sorts them out if need be. The simple treatment of fractional occupancy does not allow for transitions between states of the same partial occupancy. Also cases of occupancy inversion where the occupancy difference is negative are filtered out, since those break any kind of hermiticity of the BSE Hamiltonian. The routine crates the compined index map $\alpha \leftrightarrow \{u_\alpha, o_\alpha, \vec{k}_\alpha\}$, auxilliary maps and determins the size of the resulting hamiltonian. In all cases is u the fastest index followed by o and k.
REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.3.3 hamidx (Source File: modbse.f90)

INTERFACE:

```
integer(4) function hamidx(i1, i2, ik, n1, n2)
```

INPUT/OUTPUT PARAMETERS:

```
In:
integer(4) :: i1, i2, ik ! Indices counting from 1 continuously
integer(4) :: n1, n2     ! Maximum values of i1, i2 respectively
Out:
integer(4) :: hamidx     ! Combined index
```

DESCRIPTION:

The function returns a combined index given two band indices and a $\vec{k}$ index. It is use in the construction of the BSE Hamiltonian.
Map:
hamdix $= i_1 + (i_2 - 1)n_1 + n_1 n_2 (i_k - 1)$
Notes:
i_1 is the fastest varying index, followed in order by i_2 and i_k.
All indices are assumed to be counted from 1 onwards continuously.
REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
Changed fastes index to i1. (Aurich)
```

1.3.4 hamidx_back (Source File: modbse.f90)

INTERFACE:

```
subroutine hamidx_back(s, i1, i2, ik, n1, n2)
```

INPUT/OUTPUT PARAMETERS:

```
In:
integer(4) :: s          ! Combined index created with {\tt hamidx}
integer(4) :: n1, n2     ! Maximum values of i1, i2 respectively
Out:
integer(4) :: i1, i2, ik ! Individual indices
```

DESCRIPTION:

The subroutine does the inverse operation of the function `hamidx`.
REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.3.5 subhamidx (Source File: modbse.f90)

INTERFACE:

```
integer(4) function subhamidx(i1, i2, n1)
```

INPUT/OUTPUT PARAMETERS:

```
In:
integer(4) :: i1, i2     ! Indices counting from 1 continuously
integer(4) :: n1         ! Maximum value of i1
Out:
integer(4) :: subhamidx  ! Combined index
```

DESCRIPTION:

The function return a combined index given two indices. It is use in the construction of the BSE Hamiltonian.
Map:
hamdix $= i_1 + (i_2 - 1)n_1$
Notes:
i_1 is the fastest varying index, followed by i_2.
All indices are assumed to be counted from 1 onwards continuously.
REVISION HISTORY:

```
Created 2016 (Aurich)
Changed fastest index to i1. (Aurich)
```

1.3.6 subhamidx_back (Source File: modbse.f90)

INTERFACE:

```
subroutine subhamidx_back(s, i1, i2, n1)
```

INPUT/OUTPUT PARAMETERS:

```
In:
integer(4) :: s    ! Combined index
integer(4) :: n1   ! Maximum value of i1
Out:
integer(4) :: i1, i2   ! Individual indices
```

DESCRIPTION:

The routine performs the inverse operation to `subhamidx`.

REVISION HISTORY:

```
Created 2016 (Aurich)
Changed fastest index to i1. (Aurich)
```

1.3.7 writesymi (Source File: writesymi.F90)

INTERFACE:

```
Subroutine writesymi
```

USES:

```
Use modinput
Use modmain
Use modxs
```

DESCRIPTION:

Outputs the crystal and symmetries including their inverse elements to file `SYMINV.OUT`

REVISION HISTORY:

```
Created December 2007 (Sagmeister)
```

1.3.8 ctdfrac (Source File: ctdfrac.F90)

INTERFACE:

```
Subroutine ctdfrac (n, a, b, f)
```

INPUT/OUTPUT PARAMETERS:

```
n     : depth of continued fraction (in,integer)
a     : a-coefficients (in,complex(n))
b     : b-coefficients (in,complex(0:n))
f     : continued fraction result
```

DESCRIPTION:

Straight forward evaluation of a continued fraction with depth n

$$b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{\cdots + \cfrac{a_n}{b_n}}}}$$

without any checking of convergence.
REVISION HISTORY:

```
Created March 2006 (Sagmeister)
```

1.3.9 genwiqggp (Source File: genwiqggp.F90)

INTERFACE:

```
subroutine genwiqggp(flag, iq, igq1, igq2, clwt)
```

USES:

```
use modinput
use modmpi, only: terminate
use mod_constants, only: pi, twopi, fourpi
use mod_lattice, only: omega, bvec, binv
use mod_qpoint, only: nqpt, ngridq
use modxs, only: vgqc, sptclg, gqc
use m_genfilname
use m_getunit
```

DESCRIPTION:

Effective integrals of Coulomb interaction. See routine `genwiq2`.
REVISION HISTORY:

```
Created February 2008 (Sagmeister)
Added comments, formatting and truncated potentials (Aurich)
```

1.3.10 genpmatxs (Source File: genpmatxs.F90)

INTERFACE:

```
Subroutine genpmatxs (ngp, igpig, vgpc, evecfv, evecsv, pmat)
```

USES:

```
Use modinput
Use modmain
Use modxs, Only: apwcmt, locmt, ripaa, ripalo, riploa, riplolo
Use mod_variation, only: variation_multiplication
```

INPUT/OUTPUT PARAMETERS:

```
ngp    : number of G+p-vectors (in,integer)
igpig  : index from G+p-vectors to G-vectors (in,integer(ngkmax))
vgpc   : G+p-vectors in Cartesian coordinates (in,real(3,ngkmax))
evecfv : first-variational eigenvector (in,complex(nmatmax,nstfv))
evecsv : second-variational eigenvectors (in,complex(nstsv,nstsv))
pmat   : momentum matrix elements (out,complex(3,nstsv,nstsv))
```

DESCRIPTION:

Calculates the momentum matrix elements

$$p_{ij} = \langle \Psi_{i,\mathbf{k}} | -i\nabla | \Psi_{j,\mathbf{k}} \rangle.$$

The gradient is applied explicitly only to the radial functions and corresponding spherical harmonics for the muffin-tin part. In the interstitial region the gradient is evaluated analytically. Parts taken from the routine `genpmat`.

REVISION HISTORY:

```
Created April 2008 (Sagmeister)
```

1.3.11 writepmatxs (Source File: writepmatasc.F90)

INTERFACE:

```
subroutine writepmatasc
```

USES:

```
use modinput, only: input
use modmpi, only: procs, rank, firstofset, lastofset, barrier
use mod_misc, only: task, filext
use mod_kpoint, only: nkpt, vkl
use mod_Gkvector, only: ngkmax, vgkl, ngk, gkc, tpgkc, sfacgk, igkig, vgkc
use mod_APW_LO, only: apwordmax, nlotot, nlomax, lolmax
use mod_muffin_tin, only: lmmaxapw
use mod_atoms, only: natmtot
use mod_eigensystem, only: nmatmax
use mod_eigenvalue_occupancy, only: nstfv, nstsv
use modxas, only: ncg
use modxs, only: tscreen, fnpmat, fnpmat_t, kpari,&
               & kparf, hybridhf, ripaa, ripalo,&
               & riploa, riplolo, apwcmt, locmt,&
               & unitout, iqmtgamma, fhdf5
```

```
use m_putpmat
use m_genfilname
use mod_hdf5
use m_getunit, only: getunit
```

DESCRIPTION:

Calculates the momentum matrix elements using routine `genpmat` and writes them to direct access file `PMAT.OUT`, `PMAT_XS.OUT` or `PMAT_SCR.OUT` depending on the context of the execution.

REVISION HISTORY:

```
Created 2006 (S. Sagmeister)
Modifications, August 2010 (S. Sagmeister)
Re-purposed for HDF5 output in BSE module, July 2017 (C. Vorwerk)
```

1.3.12 writegqpts (Source File: writegqpts.F90)

INTERFACE:

```
Subroutine writegqpts (iq, filex, dirname)
```

USES:

```
Use modmain
Use modxs, only: unit1, ngq, vgqc, vgql, gqc
Use mod_qpoint, only: vqc
Use m_getunit
```

DESCRIPTION:

Writes the $\mathbf{G}+\mathbf{q}$-points in lattice coordinates, Cartesian coordinates, and lengths of $\mathbf{G}+\mathbf{q}$-vectors to the file `GQPOINTS.OUT`.

REVISION HISTORY:

```
Created October 2006 (Sagmeister)
```

1.3.13 kernxc (Source File: kernxc.F90)

INTERFACE:

```
Subroutine kernxc
```

DESCRIPTION:

Computes the ALDA exchange-correlation kernel. In the muffin-tin, the density is transformed from spherical harmonic coefficients ρ_{lm} to spherical coordinates (θ, ϕ) with a backward spherical harmonic transformation (SHT). Once calculated, the exchange-correlation potential and energy density are transformed with a forward SHT. This routine is based upon the routine `potxc`.

REVISION HISTORY:

```
Created March 2007 (Sagmeister)
```

1.3.14 findgqmap (Source File: findgqmap.F90)

INTERFACE:

```
subroutine findgqmap(iq, iqr, nsc, sc, ivgsc, n, isc, isci, ivgu, igqmap)
```

USES:

```
use modmpi, only: terminate
use modinput, only: input
use mod_qpoint, only: iqmap
use mod_lattice, only: bvec
use modxs, only: vqlr, scimap, igqig, ivgigq, ivqr
use mod_symmetry, only: lsplsymc, symlat, maxsymcrys
use mod_Gvector, only: ivg
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  integer(4), iq : Index of non-reduced q-point
  integer(4), iqr : Index of reduced q-point
  integer(4), nsc : Number of symmetry operations that map iqr into iq
  integer(4), sc(maxsymcrys) : The first nsc elements are the indices of the
                               corresponding crystal symmetries
  integer(4), ivgsc(3, maxsymcrys) : The G vectors that shift the rotated vqlr to
                                     the unit cell
  integer(4), n   : Number of G+q vectors
OUT:
  integer(4), isc : Crystal symmety operation that maps qr to q
  integer(4), isci : Inverse crystal symmetry operation that maps q to qr
  integer(4), ivgu(3) : Contains -G_s of S^T_{isc}*qr = q + G_s
  integer(4), igqmap(n) : Index mapping between the set of G+iq vectors and
                          and the G+iqrnr vectors
```

DESCRIPTION:

Given an non-reduced and a reduce q point index (iq, iqr) the routine tries to find a symmetry operation that mapps q onto qr and also keeps all rotated $\vec{G}+\vec{q}$ vectors $\mathbf{S}*(\vec{G}+\vec{q}) = \vec{G}_1 + \vec{q}_\mathrm{r}$ shorter than `gqmax`. If successful the routine returns an index map between the set of $\left\{\vec{G}+\vec{q}\right\}$ and the $\left\{\vec{G}+\vec{q}_r\right\}$ vectors.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich 2016)
Changed formatting and added comments. (Aurich 2016)
```

1.3.15 ematqk (Source File: m_ematqk.f90)

INTERFACE:

```
subroutine ematqk(iq, ik, emat, bc)
```

USES:

```
      use modinput, only: input
      use mod_misc, only: task, filext
      use mod_constants, only: zzero, zone
      use mod_eigenvalue_occupancy, only: nstfv
      use mod_muffin_tin, only: idxlm
      use mod_Gkvector, only: gkmax
      use mod_Gvector, only: intgv
      use mod_APW_LO, only: nlotot, apword, nlorb, lorbl
      use mod_Gvector, only: gc, ngvec, cfunig, ivg, ivgig
      use mod_qpoint, only: vql
      use mod_atoms, only: nspecies, natmtot, natoms, idxas
      use modxs, only: bcbs, msg, xiohalo,&
                     & xiuhloa, &
                     & apwmaxsize, apwsize, losize,&
                     & lomaxsize, cmtfun0, cmtfun,&
                     & ngq, igqig,&
                     & fnetim, fftmap_type,&
                     & cpumtaa, cpumtalo, cpumtloa, cpumtlolo,&
                     & filext0, iqmt0, iqmt1
      use summations, only: doublesummation_simple_cz
      use m_getapwcmt
      use m_getlocmt
      use m_putemat
      use m_emattim
      use m_getunit
      use m_genfilname
      use m_getgrst, only: getevecfv1, getevecfv0
      use mod_spin, only: nspnfv
#ifdef USEOMP
      use omp_lib
#endif
```

DESCRIPTION:

Calculates plane wave elements between state ranges defined in (bcemat.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
Substituted the modxs:xiou reference with the pointer modxs:emat that
can point to any of the modxs:xiXY.
```

1.3.16 ematqk (Source File: m_ematqk.f90)

INTERFACE:

```
subroutine ematqk_core(iq, ik, emat, bc, flag)
```

USES:

```
use modinput, only: input
use modxas, only: nxas
use mod_misc, only: task
use mod_constants, only: zzero, zone
use mod_muffin_tin, only: lmmaxapw
use mod_APW_LO, only: nlotot, apwordmax
use mod_atoms, only:  natmtot
use modxs, only: bcbs, msg, ngq,&
              & fnetim, fftmap_type,&
              & cpumtaa, cpumtalo, cpumtloa, cpumtlolo
use summations, only: doublesummation_simple_cz
use m_getapwcmt
use m_getlocmt
use m_putemat
use m_emattim
use m_getunit
use m_genfilname
use mod_spin, only: nspnfv
use mod_eigensystem, only: nmatmax_ptr
use m_getgrst, only: getevecfv1, getevecfv0, getevecsv0, &
  & getevecsv1, match0, match1
use mod_eigenvalue_occupancy, only: nstsv


#ifdef USEOMP
      use omp_lib
#endif
```

DESCRIPTION:

Calculates plane wave elements between state ranges defined in (bcemat.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
Substituted the modxs:xiou reference with the pointer modxs:emat that
can point to any of the modxs:xiXY.
```

1.3.17 ematradoo (Source File: m_ematqk.f90)

INTERFACE:

```
Subroutine ematradoo (iq,ik, igq, integral)
```

USES:

```
Use modmain
     use modinput, only: input
     use mod_muffin_tin, only: nrmt, nrmtmax
     use mod_atoms, only: spr
     use modxs, only: gqc
     use modxas, only: xasstart, nxas, ucore
Use modxas
```

INPUT/OUTPUT PARAMETERS:

```
iq       : q-point position (in,integer)
ik       : k-point position (in,integer)
igq      : (q+G)-point position (in,integer)
integral : radial planewave integral
           (out, complex(lmaxemat+1,nxas,nxas))
```

DESCRIPTION:

Calculates the radial integral part $R^l_{\mu\mu'}(\mathbf{q}+\mathbf{G})$ of the planewave matrix element between two core states. See Vorwerk's Master thesis for more details.

REVISION HISTORY:

```
Based on the subroutine ematqk.F90
Created November 2015 (Christian Vorwerk)
```

1.3.18 ematradou (Source File: m_ematqk.f90)

INTERFACE:

```
Subroutine ematradou (ik, iq, igq, ngp, apwalm, evecfvo, bcs, integral)
```

USES:

```
use modinput, only: input
use modxs, only: bcbs, gqc
use mod_atoms, only: spr, natmtot
use modxas, only: xasstart, nxas, ucore
use mod_muffin_tin, only: idxlm, nrcmt, nrmt, lmmaxapw, nrmtmax, &
  & nrcmtmax
use mod_eigenvalue_occupancy, only: nstfv, nstsv
use mod_Gkvector, only: ngkmax
use mod_APW_LO, only: apwordmax
use m_getgrst, only: wavefmt1, getevecsv1, wavefmtsv1
```

```
        use mod_constants, only: zzero, zi
        use mod_misc, only: filext
  Use m_getunit
  Use modxas
```

INPUT/OUTPUT PARAMETERS:

```
    iq       : q-point position (in,integer)
    ngp      : number of G+p-vectors (in,integer)
    apwalm   : APW matching coefficients
               (in,complex(ngkmax,apwordmax,lmmaxapw,natmtot))
    evecfvo  : first-variational eigenvector (in,complex(nmatmax))
    integral : radial planewave integral
               (out, complex(lmaxemat+1,lmmaxapw,nxas,sta2:sto2))
```

DESCRIPTION:

Calculates the radial integral part $R^{ll'}_{\mu m(\mathbf{k}+\mathbf{q})}(\mathbf{q}+\mathbf{G})$ of the planewave matrix element between a core state and a conduction state. See Vorwerk's Master thesis for more details.

REVISION HISTORY:

```
  Based on the subroutine ematqk.F90
  Created November 2015 (Christian Vorwerk)
```

1.3.19 ematraduo (Source File: m_ematqk.f90)

INTERFACE:

```
    Subroutine ematraduo (ik,iq, igq, ngp, apwalm, evecfvo, evecsvt, bcs, integral)
```

USES:

```
        use modinput, only: input
        use modxs, only: bcbs, gqc, filext0
        use mod_atoms, only: spr, natmtot
        use modxas, only: xasstart, nxas, ucore
        use mod_muffin_tin, only: idxlm, nrcmt, nrmt, lmmaxapw, nrmtmax, &
          & nrcmtmax
        use mod_eigenvalue_occupancy, only: nstfv, nstsv
        use mod_Gkvector, only: ngkmax
        use mod_APW_LO, only: apwordmax
        use mod_constants, only: zzero, zi
        use m_getgrst, only: wavefmt0, getevecsv0, wavefmtsv0
```

INPUT/OUTPUT PARAMETERS:

```
    iq       : q-point position (in,integer)
    ik       : k-point position (in,integer)
    ngp      : number of G+p-vectors (in,integer)
```

```
apwalm   : APW matching coefficients
           (in,complex(ngkmax,apwordmax,lmmaxapw,natmtot))
evecfvo  : first-variational eigenvector (in,complex(nmatmax))
integral : radial planewave integral
           (out, complex(lmaxemat+1,lmmaxapw,nxas,sta2:sto2))
```

DESCRIPTION:

Calculates the radial integral part $R^{ll'}_{\mu m(\mathbf{k}+\mathbf{q})}(\mathbf{q}+\mathbf{G})$ of the planewave matrix element between a core state and a conduction state. See Vorwerk's Master thesis for more details.

REVISION HISTORY:

```
Based on the subroutine ematqk.F90
Created November 2015 (Christian Vorwerk)
```

1.3.20 ematsumou (Source File: m_ematqk.f90)

INTERFACE:

```
Subroutine ematsumou (iq, igq, bcs, integral, xi)
```

USES:

```
Use modmain
       Use mod_muffin_tin, only: idxlm, lmmaxapw
       Use mod_constants, only: zzero, zil, fourpi
       Use mod_atoms, only: idxas
       Use mod_kpoint, only: vkl
       Use modinput, only: input
       Use modxs, only: sfacgq, bcbs, xsgntou, xsgntousv, ylmgq
       use modxas, only: nxas
Use m_getunit
Use modxas
```

INPUT/OUTPUT PARAMETERS:

```
iq       : q-point position (in,integer)
igq  : (q+G)-point position (in, integer)
integral : radial planewave integral
           (in, complex(lmaxemat+1,lmmaxapw,nxas,sta2:sto2))
xi       : planewave matrix element (inout,complex(nxas, sta2:sto2, ngq(iq))
```

DESCRIPTION:

Calculates planewave matrix elements between a core and a conduction state, using the radial integrals. Fore more information, see Christian Vorwerk's Master thesis, Eq. (5.20).

REVISION HISTORY:

```
Based on the subroutine ematqkgmt.F90
Created November 2015 (Christian Vorwerk)
```

1.3.21 gensumrls (Source File: gensumrls.F90)

INTERFACE:

```
Subroutine gensumrls (w, eps, sumrls)
```

USES:

```
Use modxs
Use modmpi
```

INPUT/OUTPUT PARAMETERS:

```
w      : frequency grid (in,real(:))
eps    : dielectric function tensor component (in,complex(:))
sumrls : values of three different sumrules (out,real(3))
```

DESCRIPTION:

Expressions for the sumrules taken from C. Ambrosch-Draxl, CPC 175 (2006) 1-14, p5, Eq. 26.

REVISION HISTORY:

```
Created March 2006 (Sagmeister)
```

1.3.22 copyfilesq0 (Source File: copyfilesq0.F90)

INTERFACE:

```
Subroutine copyfilesq0
Use modinput
```

USES:

```
Use modmain
Use modxs
Use m_getapwcmt
Use m_getlocmt
```

DESCRIPTION:

If a finite momentum transfer Q-point is the Gamma-point, eigenvalues eigenvectors, occupancies and muffin-tin expansion coefficients are identical to those corresponding to the unshifted mesh. Files are copied using the associated generic routines for ISO compatibility, or links are generated if ISO compatibility is dropped.

REVISION HISTORY:

```
Created January 2008 (Sagmeister)
```

1.3.23 genfilextread (Source File: genfilextread.F90)

INTERFACE:

```
subroutine genfilextread(task)
```

USES:

```
use modinput
use m_genfilname
```

INPUT/OUTPUT PARAMETERS:

```
!IN:
integer(4) :: task ! Excitings faboulus task number
```

DESCRIPTION:

This routine sets the filename extension for the eigenvalue `EVALSV` and eigenvector `EVECSV` files corresponding to zero momentum transfer depending on the calling task. Used in `findocclims`.

REVISION HISTORY:

```
Added to documentaion scheme. 2016 (Aurich)
Changed so that only task 'screen' uses '_SCR.OUT' (Aurich)
```

1.3.24 writepwmat (Source File: writepwmat.F90)

INTERFACE:

```
Subroutine writepwmat
```

USES:

```
Use modmain
Use modxs
Use m_genfilname
```

DESCRIPTION:

Calculates the matrix elements of the plane wave $e^{-i(\mathbf{G}+\mathbf{q})\mathbf{r}}$ using routine `genpwmat` and writes them to direct access file `PWMAT.OUT`.

REVISION HISTORY:

```
Created November 2007 (Sagmeister)
```

1.3.25 zoutpr (Source File: zoutpr.F90)

INTERFACE:

```
Subroutine zoutpr (n1, n2, alpha, x, y, a)
```

INPUT/OUTPUT PARAMETERS:

```
n1,n2 : size of vectors and matrix, respectively (in,integer)
alpha : complex constant (in,complex)
x     : first input vector (in,complex(n1))
y     : second input vector (in,complex(n2))
a     : output matrix (out,complex(n1,n2))
```

DESCRIPTION:

Performs the rank-2 operation

$$A_{ij} \rightarrow \alpha \mathbf{x}_i^* \mathbf{y}_j + A_{ij}.$$

REVISION HISTORY:

```
Created April 2008 (Sagmeister)
```

1.3.26 findsymi (Source File: findsymi.F90)

INTERFACE:

```
Subroutine findsymi (epslat, maxsymcrys, nsymcrys, symlat, lsplsymc, &
& vtlsymc, isymlat, scimap)
```

USES:

```
use modmpi
```

DESCRIPTION:

Throughout the code the symmetries are understood to be applied in a way

$$(\alpha_S|\alpha_R|\mathbf{t})\mathbf{x} = \alpha_S \alpha_R(\mathbf{x}+\mathbf{t})$$

which is different from the commonly used definition $\{\alpha|\tau\}x = \alpha x + \tau$ – see routine `findsymcrys`. This difference affects the inverse of the fractional translation but has no effect on the inverse of the rotational part, so the inverse spacegroup symmetry operations are the same for both definitions.

REVISION HISTORY:

```
Created April 2007 (Sagmeister)
```

1.3.27 dzmatmult (Source File: m_dzmatmult.f90)

INTERFACE:

```
subroutine dzmatmult(zma, zmb, zmc,&
   & m, n, k, ia, ja, ib, jb, ic, jc,&
   & alpha, beta, transa, transb)
```

USES:

```
use modmpi
use modscl
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  type(dzmat) :: zma    ! A distributed complex matrix
  type(dzmat) :: zmb    ! B distributed complex matrix
  integer(4), optional :: m, n, k ! Dimensions for matrix matrix multiplication
  integer(4), optional :: ix, jx  ! Matrix subselection for global arrays (x=a,b,c)
                                  ! Indicates upper left corner of sub-matrix in
                                  ! gobal matrix indices.
  complex(8), optional :: alpha, beta      ! Scaling factors for alpha*A*B+beta*C=C
  character(1), optional :: transa, transb ! Op(A), Op(B), Op='N','T','C'
IN/OUT:
  type(dzmat) :: zmc  ! C distributed complex matrix
```

DESCRIPTION:

Wrapper for BLAS's complex matrix-matrix multiply routine `ZGEEM` and its PBLAS's counterpart `PZGEEM`. The α and β parameters default to 1 and 0. The `TRANSA TRANSB` parameter default no "N". The ix and jx defalut to 1 and 1. m, n, k defalut values are set to the respective global dimensions of OP(M), where M is the respective matrix.

REVISION HISTORY:

```
Created 2016 (Aurich)
```

1.3.28 writesymt2 (Source File: writesymt2.F90)

INTERFACE:

```
Subroutine writesymt2
```

USES:

```
Use modmain
Use modxs
```

DESCRIPTION:

Outputs the symmetrization matrices for the tensor components of a rank-2 tensor. The tensor(-field) t_{ij} in reciprocal space must be invariant under coordinate transforms of the system wrt. the rotational part of the crystal symmetries, so we can average:

$$t_{ij}^{\rm sym} = \frac{1}{N_\alpha} \sum_\alpha \sum_{k,l} \alpha_{ik} \alpha_{jl} t_{kl}.$$

The symmetrized tensor $t_{ij}^{\rm sym}$ can then be written as

$$t_{ij}^{\rm sym} = \sum_{k,l} T_{ij,kl} t_{kl},$$

with the symmetrization tensor

$$T_{ij,kl} = \frac{1}{N_\alpha} \sum_\alpha \alpha_{ik} \alpha_{jl}$$

where N_α is the number of symmetry operations in the space group. For each component ij the symmetrization tensor $T_{ij,kl}$ is written as a matrix in the components kl to the file `SYMT2.OUT`.

REVISION HISTORY:

`Created October 2008 (Sagmeister)`

1.3.29 dyson (Source File: dyson.F90)

INTERFACE:

```
Subroutine dyson (n, s0, k, s)
```

USES:

```
Use invert
```

INPUT/OUTPUT PARAMETERS:

```
n     : matrix size of local field effects (in,integer)
s0    : S0 matrix (in,complex(:,:))
k     : kernel matrix (in,complex(:,:))
s     : S (solution) matrix (in,complex(:,:))
```

DESCRIPTION:

Solve Dyson's equation

$$S = S_0 + S_0 K S$$

for S by inversion;

$$S = \left[1 + S_0 K\right]^{-1} S_0.$$

The inversion is carried out using the LAPACK routines `zgetrf` and `zgetri`.

REVISION HISTORY:

`Created March 2005 (Sagmeister)`

1.3.30 getevalsv0 (Source File: getevalsv0.F90)

INTERFACE:

```
subroutine getevalsv0(vpl, evalsvp)
```

USES:

```
use mod_eigenvalue_occupancy, only: nstsv
use mod_kpoint, only: nkptnr, vkl
use mod_misc, only: task, filext
use modxs, only: vkl0, filext0, usefilext0
```

INPUT/OUTPUT PARAMETERS:

```
IN:
real(8) :: vpl(3) ! k-point vector in lattice coordinates
OUT:
real(8) :: evalsvp(nstsv) ! Eigenvalues

!DESCTIPTION:
  This routine reads the eigenvalues for a given k-vector
  form the eigenvalue file assoziated with zero momentum
  transfer.
```

REVISION HISTORY:

```
Added to documentation scheme. 2016 (Aurich)
```

1.3.31 exccoulintlauncher (Source File: exccoulintlauncher.f90)

INTERFACE:

```
subroutine exccoulintlauncher
```

USES:

```
use modmpi
use modxs, only: unitout
use modinput, only: input
use modbse
```

DESCRIPTION:

Launches the calculation of the exchange term of the Bethe-Salpeter Hamiltonian for the specified momentum transfer vectors $\vec{Q}_{\mathrm{mt}}$.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.3.32 kramkron (Source File: kramkron.F90)

INTERFACE:

```
Subroutine kramkron (i1, i2, eps, n, w, im, re)
```

INPUT/OUTPUT PARAMETERS:

```
i1,i2 : tensor components of dielectric function tensor (in,integer)
eps   : zero frequency tolerance
n     : number of frequency points (in,integer)
w     : frequency grid (in,real(n))
im    : imaginary part of dielectric function tensor component (in,real(n))
re    : real part of dielectric function tensor component (out,real(n))
```

DESCRIPTION:

Performs a Kramers-Kronig transformation from the imaginary part to the real part of the dielectric function. Algorithm taken from routine `linopt`.

REVISION HISTORY:

```
Created November 2007 (Sagmeister)
```

1.3.33 getoccsv0 (Source File: getoccsv0.F90)

INTERFACE:

```
subroutine getoccsv0(vpl, occsvp)
```

USES:

```
use mod_eigenvalue_occupancy, only: nstsv
use mod_kpoint, only: nkptnr, vkl
use mod_misc, only: task, filext
use modxs, only: vkl0, filext0, usefilext0
```

INPUT/OUTPUT PARAMETERS:

```
IN:
real(8) :: vpl(3) ! k-point vector in lattice coordinates
OUT:
real(8) :: occsvp(nstsv) ! Eigenvalues
```

DESCRIPTION:

This routine reads the occupancies for a given k-vector form the occupancy file associated with zero momentum transfer.

REVISION HISTORY:

```
Added to documentation scheme. 2016 (Aurich)
```

1.3.34 transik (Source File: transik.F90)

INTERFACE:

```
logical function transik(ik)
```

USES:

```
use modinput
```

DESCRIPTION:

This function returns "true" if the transition specified by the input k-point initial and final state matches with the ones specified in the `transitions` element. Wether a state is included or excluded depends on the sequence of the transitions specified (using the "include" and "exclude" attribute).

REVISION HISTORY:

```
Created July 2010 (Sagmeister)
```

1.3.35 putpmat (Source File: putpmat.F90)

INTERFACE:

```
subroutine putpmat(ik, filnam, pm, tag)
```

USES:

```
use modmain
use modmpi
use m_getunit
```

INPUT/OUTPUT PARAMETERS:

```
IN:
integer(4) :: ik
character(*) :: filnam
integer(4) :: tag
IN/OUT:
complex(8) :: pm(:,:,:)
```

DESCRIPTION:

The routine collects the momentum matrix elements for each k-point form the `MPI` processes and writes them to a direct access file. Note: The content of pm is destroyed on exit.

REVISION HISTORY:

```
Added to documentation scheme. 2016 (Aurich)
Removed parts which literately did nothing. (Aurich)
```

1.3.36 genylmgq (Source File: genylmgq.F90)

INTERFACE:

```
Subroutine genylmgq (iq, lmax)
```

USES:

```
Use modmain
Use modxs
```

DESCRIPTION:

Generates a set of spherical harmonics, $Y_{lm}(\widehat{\mathbf{G}+\mathbf{q}})$, with angular momenta up to `lmax` for the set of $\mathbf{G}+\mathbf{q}$-vectors. Based upon the routine genylmg.

REVISION HISTORY:

```
Created October 2006 (Sagmeister)
```

1.3.37 symtr2 (Source File: symtr2.F90)

INTERFACE:

```
Subroutine symtr2 (t2)
```

USES:

```
Use modmain
Use modxs
```

DESCRIPTION:

Symmetrizes a rank-2 tensor with respect to the rotational part of the crystal symmetries:

$$t_{ij}^{\mathrm{sym}}=\frac{1}{N_{\alpha}}\sum_{\alpha}\sum_{k,l=1}^{3}\alpha_{ik}\alpha_{jl}t_{kl}.$$

Here, t_{ij} are the components of the rank-2 tensor, α_{ij} denotes the rotational part of the crystal symmetries and N_{α} stands for the total number of crystal symmetries.

REVISION HISTORY:

```
Created October 2008 (Sagmeister)
```

1.3.38 setupblacs (Source File: modscl.f90)

INTERFACE:

```
subroutine setupblacs(mpicom, typechars, blacscom, np)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  type(mpiinfo) :: mpicom    ! Info type for the MPI communicator to be
                             ! used as the basis for BLACS
  character(*)  :: typechars ! String to indicate 2d or 1d BLACS grid
  integer(4), optional :: np ! Number of ranks to be used
OUT:
  type(blacsinfo) :: blacscom  ! Info type for the resulting BLACS grid
```

DESCRIPTION:

This routine creates an BLACS process grid on top of an existing `MPI` communicator. Grid shape options are “2d”, “1dr” and “1dc”. The `MPI` ranks are distributed in a row-major fashion.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.3.39 exitblacs (Source File: modscl.f90)

subroutine blacsbarrier(blacscom) !INPUT/OUTPUT PARAMTERS: IN: type(blacsinfo) :: blacscom
This routine waits untill all porcesses of the passed `BLACS` grid have reached it and then terminated the grid.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.3.40 exitblacs (Source File: modscl.f90)

subroutine exitblacs(blacscom) !INPUT/OUTPUT PARAMTERS: IN: type(blacsinfo) :: blacscom
This routine waits untill all porcesses of the passed `BLACS` grid have reached it and then terminated the grid.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.3.41 new_dzmat (Source File: modscl.f90)

INTERFACE:

```
  subroutine new_dzmat(self, nrows, ncols, binfo, rblck, cblck, fzero)
!INPUTP/OUTPUT PARAMETERS:
IN:
  integer(4) :: nrows      ! Global number of rows of self
  integer(4) :: ncols      ! Global number of columns of self
  type(blacsinfo) :: binfo ! Info type for BLACS grid the matrix
                           ! is to be distrubuted over
  integer(4), optional :: rblck, cblck ! Blocking size of rows and columns
IN/OUT:
  type(dzmat) :: self ! The distributed complex(8) matrix
```

DESCRIPTION:

This routine initialized a block cyclic distributed global matrix for a given `BLACS` process grid. It sets up descriptor arrays, builds index maps and allocates the local matrix. In the case of compilation without `ScaLAPACK` the global matrix is identical to the local matrix.

REVISION HISTORY:

```
  Created. 2016 (Aurich)
```

1.3.42 setview_dzmat (Source File: modscl.f90)

INTERFACE:

```
  subroutine setview_dzmat(self, nrows, ncols, ir, ic)
!INPUTP/OUTPUT PARAMETERS:
IN:
  integer(4) :: nrows      ! Number of rows for submatrix
  integer(4) :: ncols      ! Number of columns for submatrix
  integer(4) :: ir, ic     ! Upper left corner of submatrix within the full matrix
IN/OUT:
  type(dzmat) :: self ! The distributed complex(8) matrix
```

DESCRIPTION:

This routine sets the current submatrix view of a distributed matrix.

REVISION HISTORY:

```
  Created. 2016 (Aurich)
```

1.3.43 del_zmat (Source File: modscl.f90)

INTERFACE:

```
subroutine del_dzmat(self)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  type(dzmat) :: self
```

DESCRIPTION:

This routine deallocated a distributed complex(8) matrix.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.3.44 dzmat_send2global_root (Source File: modscl.f90)

INTERFACE:

```
subroutine dzmat_send2global_root(mat, dmat, binfo)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  type(dzmat) :: dmat      ! Distributed complex(8) matrix
  type(blacsinfo) :: binfo ! Info type for BLACS grid
OUT:
  complex(8), allocatable :: mat(:,:) ! Global matrix
```

DESCRIPTION:

This routine collects a distributed complex matrix in a global array on porcess $(0, 0)$.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.3.45 dzmat_send2global_all (Source File: modscl.f90)

INTERFACE:

```
subroutine dzmat_send2global_all(mat, dmat, binfo)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  type(dzmat) :: dmat      ! Distributed complex(8) matrix
  type(blacsinfo) :: binfo ! Info type for BLACS grid
OUT:
  complex(8), allocatable :: mat(:,:) ! Global matrix
```

DESCRIPTION:

This routine collects a distributed complex matrix in a global array on all porcesses of the grid.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.3.46 dzmat_global2local (Source File: modscl.f90)

INTERFACE:

```
subroutine dzmat_copy_global2local(mat, dmat, binfo)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  complex(8) :: mat(:,:)   ! Global matrix
  type(blacsinfo) :: binfo ! Info type for BLACS grid
  integer(4), optional :: rblck, cblck ! row and column block size
IN/OUT:
  type(dzmat) :: dmat      ! Distributed complex(8) matrix
```

DESCRIPTION:

This routine takes a global complex matrix and ceates a 2D block cyclicly distributed copy of it.

REVISION HISTORY:

```
Created. 2017 (Aurich)
```

1.3.47 dzmat_copy (Source File: modscl.f90)

INTERFACE:

```
subroutine dzmat_copy(context, m, n, dmata, dmatb, ra, ca, rb, cb)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  integer(4) :: context ! Context that is at least the union of contexts A and B
  integer(4) :: m, n    ! Number of rows and columns of the global submatrix of A
  type(dzmat), optional :: dmata  ! Distributed complex(8) matrix
  integer(4), optional  :: ra, ca ! Upper left corner coordinates of the submatrix in A
  integer(4), optional  :: rb, cb ! Upper left corner coordinates of the submatrix in B
OUT:
  type(dzmat), optional :: dmatb  ! Distributed complex(8) matrix
```

DESCRIPTION:

This routine copys the submatrix of the distributed matrix A to a equally sized submatrix of the distributed matrix B. The passed context must contain the porcesses holding A and B, all processes of that context must call this routine.

REVISION HISTORY:

```
Created. 2016 (Aurich)
```

1.3.48 fxc_alda_check (Source File: fxc_alda_check.F90)

INTERFACE:

```
Subroutine fxc_alda_check
```

USES:

```
Use modinput
Use modmain, Only: lmmaxvr, ngrtot, rhoir, rhomt, vxcir, vxcmt,xctype
Use modxcifc, Only: xcifc
Use modfxcifc
```

DESCRIPTION:

Checks the validity of the analytical expressions for the ALDA exchange-correlation kernel.

REVISION HISTORY:

```
Created April 2007 (Sagmeister)
```

1.3.49 genfilname (Source File: genfilname.F90)

INTERFACE:

```
Subroutine genfilname (nodotpar, basename, etype, asc, bzsampl, &
& acont, nar, tord, nlf, fxctype, fxctypestr, scrtype, bsetype, markfxcbse, &
& tq0, oc1, oc2, iq, iqmt, procs, rank, dotext, setfilext, &
& revertfilext, appfilext, filnam, fileext, auxtype, lambda, dirname)
```

USES:

```
use modmpi, only: terminate
Use modmain, Only: filext
Use modxs, Only: filextrevert,skipgnd
```

DESCRIPTION:

Generates file name and extension according to optional input parameters (see routine). Interpret bzsampl variable as default (Lorentzian) for 0, as tetrahedron method for 1. Trilinear method to be followed.

REVISION HISTORY:

```
Created October 2007 (Sagmeister)
```

2 Introduction

Welcome to the XS/EXCITING code developers' manual. This manual is supposed to collect the routines and modules belonging exclusively to the excited states (TDDFT and BSE) part into one document. The content of this manual is partially taken from the author's PhD thesis.

& S. Sagmeister& Leoben, August 2008

2.0.50 gengqvec (Source File: gengqvec.F90)

INTERFACE:

```
Subroutine gengqvec (iq, vpl, vpc, ngp, igpig, vgpl, vgpc, gpc, tpgpc)
```

USES:

```
Use modinput
Use modmain
Use modxs
```

INPUT/OUTPUT PARAMETERS:

```
vpl   : p-point vector in lattice coordinates (in,real(3))
vpc   : p-point vector in Cartesian coordinates (in,real(3))
ngp   : number of G+p-vectors returned (out,integer)
igpig : index from G+p-vectors to G-vectors (out,integer(ngkmax))
vgpl  : G+p-vectors in lattice coordinates (out,real(3,ngkmax))
vgpc  : G+p-vectors in Cartesian coordinates (out,real(3,ngkmax))
gpc   : length of G+p-vectors (out,real(ngkmax))
tpgpc : (theta, phi) coordinates of G+p-vectors (out,real(2,ngkmax))
```

DESCRIPTION:

Generates a set of $\mathbf{G}+\mathbf{p}$-vectors for the input $\mathbf{p}$-point with length less than `gkmax`. These are used as the plane waves in the APW functions. Also computes the spherical coordinates of each vector. Based on `gengpvec`.

REVISION HISTORY:

```
Created October 2006 (Sagmeister)
```

2.0.51 xssave0 (Source File: xssave0.F90)

INTERFACE:

```
subroutine xssave0
```

USES:

```
use mod_kpoint, only: nkptnr, nkpt, vkl
use mod_spin, only: nspnfv
use mod_Gkvector, only: ngkmax, ngk, igkig, vgkl,&
                      & vgkc, gkc, tpgkc, sfacgk
use mod_atoms, only: natmtot
use mod_eigensystem, only: nmatmax, nmat
use modxs, only: vkl0, ngk0, igkig0, vgkl0,&
               & vgkc0, gkc0, tpgkc0, sfacgk0,&
               & nmat0, nkpt0, ngkmax0,&
               & nmatmax0
```

DESCRIPTION:

This routine should be called after init0, init1 and init2 in order to save variables related to the k-point set for $\mathbf{q} = 0$.

REVISION HISTORY:

```
Created March 2005 (Sagmeister)
```

2.0.52 connecta (Source File: connecta.F90)

INTERFACE:

```
Subroutine connecta (cvec, nv, np, vvl, vpl, dv, dp)
```

INPUT/OUTPUT PARAMETERS:

```
cvec : matrix of (reciprocal) lattice vectors stored column-wise
       (in,real(3,3))
nv   : number of vertices (in,integer)
np   : number of connecting points (in,integer)
vvl  : vertex vectors in lattice coordinates (in,real(3,nv))
vpl  : connecting point vectors in lattice coordinates (out,real(3,np))
dv   : cummulative distance to each vertex (out,real(nv))
dp   : cummulative distance to each connecting point (out,real(np))
```

DESCRIPTION:

Generates a set of points which interpolate between a given set of vertices. Vertex points are supplied in lattice coordinates in the array `vvl` and converted to Cartesian coordinates with the matrix `cvec`. Interpolating points are stored in the array `vpl`. The cummulative distances to the vertices and points along the path are stored in arrays `dv` and `dp`, respectively. The given vertex points are contained in the set of output vectors. Based upon the routine `connect`.

REVISION HISTORY:

```
Created June 2007 (Sagmeister)
```

2.0.53 getridx (Source File: getridx.F90)

INTERFACE:

```
subroutine getridx(set, i, ir)
```

USES:

```
  use modmpi, only: firstofset, procofindex
```

INPUT/OUTPUT PARAMETERS:

```
IN:
integer :: set ! Number of elements distributed over MPI
integer :: i   ! Absolut element index
OUT:
integer :: ir  ! Relative element index with respect to the set of elements
               ! the current rank was given
```

DESCRIPTION:

Get index ir relative to partition determined by overall index i. **REVISION HISTORY:**

```
Added to documentation scheme and removed dead code parts. (Aurich)
```

2.0.54 bsedgrid (Source File: bsedgrid.F90)

INTERFACE:

```
Subroutine bsedgrid ()
```

USES:

```
Use modmain
Use modxs
use modinput
Use m_genfilname
Use m_getunit
Use m_genwgrid
Use m_genloss
Use m_gensigma
Use m_gensumrls
Use m_writeeps
Use m_writeloss
Use m_writesigma
Use m_writesumrls
```

DESCRIPTION:

Collects BSE results for all shifted grids in a double grid run and averages using the apropriate weights.
REVISION HISTORY:

```
Created January 2014, S. Kontur
```

2.0.55 bandgap (Source File: bandgap.F90)

INTERFACE:

```
Subroutine bandgap (n, e, ef, egf, ego, ikgf, ikgo, istho)
```

USES:

```
Use modmain
```

DESCRIPTION:

Determines the fundamental and optical band gap if present.

REVISION HISTORY:

```
Created July 2007 (Sagmeister)
```

2.0.56 dysonsym (Source File: dysonsym.F90)

INTERFACE:

```
Subroutine dysonsym (n, s0, k, s)
```

USES:

```
use modmpi
Use invert
Use modxs
```

INPUT/OUTPUT PARAMETERS:

```
n     : matrix size of local field effects (in,integer)
s0    : S0 matrix (in,complex(:,:))
k     : kernel matrix multiplied by S0 from both sides (in,complex(:,:))
s     : S (solution) matrix (in,complex(:,:))
```

DESCRIPTION:

Solve symmetric form of Dyson's equation

$$S = S_0 + S_0(1 + S_0^{-1} K S_0^{-1})S$$

for S by inversion;

$$S = S_0 \left[S_0(1 - S0) - T\right]^{-1} S_0.$$

The inversion is carried out using the LAPACK routines `zgetrf` and `zgetri`.

REVISION HISTORY:

```
Created October 2008 (Sagmeister)
```

2.0.57 fxcifc (Source File: modfxcifc.F90)

INTERFACE:

```
 Subroutine fxcifc (fxctype, w, iw, iq, ng, oct, alrc, alrcd, &
& blrcd, chim, fxcg)
    Use m_fxc_lrc, Only: fxc_lrc
    Use m_fxc_lrcd, Only: fxc_lrcd
    Use m_fxc_alda, Only: fxc_alda
    Use m_fxc_bse_ma03, Only: fxc_bse_ma03
    Use m_fxc_spk, Only: fxc_spk
```

INPUT/OUTPUT PARAMETERS:

```
fxctype : type of exchange-correlation functional (in,integer)
```

DESCRIPTION:

Interface to the exchange-correlation kernel routines. This makes it relatively simple to add new functionals which do not necessarily depend only on all input parameters. Based upon the routine `modxcifc`.

REVISION HISTORY:

```
Created October 2007 (Sagmeister)
```

2.0.58 getfxcdata (Source File: modfxcifc.F90)

INTERFACE:

```
  Subroutine getfxcdata (fxctype, fxcdescr, fxcspin)
```

INPUT/OUTPUT PARAMETERS:

```
    Use modinput
fxctype  : type of exchange-correlation functional (in,integer)
fxcdescr : description of functional (out,character(256))
fxcspin  : spin treatment (out,integer)
fxcgrad  : gradient treatment (out,integer)
```

DESCRIPTION:

Returns data on the exchange-correlation functional labelled by `fxctype`. The character array `fxctype` contains a short description of the functional including journal references. The variable `fxcspin` is set to 1 or 0 for spin-polarised or -unpolarised functionals, respectively.

REVISION HISTORY:

```
Created October 2007 (Sagmeister)
```

2.0.59 getematrad (Source File: getematrad.F90)

INTERFACE:

```
Subroutine getematrad(iqr, iq)
```

USES:

```
use modinput, only: input
use modxs, only: riaa, riloa, rilolo, ngq, ematraddir
use mod_APW_LO, only: lolmax, apwordmax, nlomax
use mod_atoms, only: natmtot
use m_genfilname
use m_getunit
```

INPUT/OUTPUT PARAMETERS:

```
IN:
  integer(4), iqr : Index of non-reduced q-point
  integer(4), iq  : Index of reduced q-point
```

DESCRIPTION:

The routine deallocates, reallocates and reads the radial integral arryas form file. `EMATRAD` is used as file basename.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich 2016)
Changed formatting. (Aurich 2016)
```

2.0.60 ematrad (Source File: ematrad.F90)

INTERFACE:

```
subroutine ematrad(iq)
```

USES:

```
use mod_misc, only: filext
use mod_APW_LO, only: lolmax, apwordmax, nlomax, apword,&
                    & apwfr, nlorb, lorbl, lofr
use mod_atoms, only: natmtot, nspecies, spr, natoms, idxas
use mod_muffin_tin, only: nrmtmax, nrmt
use modinput, only: input
use modxs, only: riaa, riloa, rilolo, ngq, gqc
use m_getunit
```

DESCRIPTION:

This routine is used in the construction of the plane wave matrix elements. It calculates the involved radial integrals for the APW-APW, APW-LO and LO-LO combinations. (Compare the R quantities section 8.1 of Sagmeisters thesis.)
REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

2.0.61 puteps0 (Source File: m_putgeteps0.f90)

INTERFACE:

```
subroutine puteps0(reduced, iq, iw, w, eps0, eps0wg, eps0hd, fname)
```

INPUT/OUTPUT PARAMETERS:

```
In:
logical :: reduced          ! q-point from reduced set
integer(4) :: iq            ! q-point index
integer(4) :: iw            ! w-point index
real(8) :: w                ! iw'th frequency
complex(8) :: eps0(:,:)     ! Body of RPA epsilon matrix
complex(8) :: eps0wg(:,:,:) ! Wings of RPA epsilon matrix
complex(8), optional :: eps0hd(:,:)   ! Body of RPA epsilon matrix
```

DESCRIPTION:

Writes the microscopic V-symmetrized Kohn-Sham dielectric function/tensor $\tilde{\epsilon}^{1}_{\mathbf{GG'}}(\mathbf{q},\omega) = \delta_{\mathbf{GG'}} - \tilde{\chi}^{0}_{\mathbf{GG'}}(\mathbf{q},\omega)$ for given frequency and q point to a direct access file. The record index is equal to frequency index, for each q point a new file is used. In each record the following is saved: iq, vql, ngq, iw, w, eps0(, eps0wg, chhd). Head and wings only present for $q = 0$.
REVISION HISTORY:

```
Created. (Aurich)
```

2.0.62 geteps0 (Source File: m_putgeteps0.f90)

INTERFACE:

```
subroutine geteps0(reduced, iq, iw, w, eps0, eps0wg, eps0hd, fname)
```

USES:

```
use m_getunit
```

INPUT/OUTPUT PARAMETERS:

```
In:
logical :: reduced          ! q-point from reduced set
```

```
integer(4) :: iq            ! q-point index
integer(4) :: iw            ! w-point index
real(8) :: w                ! iw'th frequency
Out:
complex(8) :: eps0(:,:)               ! Body of RPA epsilon matrix
complex(8), optional :: eps0wg(:,:,:) ! Wings of RPA epsilon matrix
complex(8), optional :: eps0hd(:,:)   ! Body of RPA epsilon matrix
```

DESCRIPTION:

Read the microscopic V-symmetrized Kohn-Sham dielectric function/tensor $\tilde{\epsilon}^0_{\mathbf{GG'}}(\mathbf{q},\omega) = \delta_{\mathbf{GG'}} - \tilde{\chi}^0_{\mathbf{GG'}}(\mathbf{q},\omega)$ for given frequency and q point from a direct access file. The record index is equal to frequency index In each record the following is saved: iq, vql, ngq, iw, w, eps0(, eps0wg, chhd). Head and wings only present for $q = 0$.

REVISION HISTORY:

```
Created. (Aurich)
```

2.0.63 bse (Source File: bse.f90)

INTERFACE:

```
subroutine bse(iqmt)
```

USES:

```
Basics
use modinput, only: input
use mod_constants, only: zone, zi, zzero, pi, h2ev
use mod_eigenvalue_occupancy, only: evalsv, nstsv
use mod_kpoint, only: nkptnr
MPI and BLACS/ScaLAPACK
use modmpi
use modscl
XS
use mod_xsgrids
use modxs, only: vkl0, occsv0, evalsv0, unitout, iqmtgamma
BSE
use modbse
Spectrum
use mod_symmetry, only: nsymcrys
Interface modules
use m_putgetbsemat
use m_genwgrid
use m_genfilname
use m_diagfull
use m_writeoscillator
```

```
use m_dhesolver
use m_hesolver
use m_setup_bse
use m_genexevec
use m_putgetexcitons
use m_makeoscistr
use m_makespectrum
use m_invertzmat
```

DESCRIPTION:

Solves the Bethe-Salpeter equation(BSE). The BSE is treated as equivalent effective eigenvalue problem(thanks to the spectral theorem that can be applied to the original BSE in the case of a statically screened Coulomb interaction). The effective BSE-Hamiltonian consists of three parts originating from different sources. It reads

$$H^{\mathrm{eff}} = H^{\mathrm{diag}} + 2H^{\mathrm{x}} + H^{\mathrm{c}},$$

where H^{diag} is the diagonal part stemming from the independent particle transitions, H^{x} denotes the exchange-term caused by the unscreened(bare) Coulomb interaction, whereas H^{c} accounts for the particle-hole correlations and is originating from the screened Coulomb interaction. For the purpose of describing independent particle transitions with the BSE only the diagonal term is referred to:

$$H^{\mathrm{eff}} = H^{\mathrm{diag}}.$$

By neglecting the correlation part in the effective Hamiltonian we arrive at the *random phase approximation* (RPA)

$$H^{\mathrm{eff}} = H^{\mathrm{diag}} + 2H^{\mathrm{x}}.$$

Investigations on the spin-structure of the BSE-Hamiltonian show that there are tow channels, namely the *singlet*-channel as solution to the Hamiltonian

$$H^{\mathrm{eff}} = H^{\mathrm{diag}} + 2H^{\mathrm{x}} + H^{\mathrm{c}}$$

and a *triplet* channel with the exchange-part being absent.

$$H^{\mathrm{eff}} = H^{\mathrm{diag}} + H^{\mathrm{c}}.$$

The equation of the eigenvalue problem is given by

$$\sum_{v'c'\mathbf{k}'} H^{\mathrm{eff}}_{vc\mathbf{k},v'c'\mathbf{k}'} A^{\lambda}_{v'c'\mathbf{k}'} = \varepsilon_{\lambda} A^{\lambda}_{vc\mathbf{k}}.$$

For the diagonalization of the Hamiltonian, a LAPACK-routine(`zheevx`) is invoked to obtain the eigenvalues ε_{λ} and eigenvectors $A^{\lambda}_{vc\mathbf{k}}$ (alternatively, a time-evolution method shall be implemented to obtain the macroscopic dielectric function directly). Consequently, the transition amplitudes t_{λ} are calculated according to

$$t^{i}_{\lambda} = \left| \sum_{vc\mathbf{k}} A^{\lambda}_{vc\mathbf{k}} \frac{p^{i}_{vc\mathbf{k}}}{\varepsilon_{c\mathbf{k}} - \varepsilon_{v\mathbf{k}}} \right|^2.$$

Here, the index i labels the polarization and the matrix elements $p^i_{vc\mathbf{k}}$ are the ones for the i-th component of the momentum operator in Cartesian coordinates. The macroscopic dielectric function(MDF) is obtained by the realation

$$\text{Im } \epsilon^i_{\text{M}}(\omega) = \frac{8\pi^2}{V} \sum_\lambda t^i_\lambda \delta(\omega - \varepsilon_\lambda + \Delta),$$

where ϵ^i_{M} is the MDF for the i-th polarization, V denotes the crystal volume and Δ is a constant shift of the conduction bands(scissors shift). The delta-function in the latter expression is convoluted with a(symmetrized) Lorentzian

$$\pi\delta(\omega - \omega_0) = \lim_{\eta\to 0}\left[\frac{\eta}{(\omega-\omega_0)^2+\eta^2} + \frac{\eta}{(-\omega-\omega_0)^2-\eta^2}\right] = \pi\delta(\omega-\omega_0) + \pi\delta(\omega+\omega_0)$$

which is true for $\omega \geq 0$ if $\omega_0 > 0$. In doing so, the analytic property $\text{Im}\epsilon_{\text{M}}(0) = 0$ is fulfilled. The broadening η in the latter expression is adjusted by the `broad` parameter. (All parts of the documentation written by S. Sagmeister are part of the author's PhD-thesis.)

REVISION HISTORY:

```
Created June 2008(S. Sagmeister)
Addition of explicit energy ranges for states below and above the Fermi
   level for the treatment of core excitations(using local orbitals).
   October 2010(Weine Olovsson)
Added possibility to compute off-diagonal optical components, Dec 2013(Stefan Kontur, STK)
Forked from bse.F90 and adapted for non-TDA and Q-dependent BSE. (Aurich)
```

2.0.64 setup_dmat (Source File: m_setup_dmat.f90)

INTERFACE:

```
subroutine setup_dmat(dmat)
```

INPUT/OUTPUT PARAMETERS:

```
Out:
  complex(8) :: dmat(hamsize,3)  ! Dipole operator matrix
```

DESCRIPTION:

The routine generates the resonant Dipole matrix elements $D^{\text{rr}}_{\alpha,j} = \langle u_\alpha \vec{k}_\alpha| - \hat{r}_j | o_\alpha \vec{k}_\alpha\rangle = i\frac{\langle u_\alpha \vec{k}_\alpha|\hat{p}_j|o_\alpha\vec{k}_\alpha\rangle}{\epsilon_{u_\alpha,\vec{k}_\alpha} - \epsilon_{o_alpha\vec{k}_\alpha}}$
Alpha is the combined index used in the BSE Hamiltonian.

REVISION HISTORY:

```
Created. (Aurich)
```

2.0.65 setup_dmat_dist (Source File: m_setup_dmat.f90)

INTERFACE:

```
subroutine setup_dmat_dist(dmat, binfo)
```

INPUT/OUTPUT PARAMETERS:

```
In:
  type(blacsinfo) :: binfo ! Info type for BLACS grid
In/Out:
  type(dzmat) :: dmat  ! 2D block cyclic distributed dipole operator matrix
```

DESCRIPTION:

The routine sets up the content of the local array of the 2d block cyclic distributed dipole operator matrix. Process 0 reads `PMATXS.OUT` for each `ik` record and send the data blockwise to the responsible processes.
$D^{\mathrm{rr}}_{\alpha,j} = \langle u_\alpha \vec{k}_\alpha | - \hat{r}_j | o_\alpha \vec{k}_\alpha \rangle = i \frac{\langle u_\alpha \vec{k}_\alpha | \hat{p}_j | o_\alpha \vec{k}_\alpha \rangle}{\epsilon_{u_\alpha,\vec{k}_\alpha} - \epsilon_{o_alpha\vec{k}_\alpha}}$
Alpha is the combined index used in the BSE Hamiltonian.

REVISION HISTORY:

```
Created. (Aurich)
```

2.0.66 builddmat (Source File: m_setup_dmat.f90)

INTERFACE:

```
function builddmat(ib, jb, occ, ediff, pmat)
```

INPUT/OUTPUT PARAMETERS:

```
In:
integer(4) :: ib, jb        ! Sub block size
real(8)    :: occ(ib)       ! Occupation factors
real(8)    :: ediff(ib)     ! Energy difference e_u-e_o
complex(8) :: pmat(ib, jb)  ! Puo(k) slice
Out:
complex(8) :: builddmat(ib,jb)  ! Sub block of D-mat
```

DESCRIPTION:

The function returns a sub block of the distributed Dipole operator matrix:
$D(i_g : i_g + i_b - 1, j_g : j_g + j_b - 1)$ where each entry is computed according to
$D(i, \mathrm{opt}) = F(i) i P(i, \mathrm{opt}) / E(i)$
E are the Kohn-Sham transition energies, F the occupation factors and P the momentum matrix elements. The index i correspond to the combined index α and the index opt refers to a Cartesian direction.

So with $\alpha = \{u_\alpha, o_\alpha, \vec{k}_\alpha\}$:
$F_\alpha = \sqrt{\left| f_{\vec{k}_\alpha o_\alpha} - f_{\vec{k}_\alpha u_\alpha} \right|}$, $E_\alpha = \epsilon_{\vec{k}_\alpha u_\alpha} - \epsilon_{\vec{k}_\alpha o_\alpha}$, $P_{\alpha,\mathrm{opt}} = P^{\mathrm{opt}}_{u_\alpha o_\alpha \vec{k}_\alpha}$
REVISION HISTORY:

```
Created 2016 (Aurich)
```

2.0.67 getpemat (Source File: getpemat.F90)

INTERFACE:

```
subroutine getpemat(iq, ik, pfilnam, efilnam, m12, m34, p12, p34)
```

USES:

```
    use modinput, only: input
    use mod_constants, only: fourpi, zzero
    use modxs, only: ngq, vkl0,&
                   & istl1, istl2, istl3, istl4,&
                   & istu1, istu2, istu3, istu4,&
                   & nst1, nst2, nst3, nst4,&
                   & deou, docc12, deuo, docc21,&
                   & tscreen, xiou, xiuo, sptclg
#ifdef TETRA
    use modtetra
#endif
    use m_getpmat
    use m_getemat
```

DESCRIPTION:

For a given k and q point this routine reads the momentum matrix (Gamma point) and the plane wave matrix elements for the band combinations 12 and optionally 21 from file. The momentum matrix elements get renormalized by the KS transiton energies. Both momentum matrix elements and plane wave matrix elements will be multiplied with the square root of the coulomb potential.
REVISION HISTORY:

```
Added to documentations schema. (Aurich)
```

2.0.68 screen (Source File: screen.F90)

INTERFACE:

```
subroutine screen
```

USES:

```
use modmpi
use modxs, only: nwdf, unitout
use m_genfilname
use m_filedel
```

DESCRIPTION:

This is a wrapper subroutine that backs up the number of ω points and calls the `df` subroutine. Afterwards it restores them. It also sets the output file extensions for screening related quantities.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

2.0.69 writepmatxs (Source File: writepmatxs.F90)

INTERFACE:

```
subroutine writepmatxs
```

USES:

```
use modinput, only: input
use modmpi, only: procs, rank, firstofset, lastofset, barrier
use mod_misc, only: task, filext
use mod_kpoint, only: nkpt, vkl
use mod_Gkvector, only: ngkmax, vgkl, ngk, gkc, tpgkc, sfacgk, igkig, vgkc
use mod_APW_LO, only: apwordmax, nlotot, nlomax, lolmax
use mod_muffin_tin, only: lmmaxapw
use mod_atoms, only: natmtot
use mod_eigensystem, only: nmatmax
use mod_eigenvalue_occupancy, only: nstfv, nstsv
use modxas, only: ncg
use modxs, only: tscreen, fnpmat, fnpmat_t, kpari,&
                & kparf, hybridhf, ripaa, ripalo,&
                & riploa, riplolo, apwcmt, locmt,&
                & unitout, iqmtgamma
use m_putpmat
use m_genfilname
```

DESCRIPTION:

Calculates the momentum matrix elements using routine `genpmat` and writes them to direct access file `PMAT.OUT`, `PMAT_XS.OUT` or `PMAT_SCR.OUT` depending on the context of the execution.

REVISION HISTORY:

```
Created 2006 (S. Sagmeister)
Modifications, August 2010 (S. Sagmeister)
Removed dead code, added comments, 2017 (Aurich)
```

2.0.70 fermisurf_dx (Source File: fermisurf_dx.f90)

INTERFACE:

```
Subroutine fermisurf_dx
```

USES:

```
Use modmain
```

DESCRIPTION:

Add-on to the `fermisurf` routine. Generates input files for the visualization of the Fermi surface in the first Brillouin zone with OpenDX. Prerequisite is a previous run of `fermisurf` for a $2 \times 2 \times 2$ supercell of reciprocal asymmetric units. It generates a `FERMI.dx` and a `FERMI_BZ.dx` file where the data for the visualization is stored together with the clipping planes for the construction of the first Brillouin zone. The OpenDX input is complete together with the template `FERMI.net` and the macro `clipvmacro.net`.

REVISION HISTORY:

```
Created Jan 2005 (Sagmeister)
Revised, May 2008 (Sagmeister)
```

2.0.71 seceqn (Source File: residualvector.F90)

Subroutine residualvectors (n, iunconverged, h, s, evalfv, r, rnorms) Use modmain, Only: nmatmax, zone, zzero

INPUT/OUTPUT PARAMETERS:

DESCRIPTION:

the residual is

$$|\mathbf{R}\,(|\mathbf{A}^{ap}\rangle, E^{ap})\rangle = (\mathbf{H} - E^{ap}\mathbf{S})|\mathbf{A}^{ap}\rangle$$

REVISION HISTORY:

```
Created March 2004 (JKD)
```

2.0.72 mixpulay (Source File: mixpulay.f90)

INTERFACE:

```
Subroutine mixpulay (iscl, n, maxsd, nu, mu, f, d)
```

INPUT/OUTPUT PARAMETERS:

```
iscl  : self-consistent loop number (in,integer)
n     : vector length (in,integer)
maxsd : maximum subspace dimension (in,integer)
nu    : current output vector as well as the next input vector of the
        self-consistent loop (inout,real(n))
mu    : used internally (inout,real(n,maxsd))
f     : used internally (inout,real(n,maxsd))
d     : RMS difference between old and new output vectors (out,real)
```

DESCRIPTION:

Pulay's mixing scheme which uses direct inversion in the iterative subspace (DIIS). See *Chem. Phys. Lett.* **73**, 393 (1980).

REVISION HISTORY:

```
Created October 2008 (S. Suehara, NIMS)
Modified, October 2008 (JKD)
```

2.0.73 mixadapt (Source File: mixadapt.f90)

INTERFACE:

```
Subroutine mixadapt (iscl, beta0, betainc, betadec, n, nu, mu, beta, f, &
& d)
```

INPUT/OUTPUT PARAMETERS:

```
iscl    : self-consistent loop number (in,integer)
beta0   : initial value for mixing parameter (in,real)
betainc : mixing parameter increase (in,real)
betadec : mixing parameter decrease (in,real)
n       : vector length (in,integer)
nu      : current output vector as well as the next input vector of the
          self-consistent loop (inout,real(n))
mu      : used internally (inout,real(n))
beta    : used internally (inout,real(n))
f       : used internally (inout,real(n))
d       : RMS difference between old and new output vectors (out,real)
```

DESCRIPTION:

Given the input vector μ^i and output vector ν^i of the ith self-consistent loop, this routine generates the next input vector to the loop using an adaptive mixing scheme. The jth component of the output vector is mixed with a fraction of the same component of the input vector:

$$\mu_j^{i+1} = \beta_j^i \nu_j^i + (1 - \beta_j^i)\mu_j^i,$$

where β_j^i is set to β_0 at initialisation and increased by scaling with $\beta_{\rm inc}$ (> 1) if $f_j^i \equiv \nu_j^i - \mu_j^i$ does not change sign between loops. If f_j^i does change sign, then β_j^i is scaled by $\beta_{\rm dec}$ (< 1). Note that the array `nu` serves for both input and output, and the arrays `mu`, `beta` and `f` are used internally and should not be changed between calls. The routine is initialised at the first iteration and is thread-safe so long as each thread has its own independent work array. Complex arrays may be passed as real arrays with n doubled.

REVISION HISTORY:

```
Created March 2003 (JKD)
Modified, September 2008 (JKD)
```

2.0.74 mixmsec (Source File: mixmsec.f90)

INTERFACE:

```
Subroutine mixmsec (iscl, potential, residualnorm, n)
```

INPUT/OUTPUT PARAMETERS:

```
      Use modinput
  iscl     : self-consistent loop number (in,integer)
  potential : potential coefitients packed in one array
  residualnorm: measure for convergence
  n : length of mixing vector
 config params:
      Use mod_potential_and_density, Only:
      Use mod_charge_and_moment, Only: chgir, chgmttot, chgtot
  persistent arrays and create/desdruct functions
      Use modmixermsec, Only: residual, last_outputp, work2, work3, &
     & initmixermsec, freearraysmixermsec, noldstepsmax, &
     & noldstepsin_file, noldsteps, qmx, dmix, dmixout, TCharge, &
     & SCharge, splane, tplane, qmx_input, qtot
```

2.0.75 sumrule (Source File: sumrule.f90)

INTERFACE:

```
Subroutine sumrule (dynq)
```

INPUT/OUTPUT PARAMETERS:

```
      Use modinput
    dynq : dynamical matrices on q-point set (in,real(3*natmtot,3*natmtot,nqpt))
```

DESCRIPTION:

Applies the same correction to all the dynamical matrices such that the matrix for $\mathbf{q} = 0$ satisfies the acoustic sum rule. In other words, the matrices are updated with

$$D_{ij}^{\mathbf{q}} \rightarrow D_{ij}^{\mathbf{q}} - \sum_{k=1}^{3} \omega_k^0 v_{k;i}^0 v_{k;j}^0$$

for all $\mathbf{q}$, where ω_k^0 is the kth eigenvalue of the $\mathbf{q} = 0$ dynamical matrix and $v_{k;i}^0$ the ith component of its eigenvector. The eigenvalues are assumed to be arranged in ascending order. This ensures that the $\mathbf{q} = 0$ dynamical matrix has 3 zero eigenvalues, which the uncorrected matrix may not have due to the finite exchange-correlation grid.

REVISION HISTORY:

```
Created May 2005 (JKD)
```

2.0.76 writedynamicalmatrices (Source File: writedynamicalmatrices.F90)

INTERFACE:

```
subroutine writedynamicalmatrices(fname,dynmq)
```

USES:

```
use modinput
Use mod_atoms
use mod_qpoint
use mod_constants
```

DESCRIPTION:

Write out the dynamical matrices to file `DYNMAT.OUT`.

REVISION HISTORY:

```
Created February 2010 (Sagmeister)
```

2.0.77 reformatdynamicalmatrices (Source File: reformatdynamicalmatrices.F90)

INTERFACE:

```
subroutine reformatdynamicalmatrices
```

USES:

```
use modinput
Use mod_atoms
use mod_qpoint
use mod_constants
```

DESCRIPTION:

Collecting pieces of the dynamical matrices and assembling them in a nicer way, such that 3×3 matrices are displayed for each combination of atoms and each $\mathbf{q}$ point.

REVISION HISTORY:

```
Created February 2010 (Sagmeister)
```

!ROUTINE: shg **INTERFACE:**

```
subroutine shg(a,b,c)
```

USES:

```
use modinput
use modmain
use modmpi
```

DESCRIPTION:

Calculates susceptibility tensor for non-linear optical second-harmonic generation (SHG). The terms (ztm) are numbered according to Eqs. (49)-(51) of the article *Physica Scripta* **T109**, 128 (2004). Other good references are *Phys. Rev. B* **48**, 11705 (1993) and *Phys. Rev. B* **53**, 10751 (1996).

REVISION HISTORY:

```
Rewrote earlier version, June 2010 (Sharma)
Modified, April 2013 (DIN)
```

2.0.78 genpmat (Source File: genpmat.f90)

INTERFACE:

```
Subroutine genpmat (ngp, igpig, vgpc, apwalm, evecfv, evecsv, pmat)
```

USES:

```
  Use modinput
  Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
ngp    : number of G+p-vectors (in,integer)
igpig  : index from G+p-vectors to G-vectors (in,integer(ngkmax))
vgpc   : G+p-vectors in Cartesian coordinates (in,real(3,ngkmax))
apwalm : APW matching coefficients
         (in,complex(ngkmax,apwordmax,lmmaxapw,natmtot))
evecfv : first-variational eigenvector (in,complex(nmatmax,nstfv))
evecsv : second-variational eigenvectors (in,complex(nstsv,nstsv))
pmat   : momentum matrix elements (out,complex(3,nstsv,nstsv))
```

DESCRIPTION:

Calculates the momentum matrix elements

$$p_{ij} = \langle \Psi_{i,\mathbf{k}} | - i\nabla | \Psi_{j,\mathbf{k}} \rangle.$$

REVISION HISTORY:

```
Created November 2003 (Sharma)
Fixed bug found by Juergen Spitaler, September 2006 (JKD)
```

!ROUTINE: writepmat **INTERFACE:**

```
Subroutine writepmat
```

USES:

```
Use modinput
Use modmain
Use modmpi
Use modxs
```

DESCRIPTION:

Calculates the momentum matrix elements using routine `genpmat` and writes them to direct access file `PMAT.OUT`, `PMAT_XS.OUT` or `PMAT_SCR.OUT` depending on the context of the execution.

REVISION HISTORY:

```
Created 2006 (S. Sagmeister)
Modifications, August 2010 (S. Sagmeister)
Readjusted and re-parallelized, April 2013 (DIN)
```

2.0.79 r3taxi (Source File: r3taxi.f90)

INTERFACE:

```
Real (8) Function r3taxi (x, y)
```

INPUT/OUTPUT PARAMETERS:

```
x : input vector 1 (in,real(3))
y : input vector 2 (in,real(3))
```

DESCRIPTION:

Returns the taxi-cab distance between two real 3-vectors: $d = |x_1 - y_1| + |x_2 - y_2| + |x_3 - y_3|$.

REVISION HISTORY:

```
Created March 2006 (JKD)
```

2.0.80 gauntyry (Source File: gauntyry.f90)

INTERFACE:

```
Complex (8) Function gauntyry (l1, l2, l3, m1, m2, m3)
```

INPUT/OUTPUT PARAMETERS:

```
l1, l2, l3 : angular momentum quantum numbers (in,integer)
m1, m2, m3 : magnetic quantum numbers (in,integer)
```

DESCRIPTION:

Returns the complex Gaunt-like coefficient given by $\langle Y^{l_1}_{m_1}|R^{l_2}_{m_2}|Y^{l_3}_{m_3}\rangle$, where Y_{lm} and R_{lm} are the complex and real spherical harmonics, respectively. Suitable for l_i less than 50. See routine `genrlm`.

REVISION HISTORY:

```
Created November 2002 (JKD)
```

2.0.81 stheta_fd (Source File: stheta_fd.f90)

INTERFACE:

```
Real (8) Function stheta_fd (x)
```

INPUT/OUTPUT PARAMETERS:

```
x : real argument (in,real)
```

DESCRIPTION:

Returns the Fermi-Dirac approximation to the Heaviside step function

$$\tilde{\Theta}(x) = \frac{1}{1+e^{-x}}.$$

REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.82 sortidx (Source File: sortidx.f90)

INTERFACE:

```
Subroutine sortidx (n, a, idx)
```

INPUT/OUTPUT PARAMETERS:

```
n   : number of elements in array (in,integer)
idx : permutation index (out,integer(n))
a   : real array (in,real(n))
```

DESCRIPTION:

Finds the permutation index `idx` which sorts the real array `a` into ascending order. No sorting of the array `a` itself is performed. Uses the heapsort algorthim.

REVISION HISTORY:

```
Created October 2002 (JKD)
Included tolerance eps, April 2006 (JKD)
```

2.0.83 rtozflm (Source File: rtozflm.f90)

INTERFACE:

```
Subroutine rtozflm (lmax, rflm, zflm)
```

INPUT/OUTPUT PARAMETERS:

```
lmax : maximum angular momentum (in,integer)
rflm : coefficients of real spherical harmonic expansion
       (in,real((lmax+1)**2)))
zflm : coefficients of complex spherical harmonic expansion
       (out,complex((lmax+1)**2)))
```

DESCRIPTION:

Converts a real function, r_{lm}, expanded in terms of real spherical harmonics into a complex spherical harmonic expansion, z_{lm}:

$$z_{lm} = \begin{cases} \frac{1}{\sqrt{2}}(r_{lm} + i(-1)^m r_{l-m}) & m > 0 \\ \frac{1}{\sqrt{2}}((-1)^m r_{l-m} - i r_{lm}) & m < 0 \\ r_{lm} & m = 0 \end{cases} .$$

See routine `genrlm`.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.84 zmatinp (Source File: zmatinp.f90)

INTERFACE:

```
Subroutine zmatinp (tapp, n, alpha, x, y, v, a)
```

INPUT/OUTPUT PARAMETERS:

```
tapp  : .true. if the matrix is to be applied to the input vector v,
        .false. if the full matrix is to be calculated (in,logical)
n     : length of vectors (in,integer)
alpha : complex constant (in,complex)
x     : first input vector (in,complex(n))
y     : second input vector (in,complex(n))
v     : input vector to which matrix is applied if tapp is .true., otherwise
        not referenced (in,complex(n))
a     : matrix applied to v if tapp is .true., otherwise the full matrix in
        packed form (inout,complex(n+(n-1)*n/2))
```

DESCRIPTION:

Performs the rank-2 operation

$$A_{ij} \rightarrow \alpha \mathbf{x}_i^* \mathbf{y}_j + \alpha^* \mathbf{y}_i^* \mathbf{x}_j + A_{ij},$$

where A is stored in packed form. This is similar to the `BLAS` routine `zhpr2`, except that here a matrix of inner products is formed instead of an outer product of vectors. If `tapp` is `.true.` then the matrix is applied to an input vector, rather than calculated explicitly.

REVISION HISTORY:

```
Created June 2003 (JKD)
```

2.0.85 factr (Source File: factr.f90)

INTERFACE:

```
Real (8) Function factr (n, d)
```

INPUT/OUTPUT PARAMETERS:

```
n : numerator (in,integer)
d : denominator (in,integer)
```

DESCRIPTION:

Returns the ratio $n!/d!$ for $n, d \geq 0$. Performs no under- or overflow checking.

REVISION HISTORY:

```
Created October 2002 (JKD)
```

2.0.86 stheta_mp (Source File: stheta_mp.f90)

INTERFACE:

```
Real (8) Function stheta_mp (n, x)
```

INPUT/OUTPUT PARAMETERS:

```
n : order (in,integer)
x : real argument (in,real)
```

DESCRIPTION:

Returns the smooth approximation to the Heaviside step function of order N given by Methfessel and Paxton, *Phys. Rev. B* **40**, 3616 (1989),

$$\tilde{\Theta}(x) = 1 - S_N(x)$$

where

$$S_N(x) = S_0(x) + \sum_{i=1}^{N} \frac{(-1)^i}{i! 4^n \sqrt{\pi}} H_{2i-1}(x) e^{-x^2},$$
$$S_0(x) = \frac{1}{2}(1 - \mathrm{erf}(x))$$

and H_j is the jth-order Hermite polynomial. This procedure is numerically stable and accurate to near machine precision for $N \leq 10$.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.87 lopzflm (Source File: lopzflm.f90)

INTERFACE:

```
Subroutine lopzflm (lmax, zflm, ld, zlflm)
```

INPUT/OUTPUT PARAMETERS:

```
lmax  : maximum angular momentum (in,integer)
zflm  : coefficients of input spherical harmonic expansion
        (in,complex((lmax+1)**2))
ld    : leading dimension (in,integer)
zlflm : coefficients of output spherical harmonic expansion
        (out,complex(ld,3))
```

DESCRIPTION:

Applies the angular momentum operator $\mathbf{L}$ to a function expanded in terms of complex spherical harmonics. This makes use of the identities

$$(L_x + iL_y)Y_{lm}(\theta,\phi) = \sqrt{(l-m)(l+m+1)}Y_{lm+1}(\theta,\phi)$$
$$(L_x - iL_y)Y_{lm}(\theta,\phi) = \sqrt{(l+m)(l-m+1)}Y_{lm-1}(\theta,\phi)$$
$$L_z Y_{lm}(\theta,\phi) = mY_{lm}(\theta,\phi).$$

REVISION HISTORY:

```
Created March 2004 (JKD)
```

2.0.88 flushifc (Source File: flushifc.f90)

INTERFACE:

```
Subroutine flushifc (fnum)
```

INPUT/OUTPUT PARAMETERS:

```
  Use modinput
fnum : unit specifier for file (in,integer)
```

DESCRIPTION:

Interface to the Fortran `flush` statement. Some compilers do not support the `flush` command, which is very useful for keeping small formatted files up-to-date on the disk. The routine implimented below is a machine-independent emulation of `flush`, but may be replaced with the intrinsic command if preferred.

REVISION HISTORY:

```
Created September 2002 (JKD)
```

2.0.89 clebgor (Source File: clebgor.f90)

INTERFACE:

```
Real (8) Function clebgor (j1, j2, j3, m1, m2, m3)
```

INPUT/OUTPUT PARAMETERS:

```
j1, j2, j3 : angular momentum quantum numbers (in,integer)
m1, m2, m3 : magnetic quantum numbers (in,integer)
```

DESCRIPTION:

Returns the Clebsch-Gordan coefficients using the Wigner $3j$-symbols

$$C(J_1J_2J_3|m_1m_2m_3) = (-1)^{J_1-J_2+m_3}\sqrt{2J_3+1}\begin{pmatrix} J_1 & J_2 & J_3 \\ m_1 & m_2 & -m_3 \end{pmatrix}.$$

Suitable for $J_i \leq 50$.

REVISION HISTORY:

```
Created September 2003 (JKD)
```

2.0.90 polynom (Source File: polynom.f90)

INTERFACE:

```
Real (8) Function polynom (m, np, xa, ya, c, x)
```

INPUT/OUTPUT PARAMETERS:

```
m  : order of derivative (in,integer)
np : number of points to fit (in,integer)
xa : abscissa array (in,real(np))
ya : ordinate array (in,real(np))
c  : work array (out,real(np))
x  : evaluation abscissa (in,real)
```

DESCRIPTION:

Fits a polynomial of order $n_p - 1$ to a set of n_p points. If $m \geq 0$ the function returns the mth derviative of the polynomial at x, while for $m < 0$ the integral of the polynomial from the first point in the array to x is returned.

REVISION HISTORY:

```
Created October 2002 (JKD)
```

2.0.91 sphcrd (Source File: sphcrd.f90)

INTERFACE:

```
Subroutine sphcrd (v, r, tp)
```

INPUT/OUTPUT PARAMETERS:

```
v  : input vector (in,real(3))
r  : length of v (out,real)
tp : (theta, phi) coordinates (out,real(2))
```

DESCRIPTION:

Returns the spherical coordinates (r, θ, ϕ) of a vector

$$\mathbf{v} = (r\sin(\theta)\cos(\phi), r\sin(\theta)\sin(\phi), r\cos(\theta)).$$

REVISION HISTORY:

```
Created October 2002 (JKD)
```

2.0.92 sbessel (Source File: sbessel.f90)

INTERFACE:

```
Subroutine sbessel (lmax, x, jl)
```

INPUT/OUTPUT PARAMETERS:

```
lmax : maximum order of Bessel function (in,integer)
x    : real argument (in,real)
jl   : array of returned values (out,real(0:lmax))
```

DESCRIPTION:

Computes the spherical Bessel functions of the first kind, $j_l(x)$, for argument x and $l = 0, 1, \ldots, l_{\max}$. The recursion relation

$$j_{l+1}(x) = \frac{2l+1}{x} j_l(x) - j_{l-1}(x)$$

is used either downwards for $x < l$ or upwards for $x \geq l$. For $x \ll 1$ the asymtotic form is used

$$j_l(x) \approx \frac{x^l}{(2l+1)!!}.$$

This procedure is numerically stable and accurate to near machine precision for $l \leq 50$.

REVISION HISTORY:

```
Created January 2003 (JKD)
Modified to return an array of values, October 2004 (JKD)
Improved stability, August 2006 (JKD)
```

2.0.93 stheta_sq (Source File: stheta_sq.f90)

INTERFACE:

```
Real (8) Function stheta_sq (x)
```

INPUT/OUTPUT PARAMETERS:

```
x : real argument (in,real)
```

DESCRIPTION:

Returns the Heaviside step function corresponding to the square-wave pulse approximation to the Dirac delta function

$$\tilde{\Theta}(x) = \begin{cases} 0 & x \leq -1/2 \\ x + 1/2 & -1/2 < x < 1/2 \\ 1 & x \geq 1 \end{cases}$$

REVISION HISTORY:

```
Created July 2008 (JKD)
```

2.0.94 ztorflm (Source File: ztorflm.f90)

INTERFACE:

```
Subroutine ztorflm (lmax, zflm, rflm)
```

INPUT/OUTPUT PARAMETERS:

```
lmax : maximum angular momentum (in,integer)
zflm : coefficients of complex spherical harmonic expansion
       (in,complex((lmax+1)**2)))
rflm : coefficients of real spherical harmonic expansion
       (out,real((lmax+1)**2)))
```

DESCRIPTION:

Converts a real function, z_{lm}, expanded in terms of complex spherical harmonics into a real spherical harmonic expansion, r_{lm}:

$$r_{lm} = \begin{cases} \frac{1}{\sqrt{2}}\Re(z_{lm} + (-1)^m z_{l-m}) & m > 0 \\ \frac{1}{\sqrt{2}}\Im(-z_{lm} + (-1)^m z_{l-m}) & m < 0 \\ \Re(z_{lm}) & m = 0 \end{cases} .$$

See routine `genrlm`.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.95 rotaxang (Source File: rotaxang.f90)

INTERFACE:

```
Subroutine rotaxang (eps, rot, det, v, th)
```

INPUT/OUTPUT PARAMETERS:

```
eps : zero vector tolerance (in,real)
rot : rotation matrix (in,real(3,3))
det : matrix determinant (out,real)
v   : normalised axis vector (out,real(3))
th  : rotation angle (out,real)
```

DESCRIPTION:

Given a rotation matrix

$$R(\hat{\mathbf{v}}, \theta) = \begin{pmatrix} \cos\theta + x^2(1-\cos\theta) & xy(1-\cos\theta) + z\sin\theta & xz(1-\cos\theta) - y\sin\theta \\ xy(1-\cos\theta) - z\sin\theta & \cos\theta + y^2(1-\cos\theta) & yz(1-\cos\theta) + x\sin\theta \\ xz(1-\cos\theta) + y\sin\theta & yz(1-\cos\theta) - x\sin\theta & \cos\theta + z^2(1-\cos\theta) \end{pmatrix},$$

this routine determines the axis of rotation $\hat{\mathbf{v}}$ and the angle of rotation θ. If R corresponds to an improper rotation then only the proper part is used and `det` is set to -1.

REVISION HISTORY:

```
Created Decmeber 2006 (JKD)
Changed "intent(inout)" to "intent(in)" for argument "rot", 2009 (Sagmeister)
```

2.0.96 gaunt (Source File: gaunt.f90)

INTERFACE:

```
Real (8) Function gaunt (l1, l2, l3, m1, m2, m3)
```

INPUT/OUTPUT PARAMETERS:

```
l1, l2, l3 : angular momentum quantum numbers (in,integer)
m1, m2, m3 : magnetic quantum numbers (in,integer)
```

DESCRIPTION:

Returns the Gaunt coefficient given by

$$\langle Y^{l_1}_{m_1}|Y^{l_2}_{m_2}|Y^{l_3}_{m_3}\rangle = (-1)^{m_1}\left[\frac{(2l_1+1)(2l_2+1)(2l_3+1)}{4\pi}\right]^{\frac{1}{2}}\begin{pmatrix} l_1 & l_2 & l_3 \\ 0 & 0 & 0\end{pmatrix}\begin{pmatrix} l_1 & l_2 & l_3 \\ -m_1 & m_2 & m_3\end{pmatrix}.$$

Suitable for l_i less than 50.

REVISION HISTORY:

```
Created November 2002 (JKD)
```

2.0.97 fderiv (Source File: fderiv.f90)

INTERFACE:

```
Subroutine fderiv (m, n, x, f, g, cf)
```

INPUT/OUTPUT PARAMETERS:

```
m  : order of derivative (in,integer)
n  : number of points (in,integer)
x  : abscissa array (in,real(n))
f  : function array (in,real(n))
g  : (anti-)derivative of f (out,real(n))
cf : spline coefficients (out,real(3,n))
```

DESCRIPTION:

Given function f defined on a set of points x_i then if $m \geq 0$ this routine computes the mth derivative of f at each point. If $m < 0$ the anti-derivative of f given by

$$g(x_i) = \int_{x_1}^{x_i} f(x)\, dx$$

is calculated by fitting the function to a clamped cubic spline. See routine `spline`.

REVISION HISTORY:

```
Created May 2002 (JKD)
```

2.0.98 sdelta_fd (Source File: sdelta_fd.f90)

INTERFACE:

```
Real (8) Function sdelta_fd (x)
```

INPUT/OUTPUT PARAMETERS:

```
x : real argument (in,real)
```

DESCRIPTION:

Returns the Fermi-Dirac approximation to the Dirac delta function

$$\tilde{\delta}(x) = \frac{e^{-x}}{(1+e^{-x})^2}.$$

REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.99 erf (Source File: erf.f90)

INTERFACE:

```
Real (8) Function erf (x)
```

INPUT/OUTPUT PARAMETERS:

```
x : real argument (in,real)
```

DESCRIPTION:

Returns the error function $\mathrm{erf}(x)$ using a rational function approximation. This procedure is numerically stable and accurate to near machine precision.

REVISION HISTORY:

```
Modified version of a NSWC routine, April 2003 (JKD)
```

2.0.100 axangsu2 (Source File: axangsu2.f90)

Subroutine axangsu2 (v, th, su2) **INPUT/OUTPUT PARAMETERS:**

```
v   : rotation axis vector (in,real(3))
th  : rotation angle (in,real)
su2 : SU(2) representation of rotation (out,complex(2,2))
```

DESCRIPTION:

Finds the complex SU(2) representation of a rotation defined by an axis vector $\hat{\mathbf{v}}$ and angle θ. The spinor rotation matrix is given explicitly by

$$R^{1/2}(\hat{\mathbf{v}},\theta) = I\cos\frac{\theta}{2} + i(\hat{\mathbf{v}}\cdot\vec{\sigma})\sin\frac{\theta}{2}.$$

REVISION HISTORY:

```
Created August 2007 (JKD)
Reversed rotation direction, February 2008 (L. Nordstrom)
```

2.0.101 r3dot (Source File: r3dot.f90)

INTERFACE:

```
Real (8) Function r3dot (x, y)
```

INPUT/OUTPUT PARAMETERS:

```
x : input vector 1 (in,real(3))
y : input vector 2 (in,real(3))
```

DESCRIPTION:

Returns the dot-product of two real 3-vectors.

REVISION HISTORY:

```
Created January 2003 (JKD)
```

2.0.102 z2mctm (Source File: z2mctm.f90)

INTERFACE:

```
Subroutine z2mctm (a, b, c)
```

INPUT/OUTPUT PARAMETERS:

```
a   : input matrix 1 (in,complex(2,2))
b   : input matrix 2 (in,complex(2,2))
c   : output matrix (out,complex(2,2))
```

DESCRIPTION:

Multiplies the conjugate transpose of one complex 2×2 matrix with another. Note that the output matrix cannot be one of the input matrices.

REVISION HISTORY:

```
Created October 2007 (JKD)
```

2.0.103 z2mm (Source File: z2mm.f90)

INTERFACE:

```
Subroutine z2mm (a, b, c)
```

INPUT/OUTPUT PARAMETERS:

```
a   : input matrix 1 (in,complex(2,2))
b   : input matrix 2 (in,complex(2,2))
c   : output matrix (out,complex(2,2))
```

DESCRIPTION:

Multiplies two complex 2×2 matrices. Note that the output matrix cannot be one of the input matrices.

REVISION HISTORY:

```
Created October 2007 (JKD)
```

2.0.104 rlkhint (Source File: rlkhint.f90)

INTERFACE:

```
Subroutine rlkhint (m, l, e, nr, r, vr, nn, p0p, q0p, p0, p1, q0, q1)
```

INPUT/OUTPUT PARAMETERS:

```
m   : order of energy derivative (in,integer)
l   : angular momentum quantum number (in,integer)
e   : energy (in,real)
nr  : number of radial mesh points (in,integer)
r   : radial mesh (in,real(nr))
vr  : potential on radial mesh (in,real(nr))
nn  : number of nodes (out,integer)
p0p : m-1 th energy derivative of P (in,real(nr))
p0  : m th energy derivative of P (out,real(nr))
p1  : radial derivative of p0 (out,real(nr))
q0  : m th energy derivative of Q (out,real(nr))
q1  : radial derivative of q0 (out,real(nr))
```

DESCRIPTION:

Integrates the mth energy derivative of the scalar relativistic radial Schrödinger equation from $r = 0$ outwards. The kinetic energy operator is based on IORA. **REVISION HISTORY:**

```
Created July 2013 (Andris)
```

2.0.105 sdelta_mp (Source File: sdelta_mp.f90)

INTERFACE:

```
Real (8) Function sdelta_mp (n, x)
```

INPUT/OUTPUT PARAMETERS:

```
n : order (in,integer)
x : real argument (in,real)
```

DESCRIPTION:

Returns the smooth approximation to the Dirac delta function of order N given by Methfessel and Paxton, *Phys. Rev. B* **40**, 3616 (1989),

$$\tilde{\delta}(x) = \sum_{i=0}^{N} \frac{(-1)^i}{i! 4^n \sqrt{\pi}} H_{2i}(x) e^{-x^2},$$

where H_j is the jth-order Hermite polynomial. This function has the property

$$\int_{-\infty}^{\infty} \tilde{\delta}(x) P(x) = P(0),$$

where $P(x)$ is any polynomial of degree $2N + 1$ or less. The case $N = 0$ corresponds to Gaussian smearing. This procedure is numerically stable and accurate to near machine precision for $N \leq 10$.
REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.106 rschrodapp (Source File: rschrodapp.f90)

INTERFACE:

```
Subroutine rschrodapp (l, nr, r, vr, p0, q0, q1, hp0)
use modinput
```

INPUT/OUTPUT PARAMETERS:

```
l   : angular momentum quantum number (in,integer)
nr  : number of radial mesh points (in,integer)
r   : radial mesh (in,real(nr))
vr  : potential on radial mesh (in,real(nr))
p0  : m th energy derivative of P (in,real(nr))
q0  : m th energy derivative of Q (in,real(nr))
q1  : radial derivative of q0 (in,real(nr))
hp0 : H applied to P (out,real(nr))
```

DESCRIPTION:

Applies the scalar relativistic radial Hamiltonian, H, to a radial wavefunction, P_l. This is an approximation since we assume P_l is a scalar wavefunction, normalisable to unity. A Hamiltonian which satisfies $HP_l = EP_l$ is given implicitly by

$$HP_l = \left[\frac{l(l+1)}{2Mr^2} + V\right] P_l - \frac{1}{r}Q_l - \frac{d}{dr}Q_l,$$

where V is the external potential, $M = 1 - V/2c^2$ and Q_l is obtained from integrating the coupled scalar relativistic equations. See the routine `rschrodint` for further details.

REVISION HISTORY:

```
Created October 2003 (JKD)
```

2.0.107 spline (Source File: spline.f90)

INTERFACE:

```
Subroutine spline (n, x, ld, f, cf)
```

INPUT/OUTPUT PARAMETERS:

```
n  : number of points (in,integer)
x  : abscissa array (in,real(n))
ld : leading dimension (in,integer)
f  : input data array (in,real(ld,n))
cf : cubic spline coefficients (out,real(3,n))
```

DESCRIPTION:

Calculates the coefficients of a cubic spline fitted to input data. In other words, given a set of data points f_i defined at x_i, where $i = 1 \ldots n$, the coefficients c_j^i are determined such that

$$y_i(x) = f_i + c_1^i(x - x_i) + c_2^i(x - x_i)^2 + c_3^i(x - x_i)^3,$$

is the interpolating function for $x \in [x_i, x_{i+1})$. This is done by determining the end-point coefficients c_2^1 and c_2^n from the first and last three points, and then solving the tridiagonal system

$$d_{i-1}c_2^{i-1} + 2(d_{i-1} + d_i)c_2^i + d_i c_2^{i+1} = 3\left(\frac{f_{i+1} - f_i}{d_i} - \frac{f_i - f_{i-1}}{d_{i-1}}\right),$$

where $d_i = x_{i+1} - x_i$, for the intermediate coefficients.
REVISION HISTORY:

```
Created October 2004 (JKD)
Improved speed and accuracy, April 2006 (JKD)
Optimisations and improved end-point coefficients, February 2008 (JKD)
```

2.0.108 gcd (Source File: gcd.f90)

INTERFACE:

```
Integer Function gcd (x, y)
```

INPUT/OUTPUT PARAMETERS:

```
x : first integer (in,integer)
y : second integer (in,integer)
```

DESCRIPTION:

Computes the greatest common divisor (GCD) of two integers using Euclid's algorithm.
REVISION HISTORY:

```
Created September 2004 (JKD)
```

2.0.109 i3minv (Source File: i3minv.f90)

INTERFACE:

```
Subroutine i3minv (a, b)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix (in,integer(3,3))
b : output matrix (in,integer(3,3))
```

DESCRIPTION:

Computes the inverse of a integer 3×3 matrix: $B = A^{-1}$.
REVISION HISTORY:

```
Created November 2003 (JKD)
```

2.0.110 rfinterp (Source File: rfinterp.f90)

INTERFACE:

```
Subroutine rfinterp (ni, xi, ldi, fi, no, xo, ldo, fo)
```

INPUT/OUTPUT PARAMETERS:

```
ni  : number of input points (in,integer)
xi  : input abscissa array (in,real(ni))
ldi : leading dimension (in,integer)
fi  : input data array (in,real(ldi,ni)
no  : number of output points (in,integer)
xo  : output abscissa array (in,real(ni))
ldo : leading dimension (in,integer)
fo  : output interpolated function (out,real(ldo,no))
```

DESCRIPTION:

Given a function defined on a set of input points, this routine uses a clamped cubic spline to interpolate the function on a different set of points. See routine `spline`.

REVISION HISTORY:

```
Created January 2005 (JKD)
```

2.0.111 r3cross (Source File: r3cross.f90)

INTERFACE:

```
Subroutine r3cross (x, y, z)
```

INPUT/OUTPUT PARAMETERS:

```
x : input vector 1 (in,real(3))
y : input vector 2 (in,real(3))
z : output cross-product (out,real(3))
```

DESCRIPTION:

Returns the cross product of two real 3-vectors.

REVISION HISTORY:

```
Created September 2002 (JKD)
```

2.0.112 r3dist (Source File: r3dist.f90)

INTERFACE:

```
Real (8) Function r3dist (x, y)
```

INPUT/OUTPUT PARAMETERS:

```
x : input vector 1 (in,real(3))
y : input vector 2 (in,real(3))
```

DESCRIPTION:

Returns the distance between two real 3-vectors: $d = |\mathbf{x} - \mathbf{y}|$.

REVISION HISTORY:

```
Created January 2003 (JKD)
```

2.0.113 gradzfmt (Source File: gradzfmt.f90)

INTERFACE:

```
Subroutine gradzfmt (lmax, nr, r, ld1, ld2, zfmt, gzfmt)
```

INPUT/OUTPUT PARAMETERS:

```
lmax  : maximum angular momentum (in,integer)
nr    : number of radial mesh points (in,integer)
r     : radial mesh (in,real(nr))
ld1   : leading dimension 1 (in,integer)
ld2   : leading dimension 2 (in,integer)
zfmt  : complex muffin-tin function (in,complex(ld1,nr))
gzfmt : gradient of zfmt (out,complex(ld1,ld2,3))
```

DESCRIPTION:

Calculates the gradient of a complex muffin-tin function. In other words, given the spherical harmonic expansion coefficients, $f_{lm}(r)$, of a function $f(\mathbf{r})$, the routine returns $\mathbf{F}_{lm}$ where

$$\sum_{lm} \mathbf{F}_{lm}(r) Y_{lm}(\hat{\mathbf{r}}) = \nabla f(\mathbf{r}).$$

This is done using the identity

$$\begin{aligned}\nabla \left[f_{lm}(r) Y_{lm}(\hat{\mathbf{r}}) \right] = &- \left[\frac{l+1}{2l+1} \right]^{1/2} \left[\frac{d}{dr} - \frac{l}{r} \right] f_{lm}(r) \mathbf{Y}_{ll+1m}(\hat{\mathbf{r}}) \\ &+ \left[\frac{l}{2l+1} \right]^{1/2} \left[\frac{d}{dr} + \frac{l+1}{r} \right] f_{lm}(r) \mathbf{Y}_{ll-1m}(\hat{\mathbf{r}}),\end{aligned}$$

where the vector spherical harmonics are given by

$$\mathbf{Y}_{ll'm}(\hat{\mathbf{r}}) = \sum_{m'=-l'}^{l'} \sum_{m''=-1}^{1} C(l'1l|m'm''m)\, Y_{lm}(\hat{\mathbf{r}})\, \hat{\mathbf{e}}_{m''},$$

C is a Clebsch-Gordan coefficient and

$$\hat{\mathbf{e}}_{+1} = -\frac{\hat{\mathbf{x}} + i\hat{\mathbf{y}}}{\sqrt{2}}, \qquad \hat{\mathbf{e}}_0 = \hat{\mathbf{z}}, \qquad \hat{\mathbf{e}}_{-1} = \frac{\hat{\mathbf{x}} - i\hat{\mathbf{y}}}{\sqrt{2}}$$

are unit vectors. Note that the gradient returned is in terms of the global $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$ coordinate system.

REVISION HISTORY:

Created August 2003 (JKD)

2.0.114 rdiracint (Source File: rdiracint.f90)

INTERFACE:

```
Subroutine rdiracint (m, kpa, e, nr, r, vr, nn, g0p, f0p, g0, g1, f0, f1,sloppy)
```

INPUT/OUTPUT PARAMETERS:

```
m   : order of energy derivative (in,integer)
kpa : quantum number kappa (in,integer)
e   : energy (in,real)
nr  : number of radial mesh points (in,integer)
r   : radial mesh (in,real(nr))
vr  : potential on radial mesh (in,real(nr))
nn  : number of nodes (out,integer)
g0p : m-1 th energy derivative of the major component multiplied by r
      (in,real(nr))
f0p : m-1 th energy derivative of the minor component multiplied by r
      (in,real(nr))
g0  : m th energy derivative of the major component multiplied by r
      (out,real(nr))
g1  : radial derivative of g0 (out,real(nr))
f0  : m th energy derivative of the minor component multiplied by r
      (out,real(nr))
f1  : radial derivative of f0 (out,real(nr))
sloppy : whether a quick-and-dirty integration algorithm should be used.
```

DESCRIPTION:

Integrates the mth energy derivative of the radial Dirac equation from $r = 0$ outwards.

REVISION HISTORY:

Created February 2013 (Andris)

2.0.115 sdelta_sq (Source File: sdelta_sq.f90)

INTERFACE:

```
Real (8) Function sdelta_sq (x)
```

INPUT/OUTPUT PARAMETERS:

```
x : real argument (in,real)
```

DESCRIPTION:

Returns the square-wave pulse approximation to the Dirac delta function

$$\tilde{\delta}(x) = \begin{cases} 1 & |x| \le 1/2 \\ 0 & |x| > 1/2 \end{cases}$$

REVISION HISTORY:

```
Created July 2008 (JKD)
```

2.0.116 sphcover (Source File: sphcover.f90)

INTERFACE:

```
Subroutine sphcover (n, tp)
```

INPUT/OUTPUT PARAMETERS:

```
n  : number of required points (in,integer)
tp : (theta, phi) coordinates (out,real(2,n))
```

DESCRIPTION:

Produces a set of N points which cover the unit sphere nearly optimally. The points in (θ, ϕ) coordinates are generated using the explicit formula

$$\theta_k = \arccos(h_k), \qquad h_k = \frac{2(k-1)}{N-1} - 1, \qquad 1 \le k \le N$$
$$\phi_k = \left(\phi_{k-1} + C/\sqrt{N(1-h_k^2)} \right) \pmod{2\pi}, \qquad 2 \le k \le N-1, \qquad \phi_1 = \phi_N = 0,$$

where $C = (8\pi/\sqrt{3})^{1/2}$. See E. B. Saff and A. B. J. Kuijlaars, *Math. Intell.* **19**, 5 (1997).

REVISION HISTORY:

```
Created April 2008 (JKD)
```

2.0.117 genrlm (Source File: genrlm.f90)

INTERFACE:

```
Subroutine genrlm (lmax, tp, rlm)
```

INPUT/OUTPUT PARAMETERS:

```
lmax : maximum angular momentum (in,integer)
tp   : (theta, phi) coordinates (in,real(2))
rlm  : array of real spherical harmonics (out,real((lmax+1)**2))
```

DESCRIPTION:

Generates a sequence of real spherical harmonics evaluated at angles (θ, ϕ) for $0 < l < l_{\max}$. The values are returned in a packed array `rlm` indexed with $j = l(l+1) + m + 1$. Real spherical harmonics are defined by

$$R_{lm}(\theta,\phi) = \begin{cases} \sqrt{2}\,\Re\{Y_{lm}(\theta,\phi)\} & m > 0 \\ \sqrt{2}\,\Im\{Y_{lm}(\theta,\phi)\} & m < 0\,, \\ \Re\{Y_{lm}(\theta,\phi)\} & m = 0 \end{cases}$$

where Y_{lm} are the complex spherical harmonics. These functions are orthonormal and complete and may be used for expanding real-valued functions on the sphere. This routine is numerically stable and accurate to near machine precision for $l \leq 50$. See routine `genylm`.

REVISION HISTORY:

```
Created March 2004 (JKD)
```

2.0.118 genylm (Source File: genylm.f90)

INTERFACE:

```
Subroutine genylm (lmax, tp, ylm)
```

INPUT/OUTPUT PARAMETERS:

```
lmax : maximum angular momentum (in,integer)
tp   : (theta, phi) coordinates (in,real(2))
ylm  : array of spherical harmonics (out,complex((lmax+1)**2))
```

DESCRIPTION:

Generates a sequence of spherical harmonics, including the Condon-Shortley phase, evaluated at angles (θ, ϕ) for $0 < l < l_{\max}$. The values are returned in a packed array `ylm` indexed with $j = l(l+1) + m + 1$. The algorithm of Masters and Richards-Dinger is used, *Geophys. J. Int.* **135**, 307 (1998). This routine is numerically stable and accurate to near machine precision for $l \leq 50$.

REVISION HISTORY:

```
Created March 2004 (JKD)
Improved stability, December 2005 (JKD)
```

2.0.119 i3mtv (Source File: i3mtv.f90)

INTERFACE:

```
Subroutine i3mtv (a, x, y)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix (in,integer(3,3))
x : input vector (in,integer(3))
y : output vector (out,integer(3))
```

DESCRIPTION:

Multiplies the transpose of an integer 3×3 matrix with a vector.

REVISION HISTORY:

```
Created April 2007 (JKD)
```

2.0.120 hermite (Source File: hermite.f90)

INTERFACE:

```
Real (8) Function hermite (n, x)
```

INPUT/OUTPUT PARAMETERS:

```
n : order of Hermite polynomial (in,integer)
x : real argument (in,real)
```

DESCRIPTION:

Returns the nth Hermite polynomial. The recurrence relation

$$H_i(x) = 2xH_{i-1}(x) - 2nH_{i-2}(x),$$

with $H_0 = 1$ and $H_1 = 2x$, is used. This procedure is numerically stable and accurate to near machine precision for $n \leq 20$.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.121 rschrodint (Source File: rschrodint.f90)

INTERFACE:

```
Subroutine rschrodint (m, l, e, nr, r, vr, nn, rmfactor, p0p, p0, p1, q0, q1)
```

INPUT/OUTPUT PARAMETERS:

```
m   : order of energy derivative (in,integer)
l   : angular momentum quantum number (in,integer)
e   : energy (in,real)
nr  : number of radial mesh points (in,integer)
r   : radial mesh (in,real(nr))
vr  : potential on radial mesh (in,real(nr))
nn  : number of nodes (out,integer)
p0p : m-1 th energy derivative of P (in,real(nr))
p0  : m th energy derivative of P (out,real(nr))
p1  : radial derivative of p0 (out,real(nr))
q0  : m th energy derivative of Q (out,real(nr))
q1  : radial derivative of q0 (out,real(nr))
```

DESCRIPTION:

Integrates the mth energy derivative of the scalar relativistic radial Schrödinger equation from $r = 0$ outwards. The kinetic energy operator is based on ZORA.

REVISION HISTORY:

```
Created February 2013 (Andris)
```

2.0.122 i3mdet (Source File: i3mdet.f90)

INTERFACE:

```
Integer Function i3mdet (a)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix (in,integer(3,3))
```

DESCRIPTION:

Returns the determinant of an integer 3×3 matrix A.

REVISION HISTORY:

```
Created October 2004 (JKD)
```

2.0.123 z2mmct (Source File: z2mmct.f90)

INTERFACE:

```
Subroutine z2mmct (a, b, c)
```

INPUT/OUTPUT PARAMETERS:

```
a   : input matrix 1 (in,complex(2,2))
b   : input matrix 2 (in,complex(2,2))
c   : output matrix (out,complex(2,2))
```

DESCRIPTION:

Multiplies a 2×2 matrix with the conjugate transpose of another. Note that the output matrix cannot be one of the input matrices.

REVISION HISTORY:

```
Created October 2007 (JKD)
```

2.0.124 sbesseldm (Source File: sbesseldm.f90)

INTERFACE:

```
Subroutine sbesseldm (m, lmax, x, djl)
```

INPUT/OUTPUT PARAMETERS:

```
m    : order of derivatve (in,integer)
lmax : maximum order of Bessel function (in,integer)
x    : real argument (in,real)
djl  : array of returned values (out,real(0:lmax))
```

DESCRIPTION:

Computes the mth derivative of the spherical Bessel function of the first kind, $j_l(x)$, for argument x and $l = 0, 1, \ldots, l_{\max}$. For $x \geq 1$ this is done by repeatedly using the relations

$$\frac{d}{dx} j_l(x) = \frac{l}{x} j_l(x) - j_{l+1}(x)$$
$$j_{l+1}(x) = \frac{2l+1}{x} j_l(x) - j_{l-1}(x).$$

While for $x < 1$ the series expansion of the Bessel function is used

$$\frac{d^m}{dx^m} j_l(x) = \sum_{i=0}^{\infty} \frac{(2i+l)!}{(-2)^i i!(2i+l-m)!(2i+2l+1)!!} x^{2i+l-m}.$$

This procedure is numerically stable and accurate to near machine precision for $l \leq 30$ and $m \leq 6$.

REVISION HISTORY:

```
Created March 2003 (JKD)
Modified to return an array of values, October 2004 (JKD)
```

2.0.125 rdirac (Source File: rdirac.f90)

INTERFACE:

```
Subroutine rdirac (m, n, l, k, nr, r, vr, eval, g0, f0,dirac_eq,sloppy)
use modinput
```

INPUT/OUTPUT PARAMETERS:

```
m    : energy-derivative order (in,integer)
n    : principal quantum number (in,integer)
l    : quantum number l (in,integer)
k    : quantum number k (l or l+1) (in,integer)
nr   : number of radial mesh points (in,integer)
r    : radial mesh (in,real(nr))
vr   : potential on radial mesh (in,real(nr))
eval : eigenvalue without rest-mass energy (inout,real)
g0   : major component of the radial wavefunction (out,real(nr))
f0   : minor component of the radial wavefunction (out,real(nr))
dirac_eq : flag to pick the equation (Dirac or Schroedinger)
sloppy   : flag to pick a quick-and-dirty algorithm for integrating the Dirac equation
```

DESCRIPTION:

Finds the solution to the radial Dirac equation for a given potential $v(r)$ and quantum numbers n, k and l. The method involves integrating the equation using the predictor-corrector method and adjusting E until the number of nodes in the wavefunction equals $n-l-1$. The calling routine must provide an initial estimate for the eigenvalue. Note that the arrays `g0` and `f0` represent the radial functions multiplied by r.

REVISION HISTORY:

```
Created September 2002 (JKD)
Rewritten 2013 (Andris)
```

2.0.126 r3mmt (Source File: r3mmt.f90)

INTERFACE:

```
Subroutine r3mmt (a, b, c)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix 1 (in,real(3,3))
b : input matrix 2 (in,real(3,3))
c : output matrix (out,real(3,3))
```

DESCRIPTION:

Multiplies a real matrix with the transpose of another.

REVISION HISTORY:

```
Created January 2003 (JKD)
```

2.0.127 zflmconj (Source File: zflmconj.f90)

INTERFACE:

```
Subroutine zflmconj (lmax, zflm1, zflm2)
```

INPUT/OUTPUT PARAMETERS:

```
lmax  : maximum angular momentum (in,integer)
zflm1 : coefficients of input complex spherical harmonic expansion
        (in,complex((lmax+1)**2)))
zflm2 : coefficients of output complex spherical harmonic expansion
        (out,complex((lmax+1)**2)))
```

DESCRIPTION:

Returns the complex conjugate of a function expanded in spherical harmonics. In other words, given the input function coefficients z_{lm}, the routine returns $z'_{lm} = (-1)^m z^*_{l-m}$ so that

$$\sum_{lm} z'_{lm} Y_{lm}(\theta,\phi) = \left(\sum_{lm} z_{lm} Y_{lm}(\theta,\phi) \right)^*$$

for all (θ,ϕ). Note that `zflm1` and `zflm2` can refer to the same array.

REVISION HISTORY:

```
Created April 2004 (JKD)
```

2.0.128 gradrfmt (Source File: gradrfmt.f90)

INTERFACE:

```
Subroutine gradrfmt (lmax, nr, r, ld1, ld2, rfmt, grfmt)
```

INPUT/OUTPUT PARAMETERS:

```
lmax  : maximum angular momentum (in,integer)
nr    : number of radial mesh points (in,integer)
r     : radial mesh (in,real(nr))
ld1   : leading dimension 1 (in,integer)
ld2   : leading dimension 2 (in,integer)
rfmt  : real muffin-tin function (in,real(ld1,nr))
grfmt : gradient of rfmt (out,real(ld1,ld2,3))
```

DESCRIPTION:

Calculates the gradient of a real muffin-tin function. In other words, given the real spherical harmonic expansion coefficients, $f_{lm}(r)$, of a function $f(\mathbf{r})$, the routine returns $\mathbf{F}_{lm}$ where

$$\sum_{lm} \mathbf{F}_{lm}(r) R_{lm}(\hat{\mathbf{r}}) = \nabla f(\mathbf{r}),$$

and R_{lm} is a real spherical harmonic function. This is done by first converting the function to a complex spherical harmonic expansion and then using the routine `gradzfmt`. See routine `genrlm`.

REVISION HISTORY:

`Created August 2003 (JKD)`

2.0.129 r3frac (Source File: r3frac.f90)

INTERFACE:

```
Subroutine r3frac (eps, v, iv)
```

INPUT/OUTPUT PARAMETERS:

```
eps : zero component tolerance (in,real)
v   : input vector (inout,real(3))
iv  : integer parts of v (out,integer(3))
```

DESCRIPTION:

Finds the fractional part of each component of a real 3-vector using the function $\text{frac}\,(x) = x - \lfloor x \rfloor$. A component is taken to be zero if it lies within the intervals $[0, \epsilon)$ or $(1 - \epsilon, 1]$. The integer components of `v` are returned in the variable `iv`.

REVISION HISTORY:

`Created January 2003 (JKD)`

2.0.130 wigner3j (Source File: wigner3j.f90)

INTERFACE:

```
Real (8) Function wigner3j (j1, j2, j3, m1, m2, m3)
```

INPUT/OUTPUT PARAMETERS:

```
j1, j2, j3 : angular momentum quantum numbers (in,integer)
m1, m2, m3 : magnetic quantum numbers (in,integer)
```

DESCRIPTION:

Returns the Wigner $3j$-symbol. There are many equivalent definitions for the $3j$-symbols, the following provides high accuracy for $j \leq 50$

$$\begin{pmatrix} j_1 & j_2 & j_3 \\ m_1 & m_2 & m_3 \end{pmatrix} = (-1)^{j1+j2+m3}$$

$$\times \sqrt{\frac{(j_1+m_1)!(j_2+m_2)!(j_3+m_3)!(j_3-m_3)!(j_1-m_1)!(j_2-m_2)!}{(j_2-j_1+j_3)!(j_1-j_2+j_3)!(j_1+j_2-j_3)!(1+j_1+j_2+j_3)!}} \times \sum_{\max(0,j_2-j_3-m_1,j_1-j_3+m_2)}^{\min(j_1+j_2-j_3,j_1-m_1,j_2+m_2)}$$

$$(-1)^k \frac{(j_2-j_1+j_3)!(j_1-j_2+j_3)!(j_1+j_2-j_3)!}{(j_3-j_1-m_2+k)!(j_3-j_2+m_1+k)!(j_1+j_2-j_3-k)!k!(j_1-m_1-k)!(j_2+m_2-k)}.$$

REVISION HISTORY:

```
Created November 2002 (JKD)
```

2.0.131 rdiracdme (Source File: rdiracdme.f90)

INTERFACE:

```
Subroutine rdiracdme (m, kpa, e, nr, r, vr, nn, g0, g1, f0, f1, sloppy)
use mod_timing
```

INPUT/OUTPUT PARAMETERS:

```
m   : order of energy derivative (in,integer)
kpa : quantum number kappa (in,integer)
e   : energy (in,real)
nr  : number of radial mesh points (in,integer)
r   : radial mesh (in,real(nr))
vr  : potential on radial mesh (in,real(nr))
nn  : number of nodes (out,integer)
g0  : m th energy derivative of the major component multiplied by r
      (out,real(nr))
g1  : radial derivative of g0 (out,real(nr))
f0  : m th energy derivative of the minor component multiplied by r
      (out,real(nr))
f1  : radial derivative of f0 (out,real(nr))
```

DESCRIPTION:

Finds the solution to the mth energy derivative of the radial Dirac equation using the routine `rdiracint`.

REVISION HISTORY:

```
Created March 2003 (JKD)
```

2.0.132 euler (Source File: euler.f90)

INTERFACE:

```
Subroutine euler (rot, ang)
```

INPUT/OUTPUT PARAMETERS:

```
rot : rotation matrix (in,real(3,3))
ang : euler angles (alpha, beta, gamma) (out,real(3))
```

DESCRIPTION:

Given a rotation matrix

$$R(\alpha,\beta,\gamma) = \begin{pmatrix} \cos\gamma\cos\beta\cos\alpha - \sin\gamma\sin\alpha & \cos\gamma\cos\beta\sin\alpha + \sin\gamma\cos\alpha & -\cos\gamma\sin\beta \\ -\sin\gamma\cos\beta\cos\alpha - \cos\gamma\sin\alpha & -\sin\gamma\cos\beta\sin\alpha + \cos\gamma\cos\alpha & \sin\gamma\sin\beta \\ \sin\beta\cos\alpha & \sin\beta\sin\alpha & \cos\beta \end{pmatrix},$$

this routine determines the Euler angles, (α,β,γ). This corresponds to the so-called "y-convention", which involves the following successive rotations of the coordinate system:

1. The x_1'-, x_2'-, x_3'-axes are rotated anticlockwise through an angle α about the x_3 axis
2. The x_1''-, x_2''-, x_3''-axes are rotated anticlockwise through an angle β about the x_2' axis
3. The x_1'''-, x_2'''-, x_3'''-axes are rotated anticlockwise through an angle γ about the x_3'' axis

Note that the Euler angles are not necessarily unique for a given rotation matrix.
REVISION HISTORY:

```
Created May 2003 (JKD)
```

2.0.133 brzint (Source File: brzint.f90)

INTERFACE:

```
Subroutine brzint (nsm, ngridk, nsk, ikmap, nw, wint, n, ld, e, f, g)
```

INPUT/OUTPUT PARAMETERS:

```
nsm    : level of smoothing for output function (in,integer)
ngridk : k-point grid size (in,integer(3))
nsk    : k-point subdivision grid size (in,integer(3))
ikmap  : map from grid to k-point set
         (in,integer(0:ngridk(1)-1,0:ngridk(2)-1,0:ngridk(3)-1))
nw     : number of energy divisions (in,integer)
wint   : energy interval (in,real(2))
n      : number of functions to integrate (in,integer)
ld     : leading dimension (in,integer)
e      : array of energies as a function of k-points (in,real(ld,*))
f      : array of weights as a function of k-points (in,real(ld,*))
g      : output function (out,real(nw))
```

DESCRIPTION:

Given energy and weight functions, e and f, on the Brillouin zone and a set of equidistant energies ω_i, this routine computes the integrals

$$g(\omega_i) = \frac{\Omega}{(2\pi)^3}\int_{\mathrm{BZ}} f(\mathbf{k})\delta(\omega_i - e(\mathbf{k}))d\mathbf{k},$$

where Ω is the unit cell volume. This is done by first interpolating e and f on a finer k-point grid using the trilinear method. Then for each $e(\mathbf{k})$ on the finer grid the nearest ω_i is found and $f(\mathbf{k})$ is accumulated in $g(\omega_i)$. If the output function is noisy then either `nsk` should be increased or `nw` decreased. Alternatively, the output function can be artificially smoothed up to a level given by `nsm`. See routine `fsmooth`.

REVISION HISTORY:

```
Created October 2003 (JKD)
Improved efficiency May 2007 (Sebastian Lebegue)
```

2.0.134 r3mtm (Source File: r3mtm.f90)

INTERFACE:

```
Subroutine r3mtm (a, b, c)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix 1 (in,real(3,3))
b : input matrix 2 (in,real(3,3))
c : output matrix (out,real(3,3))
```

DESCRIPTION:

Multiplies the transpose of one real 3×3 matrix with another.

REVISION HISTORY:

```
Created January 2003 (JKD)
```

2.0.135 factnm (Source File: factnm.f90)

INTERFACE:

```
Real (8) Function factnm (n, m)
```

INPUT/OUTPUT PARAMETERS:

```
n : input (in,integer)
m : order of multifactorial (in,integer)
```

DESCRIPTION:

Returns the multifactorial

$$n \underbrace{!!...!}_{m\,\text{times}} = \prod_{i \geq 0,\, n-im>0} n - im$$

for n, $m \geq 0$. n should be less than 150.

REVISION HISTORY:

```
Created January 2003 (JKD)
```

2.0.136 z3minv (Source File: z3minv.f90)

INTERFACE:

```
Subroutine z3minv (a, b)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix (in,complex(3,3))
b : output matrix (out,complex(3,3))
```

DESCRIPTION:

Computes the inverse of a complex 3×3 matrix.

REVISION HISTORY:

```
Created March 2014 (STK)
```

2.0.137 r3mtv (Source File: r3mtv.f90)

INTERFACE:

```
Subroutine r3mtv (a, x, y)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix (in,real(3,3))
x : input vector (in,real(3))
y : output vector (out,real(3))
```

DESCRIPTION:

Multiplies the transpose of a real 3×3 matrix with a vector.

REVISION HISTORY:

```
Created January 2003 (JKD)
```

2.0.138 r3minv (Source File: r3minv.f90)

INTERFACE:

```
Subroutine r3minv (a, b)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix (in,real(3,3))
b : output matrix (in,real(3,3))
```

DESCRIPTION:

Computes the inverse of a real 3×3 matrix.
REVISION HISTORY:

```
Created April 2003 (JKD)
Changed May 2018 (SeTi)
```

2.0.139 fsmooth (Source File: fsmooth.f90)

Subroutine fsmooth (m, n, ld, f) **INPUT/OUTPUT PARAMETERS:**

```
m  : number of 3-point running averages to perform (in,integer)
n  : number of point (in,integer)
ld : leading dimension (in,integer)
f  : function array (inout,real(ld,n))
```

DESCRIPTION:

Removes numerical noise from a function by performing m successive 3-point running averages on the data. The endpoints are kept fixed.
REVISION HISTORY:

```
Created December 2005 (JKD)
```

2.0.140 r3mm (Source File: r3mm.f90)

INTERFACE:

```
Subroutine r3mm (a, b, c)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix 1 (in,real(3,3))
b : input matrix 2 (in,real(3,3))
c : output matrix (out,real(3,3))
```

DESCRIPTION:

Multiplies two real 3×3 matrices.
REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.141 r3mv (Source File: r3mv.f90)

INTERFACE:

```
Subroutine r3mv (a, x, y)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix (in,real(3,3))
x : input vector (in,real(3))
y : output vector (out,real(3))
```

DESCRIPTION:

Multiplies a real 3×3 matrix with a vector.

REVISION HISTORY:

```
Created January 2003 (JKD)
```

2.0.142 connect (Source File: connect.f90)

INTERFACE:

```
Subroutine connect (cvec, plotdef, nv, np, vpl, dv, dp)
```

INPUT/OUTPUT PARAMETERS:

```
cvec : matrix of (reciprocal) lattice vectors stored column-wise
       (in,real(3,3))
nv   : number of vertices (in,integer)
np   : number of connecting points (in,integer)
vvl  : vertex vectors in lattice coordinates (in,real(3,nv))
vpl  : connecting point vectors in lattice coordinates (out,real(3,np))
dv   : cummulative distance to each vertex (out,real(nv))
dp   : cummulative distance to each connecting point (out,real(np))
```

DESCRIPTION:

Generates a set of points which interpolate between a given set of vertices. Vertex points are supplied in lattice coordinates in the array `vvl` and converted to Cartesian coordinates with the matrix `cvec`. Interpolating points are stored in the array `vpl`. The cummulative distances to the vertices and points along the path are stored in arrays `dv` and `dp`, respectively.

REVISION HISTORY:

```
Created June 2003 (JKD)
Improved September 2007 (JKD)
```

2.0.143 brzint_jdos (Source File: brzint_jdos.f90)

INTERFACE:

```
subroutine brzint_jdos( nsm, ngridk, nsk, ikmap, nw, wint, n, ld, e, f, fmax, g)
```

INPUT/OUTPUT PARAMETERS:

```
nsm    : level of smoothing for output function (in,integer)
ngridk : k-point grid size (in,integer(3))
nsk    : k-point subdivision grid size (in,integer(3))
ikmap  : map from grid to k-point set
         (in,integer(0:ngridk(1)-1,0:ngridk(2)-1,0:ngridk(3)-1))
nw     : number of energy divisions (in,integer)
wint   : energy interval (in,real(2))
n      : number of functions to integrate (in,integer)
ld     : leading dimension (in,integer)
e      : array of energies as a function of k-points (in,real(ld,*))
f      : array of weights as a function of k-points (in,real(ld,*))
g      : output function (out,real(nw,n))
```

DESCRIPTION:

See routine `brzint`. This routine treats the integral over the trilinear interpolated values analytically in two dimensions and numerically in the third dimension (instead of numerically in all three dimensions). This and some smaller changes makes this routine much faster and more accurate for dense interpolation grids `nsk` or a large number of energy divisions `nw`. The solution does usually not show strong noise.

REVISION HISTORY:

```
Created June 2018 (SeTi)
```

2.0.144 rschroddme (Source File: rschroddme.f90)

INTERFACE:

```
Subroutine rschroddme (m, l, k, e, nr, r, vr, nn, p0, p1, q0, q1)
use mod_timing
use modinput
```

INPUT/OUTPUT PARAMETERS:

```
m   : order of energy derivative (in,integer)
l   : angular momentum quantum number (in,integer)
k   : quantum number k, zero if Dirac eqn. is not to be used (in,integer)
e   : energy (in,real)
nr  : number of radial mesh points (in,integer)
r   : radial mesh (in,real(nr))
```

```
vr  : potential on radial mesh (in,real(nr))
nn  : number of nodes (out,integer)
p0  : m th energy derivative of P (out,real(nr))
p1  : radial derivative of p0 (out,real(nr))
q0  : m th energy derivative of Q (out,real(nr))
q1  : radial derivative of q0 (out,real(nr))
```

DESCRIPTION:

Finds the solution to the mth energy derivative of the scalar relativistic radial Schrödinger equation using the routine `rschrodint`.

REVISION HISTORY:

```
Created June 2003 (JKD)
```

2.0.145 stheta (Source File: stheta.f90)

INTERFACE:

```
Real (8) Function stheta (stype, x)
```

INPUT/OUTPUT PARAMETERS:

```
stype : smearing type (in,integer)
x     : real argument (in,real)
```

DESCRIPTION:

Returns the Heaviside step function corresponding to the smooth approximation to the Dirac delta function:

$$\tilde{\Theta}(x) = \int_{-\infty}^{x} dt\, \tilde{\delta}(t).$$

See function `sdelta` for details.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.146 brzint_new (Source File: brzint_new.f90)

INTERFACE:

```
subroutine brzint_new( nsm, ngridk, nsk, ikmap, nw, wint, n, ld, e, f, g)
```

INPUT/OUTPUT PARAMETERS:

```
nsm    : level of smoothing for output function (in,integer)
ngridk : k-point grid size (in,integer(3))
nsk    : k-point subdivision grid size (in,integer(3))
```

```
ikmap  : map from grid to k-point set
         (in,integer(0:ngridk(1)-1,0:ngridk(2)-1,0:ngridk(3)-1))
nw     : number of energy divisions (in,integer)
wint   : energy interval (in,real(2))
n      : number of functions to integrate (in,integer)
ld     : leading dimension (in,integer)
e      : array of energies as a function of k-points (in,real(ld,*))
f      : array of weights as a function of k-points (in,real(ld,*))
g      : output function (out,real(nw))
```

DESCRIPTION:

See routine `brzint`. This routine treats the integral over the trilinear interpolated values analytically in two dimensions and numerically in the third dimension (instead of numerically in all three dimensions). This and some smaller changes makes this routine much faster and more accurate for dense interpolation grids `nsk` or a large number of energy divisions `nw`. The solution does usually not show strong noise.

REVISION HISTORY:

```
Created June 2018 (SeTi)
```

2.0.147 sdelta (Source File: sdelta.f90)

INTERFACE:

```
Function sdelta (stype, x)
```

INPUT/OUTPUT PARAMETERS:

```
stype : smearing type (in,integer)
x     : real argument (in,real)
```

DESCRIPTION:

Returns a normalised smooth approximation to the Dirac delta function. These functions are defined such that

$$\int \tilde{\delta}(x)dx = 1.$$

The effective width, w, of the delta function may be varied by using the normalising transformation

$$\tilde{\delta}_w(x) \equiv \frac{\tilde{\delta}(x/w)}{w}.$$

Currently implimented are:

0. Gaussian
1. Methfessel-Paxton order 1
2. Methfessel-Paxton order 2
3. Fermi-Dirac

4. Square-wave impulse

See routines `stheta`, `sdelta_mp`, `sdelta_fd` and `sdelta_sq`.
REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.148 getsdata (Source File: sdelta.f90)

INTERFACE:

```
Subroutine getsdata (stype, sdescr)
```

INPUT/OUTPUT PARAMETERS:

```
stype  : smearing type (in,integer)
sdescr : smearing scheme description (out,character(256))
```

DESCRIPTION:

Returns a description of the smearing scheme as string `sdescr` up to 256 characters long.
REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.149 rkhint (Source File: rkhint.f90)

INTERFACE:

```
Subroutine rkhint (m, l, e, nr, r, vr, nn, p0p, q0p, p0, p1, q0, q1)
```

INPUT/OUTPUT PARAMETERS:

```
m   : order of energy derivative (in,integer)
l   : angular momentum quantum number (in,integer)
e   : energy (in,real)
nr  : number of radial mesh points (in,integer)
r   : radial mesh (in,real(nr))
vr  : potential on radial mesh (in,real(nr))
nn  : number of nodes (out,integer)
p0p : m-1 th energy derivative of P (in,real(nr))
p0  : m th energy derivative of P (out,real(nr))
p1  : radial derivative of p0 (out,real(nr))
q0  : m th energy derivative of Q (out,real(nr))
q1  : radial derivative of q0 (out,real(nr))
```

DESCRIPTION:

Integrates the mth energy derivative of the scalar relativistic radial Schrödinger equation from $r = 0$ outwards.

REVISION HISTORY:

```
Created July 2013 (Andris)
```

2.0.150 vecfbz (Source File: vecfbz.f90)

INTERFACE:

```
Subroutine vecfbz (eps, bvec, vpl, iv)
```

INPUT/OUTPUT PARAMETERS:

```
eps  : zero component tolerance (in,real)
bvec : reciprocal lattice vectors (in,real(3,3))
vpl  : input vector in lattice coordinates (inout,real(3))
iv   : integer parts of vpl (out,integer(3))
```

DESCRIPTION:

Maps a vector in lattice coordinates to the first Brillouin zone. This is done by first removing its integer components and then adding primitive reciprocal lattice vectors until the shortest vector is found.

REVISION HISTORY:

```
Created September 2008 (JKD)
```

2.0.151 r3ws (Source File: r3ws.f90)

INTERFACE:

```
subroutine r3ws( eps, b, v, iv)
```

INPUT/OUTPUT PARAMETERS:

```
eps : zero component tolerance (in, real)
b   : basis vectors (in, real(3,3))
v   : input lattice vector (inout, real(3))
iv  : lattice vector that maps v in WS-cell (out,integer(3))
```

DESCRIPTION:

Finds the lattice vector `iv` that maps the real 3-vector `v` into the Wigner-Seitz cell. On exit, `v` contains a vector within the Wigner-Seitz cell.

REVISION HISTORY:

```
Created May 2018 (SeTi)
```

2.0.152 r3mdet (Source File: r3mdet.f90)

INTERFACE:

```
Real (8) Function r3mdet (a)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix (in,real(3,3))
```

DESCRIPTION:

Returns the determinant of a real 3×3 matrix A.

REVISION HISTORY:

```
Created May 2003 (JKD)
```

2.0.153 vnlrhomt (Source File: vnlrhomt.f90)

INTERFACE:

```
Subroutine vnlrhomt (tsh, is, wfmt1, wfmt2, zrhomt)
```

USES:

```
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
tsh    : .true. if the density is to be in spherical harmonics (in,logical)
is     : species number (in,integer)
wfmt1  : muffin-tin part of wavefunction 1 in spherical coordinates
         (in,complex(lmmaxvr,nrcmtmax,natmtot,nspinor))
wfmt2  : muffin-tin part of wavefunction 2 in spherical coordinates
         (in,complex(lmmaxvr,nrcmtmax,natmtot,nspinor))
zrhomt : muffin-tin charge density in spherical harmonics/coordinates
         (out,complex(lmmaxvr,nrcmtmax))
```

DESCRIPTION:

Calculates the complex overlap density in a single muffin-tin from two input wavefunctions expressed in spherical coordinates. If `tsh` is `.true.` then the output density is converted to a spherical harmonic expansion. See routine `vnlrho`.

REVISION HISTORY:

```
Created November 2004 (Sharma)
Modified April 2014 (UW)
```

2.0.154 vnlrho (Source File: vnlrho.f90)

INTERFACE:

```
Subroutine vnlrho (tsh, wfmt1, wfmt2, wfir1, wfir2, zrhomt, zrhoir)
```

USES:

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
tsh    : .true. if the muffin-tin density is to be in spherical harmonics
         (in,logical)
wfmt1  : muffin-tin part of wavefunction 1 in spherical coordinates
         (in,complex(lmmaxvr,nrcmtmax,natmtot,nspinor))
wfmt2  : muffin-tin part of wavefunction 2 in spherical coordinates
         (in,complex(lmmaxvr,nrcmtmax,natmtot,nspinor))
wfir1  : interstitial wavefunction 1 (in,complex(ngrtot))
wfir2  : interstitial wavefunction 2 (in,complex(ngrtot))
zrhomt : muffin-tin charge density in spherical harmonics/coordinates
         (out,complex(lmmaxvr,nrcmtmax,natmtot))
zrhoir : interstitial charge density (out,complex(ngrtot))
```

DESCRIPTION:

Calculates the complex overlap charge density from two input wavefunctions:

$$\rho(\mathbf{r}) \equiv \Psi_1^{\dagger}(\mathbf{r}) \cdot \Psi_2(\mathbf{r}).$$

Note that the muffin-tin wavefunctions are provided in spherical coordinates and the returned density is either in terms of spherical harmonic coefficients or spherical coordinates when `tsh` is `.true.` or `.false.`, respectively. See also the routine `vnlrhomt`.

REVISION HISTORY:

```
Created November 2004 (Sharma)
```

2.0.155 oepmain (Source File: oepmain.f90)

INTERFACE:

```
Subroutine oepmain
```

USES:

```
Use modmain
Use modinput
use modmpi
```

DESCRIPTION:

Main routine for the calculation of the optimized effective potential.

REVISION HISTORY:

```
Created ()
Modified Feb 2014 (UW)
Modified March 2014 (UW)
```

2.0.156 genwiq2 (Source File: genwiq2.f90)

INTERFACE:

```
Subroutine genwiq2
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

The Fock matrix elements

$$V_{ij\mathbf{k}}^{\mathrm{NL}} \equiv \sum_{l\mathbf{k}'} \int \frac{\Psi_{i\mathbf{k}}^{\dagger}(\mathbf{r}) \cdot \Psi_{l\mathbf{k}'}(\mathbf{r}) \Psi_{l\mathbf{k}'}^{\dagger}(\mathbf{r}') \cdot \Psi_{j\mathbf{k}}(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}\, d\mathbf{r}\, d\mathbf{r}'$$

contain a divergent term in the sum over $\mathbf{k}'$ which behaves as $1/q^2$, where $\mathbf{q} \equiv \mathbf{k} - \mathbf{k}'$ is in the first Brillouin zone. The resulting convergence with respect to the number of discrete q-points is very slow. This routine computes the weights

$$w_{\mathbf{q}_i} \equiv \int_{V_i} \frac{1}{q^2}\, d\mathbf{q}\,, \tag{10}$$

where the integral is over the small parallelepiped centered on $\mathbf{q}_i$, so that integrals over the first Brillouin zone of the form

$$I = \int_{\mathrm{BZ}} \frac{f(\mathbf{q})}{q^2}\, d\mathbf{q}\,,$$

can be approximated by the sum

$$I \approx \sum_i w_{\mathbf{q}_i} f(\mathbf{q}_i)$$

which converges rapidly with respect to the number of q-points for smooth functions f. The integral in (10) is determined by evaluating it numerically on increasingly finer grids and extrapolating to the continuum. Agreement with Mathematica to at least 10 significant figures.

REVISION HISTORY:

```
Created August 2004 (JKD,SS)
```

2.0.157 hmlistl (Source File: hmlistl.f90)

INTERFACE:

```
Subroutine hmlistl (tapp, ngp, igpig, vgpc, v, h)
```

USES:

```
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
tapp  : .true. if the Hamiltonian is to be applied to the input vector,
        .false. if the full matrix is to be calculated (in,logical)
ngp   : number of G+p-vectors (in,integer)
igpig : index from G+p-vectors to G-vectors (in,integer(ngkmax))
vgpc  : G+p-vectors in Cartesian coordinates (in,real(3,ngkmax))
v     : input vector to which H is applied if tapp is .true., otherwise
        not referenced (in,complex(nmatmax))
h     : H applied to v if tapp is .true., otherwise it is the Hamiltonian
        matrix in packed form (inout,complex(*))
```

DESCRIPTION:

Computes the interstitial contribution to the Hamiltonian matrix for the APW basis functions. The Hamiltonian is given by

$$H^{\rm I}(\mathbf{G}+\mathbf{k},\mathbf{G}'+\mathbf{k})=\frac{1}{2}(\mathbf{G}+\mathbf{k})\cdot(\mathbf{G}'+\mathbf{k})\tilde{\Theta}(\mathbf{G}-\mathbf{G}')+V^{\sigma}(\mathbf{G}-\mathbf{G}'),$$

where V^{σ} is the effective interstitial potential for spin σ and $\tilde{\Theta}$ is the characteristic function. See routine `gencfun`.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.158 olpistl (Source File: olpistl.f90)

INTERFACE:

```
Subroutine olpistl (tapp, ngp, igpig, v, o)
```

USES:

```
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
ngp   : number of G+p-vectors (in,integer)
igpig : index from G+p-vectors to G-vectors (in,integer(ngkmax))
v     : input vector to which O is applied if tapp is .true., otherwise
        not referenced (in,complex(nmatmax))
o     : O applied to v if tapp is .true., otherwise it is the overlap
        matrix in packed form (inout,complex(*))
```

DESCRIPTION:

Computes the interstitial contribution to the overlap matrix for the APW basis functions. The overlap is given by

$$O^{\mathrm{I}}(\mathbf{G}+\mathbf{k},\mathbf{G}'+\mathbf{k}) = \tilde{\Theta}(\mathbf{G}-\mathbf{G}'),$$

where $\tilde{\Theta}$ is the characteristic function. See routine `gencfun`.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.159 hmlistl (Source File: hmlistln.f90)

INTERFACE:

```
Subroutine hmlistln (hamilton, ngp, igpig, vgpc)
```

USES:

```
Use modmain
Use modfvsystem
```

INPUT/OUTPUT PARAMETERS:

```
tapp  : .true. if the Hamiltonian is to be applied to the input vector,
        .false. if the full matrix is to be calculated (in,logical)
ngp   : number of G+p-vectors (in,integer)
igpig : index from G+p-vectors to G-vectors (in,integer(ngkmax))
vgpc  : G+p-vectors in Cartesian coordinates (in,real(3,ngkmax))
v     : input vector to which H is applied if tapp is .true., otherwise
        not referenced (in,complex(nmatmax))
h     : H applied to v if tapp is .true., otherwise it is the Hamiltonian
        matrix in packed form (inout,complex(npmatmax))
```

DESCRIPTION:

Computes the interstitial contribution to the Hamiltonian matrix for the APW basis functions. The Hamiltonian is given by

$$H^{\mathrm{I}}(\mathbf{G}+\mathbf{k},\mathbf{G}'+\mathbf{k}) = \frac{1}{2}(\mathbf{G}+\mathbf{k})\cdot(\mathbf{G}'+\mathbf{k})\tilde{\Theta}(\mathbf{G}-\mathbf{G}') + V^{\sigma}(\mathbf{G}-\mathbf{G}'),$$

where V^{σ} is the effective interstitial potential for spin σ and $\tilde{\Theta}$ is the characteristic function. See routine `gencfun`.

REVISION HISTORY:

```
Created April 2003 (JKD)
```

2.0.160 hmlaa (Source File: hmlaa.f90)

INTERFACE:

```
Subroutine hmlaa (tapp, is, ia, ngp, apwalm, v, h)
```

USES:

```
Use modinput
Use modmain
```

INPUT/OUTPUT PARAMETERS:

```
tapp   : .true. if the Hamiltonian is to be applied to the input vector,
         .false. if the full matrix is to be calculated (in,logical)
is     : species number (in,integer)
ia     : atom number (in,integer)
ngp    : number of G+p-vectors (in,integer)
apwalm : APW matching coefficients
         (in,complex(ngkmax,apwordmax,lmmaxapw,natmtot))
v      : input vector to which H is applied if tapp is .true., otherwise
         not referenced (in,complex(nmatmax))
h      : H applied to v if tapp is .true., otherwise it is the Hamiltonian
         matrix in packed form (inout,complex(*))
```

DESCRIPTION:

Calculates the APW-APW contribution to the Hamiltonian matrix.

REVISION HISTORY:

```
Created October 2002 (JKD)
```

2.0.161 olpistl (Source File: olpistln.f90)

INTERFACE:

```
Subroutine olpistln (overlap, ngp, igpig, vgpc)
```

USES:

```
Use modmain
Use modfvsystem
```

INPUT/OUTPUT PARAMETERS:

```
ngp   : number of G+p-vectors (in,integer)
igpig : index from G+p-vectors to G-vectors (in,integer(ngkmax))
v     : input vector to which O is applied if tapp is .true., otherwise
        not referenced (in,complex(nmatmax))
o     : O applied to v if tapp is .true., otherwise it is the overlap
      : matrix in packed form (inout,complex(npmatmax))
```

DESCRIPTION:

Computes the interstitial contribution to the overlap matrix for the APW basis functions. The overlap is given by

$$O^{\mathrm{I}}(\mathbf{G}+\mathbf{k},\mathbf{G}'+\mathbf{k})=\tilde{\Theta}(\mathbf{G}-\mathbf{G}'),$$

where $\tilde{\Theta}$ is the characteristic function. See routine `gencfun`.

REVISION HISTORY:

`Created April 2003 (JKD)`

2.0.162 olprad (Source File: olprad.f90)

INTERFACE:

`Subroutine olprad`

USES:

`Use modmain`

DESCRIPTION:

Calculates the radial overlap integrals of the APW and local-orbital basis functions. In other words, for spin σ and atom j of species i, it computes integrals of the form

$$o_{qp}^{\sigma;ij}=\int_0^{R_i}u_{q;l_p}^{\sigma;ij}(r)v_p^{\sigma;ij}(r)r^2dr$$

and

$$o_{pp'}^{\sigma;ij}=\int_0^{R_i}v_p^{\sigma;ij}(r)v_{p'}^{\sigma;ij}(r)r^2dr,\quad l_p=l_{p'}$$

where $u_{q;l}^{\sigma;ij}$ is the qth APW radial function for angular momentum l; and $v_p^{\sigma;ij}$ is the pth local-orbital radial function and has angular momentum l_p.

REVISION HISTORY:

`Created November 2003 (JKD)`

2.0.163 hmlint (Source File: hmlint.f90)

INTERFACE:

`Subroutine hmlint`

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Calculates the "muffin-tin" Hamiltonian.

2.0.164 hmlrad (Source File: hmlrad.f90)

INTERFACE:

```
Subroutine hmlrad
```

USES:

```
Use modinput
Use modmain
```

DESCRIPTION:

Calculates the radial Hamiltonian integrals of the APW and local-orbital basis functions. In other words, for spin σ and atom j of species i, it computes integrals of the form

$$h^{\sigma;ij}_{qq';ll'l''m''}=\begin{cases}\int_0^{R_i}u^{\sigma;ij}_{q;l}(r)Hu^{\sigma;ij}_{q';l'}(r)r^2dr & l''=0\\ \int_0^{R_i}u^{\sigma;ij}_{q;l}(r)V^{\sigma;ij}_{l''m''}(r)u^{\sigma;ij}_{q';l'}(r)r^2dr & l''>0\end{cases},$$

where $u^{\sigma;ij}_{q;l}$ is the qth APW radial function for angular momentum l; H is the Hamiltonian of the radial Schrödinger equation; and $V^{\sigma;ij}_{l''m''}$ is the effective muffin-tin potential. Similar integrals are calculated for APW-local-orbital and local-orbital-local-orbital contributions.

REVISION HISTORY:

```
Created December 2003 (JKD)
```

2.0.165 initmpi (Source File: modmpi.F90)

INTERFACE:

```
subroutine initmpi
```

DESCRIPTION:

Initializes MPI and sets procs and rank numbers. Sets splittfile and initializes the first in node list. Or if -DMPI is not used sets procs=1 and rank=1 and splittfile=.false.

REVISION HISTORY:

```
Added to documentation scheme (Aurich)
```

2.0.166 finitmpi (Source File: modmpi.F90)

INTERFACE:

```
subroutine finitmpi
```

DESCRIPTION:

If -DMPI calls `mpi_finalize`, after wainting for the processes of mpiglobal, mpinodesto finish communicating.
Note: For other communicators you need to make shure that they finished communication.
REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
Modified to use new mpi types. (Aurich)
```

2.0.167 terminate (Source File: modmpi.F90)

INTERFACE:

```
subroutine terminate
```

DESCRIPTION:

Kills the program in `MPI` or single execution.
REVISION HISTORY:

```
Added to documentation scheme and moved to
modmpi. 2016 (Aurich)
```

2.0.168 nofset (Source File: modmpi.F90)

INTERFACE:

```
function nofset(myrank, set, nprocs)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
integer(4) :: myrank  ! MPI rank
integer(4) :: set     ! Total number of elements to distribute
integer(4), optional :: nprocs ! Number of processes in communicator
OUT:
integer(4) :: nofset  ! Number of elements for that rank
```

DESCRIPTION:

This functions helps with distributing a set of N_{el} elements to N_{p} `MPI` processes in continuous blocks. The function calculates the number of elements $N_{\mathrm{el}}(p)$ a given process is responsible for.
Example:
$N_{\mathrm{el}} = 10, N_{\mathrm{p}} = 3 \rightarrow N_{\mathrm{el}}(0) = 4, N_{\mathrm{el}}(1) = 3, N_{\mathrm{el}}(2) = 3$
Example:
$N_{\mathrm{el}} = 2, N_{\mathrm{p}} = 3 \rightarrow N_{\mathrm{el}}(0) = 1, N_{\mathrm{el}}(1) = 1, N_{\mathrm{el}}(2) = 0$
REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
Added sanity checks. (Aurich)
```

2.0.169 firstofset (Source File: modmpi.F90)

INTERFACE:

```
function firstofset(myrank, set, nprocs)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
integer(4) :: myrank  ! MPI rank
integer(4) :: set     ! Total number of elements to distribute
integer(4), optional :: nprocs  ! Number of processes in commuincator
OUT:
integer(4) :: firstofset ! Index of the total set for the first index
                         ! of the current subset
```

DESCRIPTION:

This functions helps with distributing a set of $N_{\rm el}$ elements to $N_{\rm p}$ `MPI` processes in continuous blocks. The function calculates the index of the fist element $i_{\rm el}(p)$ a given process is responsible for. If there are more processes than elements a process responsible for no element gets the assignment $i_{\rm el}(p > N_{\rm el} - 1) = 0$.
Example:
$N_{\rm el} = 10, N_{\rm p} = 3 \rightarrow i_{\rm el}(0) = 1, i_{\rm el}(1) = 5, i_{\rm el}(2) = 8$
Example:
$N_{\rm el} = 2, N_{\rm p} = 4 \rightarrow i_{\rm el}(0) = 1, i_{\rm el}(1) = 2, i_{\rm el}(2) = 0, i_{\rm el}(3) = 0$
REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
Changed behaviour if there are more processes than elements. (Aurich)
Added sanity checks. (Aurich)
```

2.0.170 lastofset (Source File: modmpi.F90)

INTERFACE:

```
function lastofset(myrank, set, nprocs)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
integer(4) :: myrank  ! MPI rank
integer(4) :: set     ! Total number of elements to distribute
integer(4) :: nprocs  ! Number of processes in commuincator
```

```
OUT:
integer(4) :: lastofset  ! Index of the total set for the first index
                         ! of the current subset
```

DESCRIPTION:

This functions helps with distributing a set of $N_{\rm el}$ elements to $N_{\rm p}$ `MPI` processes in continuous blocks. The function calculates the index of the last element $j_{\rm el}(p)$ a given process is responsible for. If there are more processes than elements, a process responsible for no element gets the assignment $j_{\rm el}(p > N_{\rm el} - 1) = -1$.
Example:
$N_{\rm el} = 10, N_{\rm p} = 3 \rightarrow j_{\rm el}(0) = 4, j_{\rm el}(1) = 7, j_{\rm el}(2) = 10$
Example:
$N_{\rm el} = 2, N_{\rm p} = 4 \rightarrow j_{\rm el}(0) = 1, j_{\rm el}(1) = 2, j_{\rm el}(2) = -1, j_{\rm el}(3) = -1$
REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
Changed behaviour if there are more processes than elements. (Aurich)
Added sanity checks. (Aurich)
```

2.0.171 procofindex (Source File: modmpi.F90)

INTERFACE:

```
function procofindex(k, set, nprocs)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
integer(4) :: k    ! Element number k
integer(4) :: set  ! Total number of distributed elements
integer(4), optional :: nprocs ! Number of processes in communicator
OUT:
integer(4) :: procofindex  ! Rank that holds the element
```

DESCRIPTION:

This functions helps with distributing a set of $N_{\rm el}$ elements to $N_{\rm p}$ `MPI` processes in continuous blocks. The function calculates the index of the process $i_{\rm p}(k)$ that is responsible for the element with index k. If k is larger than $N_{\rm el}$ or smaller than 1 the routine returns terminates the execution. Example:
$N_{\rm el} = 10, N_{\rm p} = 3 \rightarrow i_{\rm p}(1) = 1, i_{\rm p}(4) = 1, i_{\rm p}(5) = 2, \ldots$
REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
Adapted to changes in lastofset. (Aurich)
Changed behaviour if k is smaller of larger than the set. (Aurich)
Added sanity checks. (Aurich)
```

2.0.172 lastproc (Source File: modmpi.F90)

INTERFACE:

```
function lastproc(col, set, nprocs)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
integer(4) :: col ! ‘‘Column" of process grid (i.e. an element index of rank 0)
integer(4) :: set ! Total number of distributed elements.
integer(4), optional :: nprocs  ! Number of processes in communicator
OUT:
integer(4) :: lastproc ! Number of processes active in process column
```

DESCRIPTION:

This functions helps with collecting a set of N_{el} elements which were distributed to N_{p} `MPI` processes in continuous blocks and is used for example for the writing of `PMATXS.OUT`, `EMAT.OUT`, `SCCLI.OUT` and `EXCLI.OUT`. For further describing the functionality of this routine, let us consider an example distribution:
Let $N_{\mathrm{el}} = 13$ and $N_{\mathrm{p}} = 5$:

rank	firstofset	...	lastofset	nofset
0	1	2	3	3
1	4	5	6	3
2	7	6	9	3
3	10	11	-	2
4	12	13	-	2

For inputs of col $= \{1, 2, 3\}$, set $= 13$ the routine returns $\{4, 4, 2\}$, i.e. the process index of the last active process in the respective column. For all other input for col the routine halts execution. In the pathological case, where we have more processes than elements the following example depicts the routines behaviour:
Let $N_{\mathrm{el}} = 3$ and $N_{\mathrm{p}} = 5$:

rank	firstofset	...	lastofset	nofset
0	1	1	-	1
1	2	2	-	1
2	3	3	-	1
3	0	-1	-	0
4	0	-1	-	0

For inputs of col $= 1$, set $= 3$ the routine returns 2. For all other input for col execution is halted. In the other case pathological case, where we have only one processes the following example depicts the routines behaviour:
Let $N_{\mathrm{el}} = 3$ and $N_{\mathrm{p}} = 1$:

rank	firstofset	...	lastofset	nofset
0	1	2	3	3

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
Added sanity checks. (Aurich)
```

2.0.173 barrier (Source File: modmpi.F90)

INTERFACE:

```
subroutine barrier(mpicom, callername)
```

DESCRIPTION:

If -DMPI calls `mpi_barrier`, else nothing.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
```

2.0.174 mpi_allgatherv_ifc (Source File: modmpi.F90)

INTERFACE:

```
subroutine mpi_allgatherv_ifc(set, rlen, rlenv, ibuf, rlpbuf, rbuf, zbuf,&
  & inplace, comm)
```

INPUT/OUTPUT PARAMETERS:

```
In:
integer(4) :: set              ! Number of elements in distributed set
integer(4), optional :: rlen   ! Number of data elements per element (constant)
integer(4), optional :: rlenv(set) ! Number of data elements per element
logical, optional :: inplace   ! Use mpi_in_place
type(mpiifo), optional :: comm ! MPI communicator type
In/Out:
integer(4), optional :: ibuf(*) ! Buffers to send/recive
real(4), optional :: rlbuf(*)   ! for different data types
real(8), optional :: rbuf(*)    !
complex(8), optional :: zbuf(*) !
```

DESCRIPTION:

Wrapper routine for `MPI_ALLGATHERV` for different data types which is adapted for the k-point set distribution scheme. That is this works, if *set* number of elements (e.g. k-points) is distributed over all processes in the `MPI` communicator *comm* using a continuous distribution as created by the functions `nofset, firstofset, lastofset`. The routine can handle a constant number of data elements per set element by specifying `rlen` or a set element dependent number of data elements by passing `rlenv(set)`.

REVISION HISTORY:

```
Added to documentation scheme. (Aurich)
Added input parameter for communicator and
a switch for inplace allgather. (Aurich)
Added support for set element dependent number
of data elements. (Aurich)
```

2.0.175 setup_proc_groups (Source File: modmpi.F90)

INTERFACE:

```
subroutine setup_proc_groups(ngroups, mygroup)
```

INPUT/OUTPUT PARAMETERS:

```
IN:
input(4) :: ngroups ! number of groups to be formed
                    ! form the available MPI threads
OUT:
type(procgroup) :: mygroup ! Type containing MPI information
```

DESCRIPTION:

Given a total of N_p `MPI` processes this routine generates N_g process groups of size $N_\mathrm{p}/N_\mathrm{g}$ (no accutal `MPI` groups, but rather full `MPI` communicators). $N_\mathrm{p} \geq N_\mathrm{g}$ is required. If $\mathrm{mod}(N_\mathrm{p}, N_\mathrm{g}) \neq 0$ then one more group is created with the rest of the processes. Also a commuicator between the first processes in each such group is created.

REVISION HISTORY:

```
Based of setup_ProcGroups form mpi_mortadella branch. (Aurich)
```

2.0.176 setup_node_groups (Source File: modmpi.F90)

INTERFACE:

```
subroutine setup_node_groups
```

INPUT/OUTPUT PARAMETERS:

```
Module OUT:
type(procgroup) :: mpinodes ! Partitioning according to processor name
```

DESCRIPTION:

WARNING: This can only ever work if you pin the `MPI` threads to the processors during program execution.
Given a total of N_p `MPI` processes this routine generates N_nodes process groups. The size of each process group depends on the respective node size and node utilization. Also a communicator between the first processes in each node is created. A node in the context of this routine is a processor (return value of `mpi_get_processor_name`).

REVISION HISTORY:

```
Based of setup_ProcGroups form mpi_mortadella branch
and get_isfirstinnode form master. (Aurich)
```

2.0.177 mpisumrhoandmag (Source File: mpisumrhoandmag.F90)

INTERFACE:

```
Subroutine mpisumrhoandmag
#ifdef MPI
      Use modinput
```

USES:

```
      Use modmpi
      Use modmain
```

DESCRIPTION:

This subroutine adds up the partial sums for the charge density, the magnetisation, and the partial charges for all processors and broadcasts it to all processors.

REVISION HISTORY:

```
    Created October September 2006 (Christian Meisenbichler)
    Modifications, August 2010 (Stephan Sagmeister)
```

2.0.178 mpiresumeevec (Source File: mpiresumeevecfiles.F90)

INTERFACE:

```
Subroutine mpiresumeevecfiles
      Use modmain
#ifdef MPI
      Use modmpi
```

DESCRIPTION:

Routine reads the lokal EIGVECk1-k2.OUT files that were created to avoid file system inconsistencies. And writes them into the standard EIGVEC.OUT File

REVISION HISTORY:

```
    Created October SEPT 2006 (MULEOBEN)
    by Cristian Meisenbichler
```

2.0.179 xc_vbh (Source File: xc_vbh.f90)

INTERFACE:

```
subroutine xc_vbh(n,rhoup,rhodn,ex,ec,vxup,vxdn,vcup,vcdn)
```

INPUT/OUTPUT PARAMETERS:

```
n     : number of density points (in,integer)
rhoup : spin-up charge density (in,real(n))
rhodn : spin-down charge density (in,real(n))
ex    : exchange energy density (out,real(n))
ec    : correlation energy density (out,real(n))
vxup  : spin-up exchange potential (out,real(n))
vxdn  : spin-down exchange potential (out,real(n))
vcup  : spin-up correlation potential (out,real(n))
vcdn  : spin-down correlation potential (out,real(n))
```

DESCRIPTION:

Spin-polarised exchange-correlation potential and energy functional of von Barth and Hedin: *J. Phys. C* **5**, 1629 (1972). Note that the implementation is in Rydbergs in order to follow the paper step by step, at the end the potential and energy are converted to Hartree.

REVISION HISTORY:

```
Created September 2007 (F. Cricchio)
```

2.0.180 ggair_1 (Source File: ggair_1.f90)

INTERFACE:

```
subroutine ggair_1(grho, g2rho, g3rho)
```

USES:

```
use mod_Gvector
use mod_potential_and_density
```

DESCRIPTION:

Spin-unpolarised version of `ggair_sp_1`. The gradient $(\nabla\rho)\cdot\nabla(|\nabla\rho|)$ is evaluated using the relation

$$(\nabla\rho)\cdot\nabla(|\nabla\rho|) = \frac{(\nabla\rho)\cdot(\nabla\otimes\nabla\rho)\cdot(\nabla\rho)}{|\nabla\rho|}.$$

REVISION HISTORY:

```
Created November 2009 (JKD)
Modified third order gradients: improved numerical stability, April 2011
(S. Sagmeister)
```

2.0.181 xc_am05 (Source File: xc_am05.f90)

INTERFACE:

```
subroutine xc_am05(n,rho,grho,g2rho,g3rho,ex,ec,vx,vc)
```

INPUT/OUTPUT PARAMETERS:

```
n     : number of density points (in,integer)
rho   : charge density (in,real(n))
grho  : |grad rho| (in,real(n))
g2rho : grad^2 rho (in,real(n))
g3rho : (grad rho).(grad |grad rho|) (in,real(n))
ex    : exchange energy density (out,real(n))
ec    : correlation energy density (out,real(n))
vx    : spin-unpolarised exchange potential (out,real(n))
vc    : spin-unpolarised correlation potential (out,real(n))
```

DESCRIPTION:

Spin-unpolarised exchange-correlation potential and energy functional of R. Armiento and A. E. Mattsson, *Phys. Rev. B* **72**, 085108 (2005).

REVISION HISTORY:

```
Created April 2005 (RAR); based on xc_pbe
```

2.0.182 xc_am05_point (Source File: xc_am05.f90)

INTERFACE:

```
subroutine xc_am05_point(rho,s,u,v,ex,ec,vx,vc,pot)
```

INPUT/OUTPUT PARAMETERS:

```
rho : electron density (in,real)
s   : gradient of n / (2 kF n)
u   : grad n * grad | grad n | / (n**2 (2 kF)**3)
v   : laplacian of density / (n**2 (2.d0*kf)**3)
ex  : exchange energy density (out,real)
ec  : correlation energy density (out,real)
vx  : spin-unpolarised exchange potential (out,real)
vc  : spin-unpolarised correlation potential (out,real)
```

DESCRIPTION:

Calculate the spin-unpolarised exchange-correlation potential and energy for the Armiento-Mattsson 05 functional for a single point.

REVISION HISTORY:

```
Created April 2005 (RAR)
```

2.0.183 xc_am05_ldax (Source File: xc_am05.f90)

INTERFACE:

```
subroutine xc_am05_ldax(n,ex,vx)
```

INPUT/OUTPUT PARAMETERS:

```
n  : electron density (in,real)
ex : exchange energy per electron (out,real)
vx : exchange potential (out,real)
```

DESCRIPTION:

Local density approximation exchange.

REVISION HISTORY:

```
Created April 2005 (RAR)
```

2.0.184 xc_am05_ldapwc (Source File: xc_am05.f90)

INTERFACE:

```
subroutine xc_am05_ldapwc(n,ec,vc)
```

INPUT/OUTPUT PARAMETERS:

```
n  : electron density (in,real)
ec : correlation energy per electron (out,real)
vc : correlation potential (out,real)
```

DESCRIPTION:

Correlation energy and potential of the Perdew-Wang parameterisation of the Ceperley-Alder electron gas *Phys. Rev. B* **45**, 13244 (1992) and *Phys. Rev. Lett.* **45**, 566 (1980). This is a clean-room implementation from paper.

REVISION HISTORY:

```
Created April 2005 (RAR)
```

2.0.185 xc_am05_labertw (Source File: xc_am05.f90)

INTERFACE:

```
subroutine xc_am05_labertw(z,val)
```

INPUT/OUTPUT PARAMETERS:

```
z   : function argument (in,real)
val : value of lambert W function of z (out,real)
```

DESCRIPTION:

Lambert W-function using the method of Corless, Gonnet, Hare, Jeffrey and Knuth, *Adv. Comp. Math.* **5**, 329 (1996). The approach is based loosely on that in GNU Octave by N. N. Schraudolph, but this implementation is for real values and the principal branch only.

REVISION HISTORY:

```
Created April 2005 (RAR)
```

2.0.186 ggamt_2a (Source File: ggamt_2a.f90)

INTERFACE:

```
subroutine ggamt_2a(is, ia, g2rho, gvrho, grho2)
```

USES:

```
use modinput
use mod_Gvector
use mod_muffin_tin
use mod_SHT
use mod_atoms
use mod_potential_and_density
```

DESCRIPTION:

Spin-unpolarised version of `ggamt_sp_2a`.

REVISION HISTORY:

```
Created November 2009 (JKD and TMcQ)
```

2.0.187 ggamt_2b (Source File: ggamt_2b.f90)

INTERFACE:

```
subroutine ggamt_2b(is, g2rho, gvrho, vx, vc, dxdg2, dcdg2)
```

USES:

```
use modinput
use mod_Gvector
use mod_muffin_tin
use mod_SHT
use mod_atoms
```

DESCRIPTION:

Spin-unpolarised version of `ggamt_sp_2b`.

REVISION HISTORY:

```
Created November 2009 (JKD and TMcQ)
```

2.0.188 xc_pbe (Source File: xc_pbe.f90)

INTERFACE:

```
subroutine xc_pbe(n,kappa,mu,beta,rhoup,rhodn,grho,gup,gdn,g2up,g2dn,g3rho, &
 g3up,g3dn,ex,ec,vxup,vxdn,vcup,vcdn)
Use mod_potential_and_density, only: xctype
```

INPUT/OUTPUT PARAMETERS:

```
n     : number of density points (in,integer)
kappa : parameter for large-gradient limit (in,real)
mu    : gradient expansion coefficient (in,real)
beta  : gradient expansion coefficient (in,real)
rhoup : spin-up charge density (in,real(n))
rhodn : spin-down charge density (in,real(n))
grho  : |grad rho| (in,real(n))
gup   : |grad rhoup| (in,real(n))
gdn   : |grad rhodn| (in,real(n))
g2up  : grad^2 rhoup (in,real(n))
g2dn  : grad^2 rhodn (in,real(n))
g3rho : (grad rho).(grad |grad rho|) (in,real(n))
g3up  : (grad rhoup).(grad |grad rhoup|) (in,real(n))
g3dn  : (grad rhodn).(grad |grad rhodn|) (in,real(n))
ex    : exchange energy density (out,real(n))
ec    : correlation energy density (out,real(n))
vxup  : spin-up exchange potential (out,real(n))
vxdn  : spin-down exchange potential (out,real(n))
vcup  : spin-up correlation potential (out,real(n))
vcdn  : spin-down correlation potential (out,real(n))
```

DESCRIPTION:

Spin-polarised exchange-correlation potential and energy of the generalised gradient approximation functional of J. P. Perdew, K. Burke and M. Ernzerhof *Phys. Rev. Lett.* **77**, 3865 (1996) and **78**, 1396(E) (1997). The parameter κ, which controls the large-gradient limit, can be set to 0.804 or 1.245 corresponding to the value in the original article or the revised version of Y. Zhang and W. Yang, *Phys. Rev. Lett.* **80**, 890 (1998).

REVISION HISTORY:

```
Modified routines written by K. Burke, October 2004 (JKD)
```

2.0.189 ggamt_sp_1 (Source File: ggamt_sp_1.f90)

INTERFACE:

```
subroutine ggamt_sp_1(is, rhoup, rhodn, grho, gup, gdn, g2up, g2dn, g3rho, g3up, g3dn)
```

USES:

```
use modinput
use mod_Gvector
use mod_muffin_tin
use mod_SHT
use mod_atoms
use mod_potential_and_density
```

INPUT/OUTPUT PARAMETERS:

```
is    : species number (in,integer)
rhoup : spin-up density in spherical coordinates (in,real(lmmaxvr,nrmtmax))
rhodn : spin-down density (in,real(lmmaxvr,nrmtmax))
grho  : |grad rho| (out,real(lmmaxvr,nrmtmax))
gup   : |grad rhoup| (out,real(lmmaxvr,nrmtmax))
gdn   : |grad rhodn| (out,real(lmmaxvr,nrmtmax))
g2up  : grad^2 rhoup (out,real(lmmaxvr,nrmtmax))
g2dn  : grad^2 rhodn (out,real(lmmaxvr,nrmtmax))
g3rho : (grad rho).(grad |grad rho|) (out,real(lmmaxvr,nrmtmax))
g3up  : (grad rhoup).(grad |grad rhoup|) (out,real(lmmaxvr,nrmtmax))
g3dn  : (grad rhodn).(grad |grad rhodn|) (out,real(lmmaxvr,nrmtmax))
```

DESCRIPTION:

Computes $|\nabla\rho|$, $|\nabla\rho^{\uparrow}|$, $|\nabla\rho^{\downarrow}|$, $\nabla^2\rho^{\uparrow}$, $\nabla^2\rho^{\downarrow}$, $\nabla\rho\cdot(\nabla|\nabla\rho|)$, $\nabla\rho^{\uparrow}\cdot(\nabla|\nabla\rho^{\uparrow}|)$ and $\nabla\rho^{\downarrow}\cdot(\nabla|\nabla\rho^{\downarrow}|)$ for a muffin-tin charge density, as required by the generalised gradient approximation functionals of type 1 for spin-polarised densities. The input densities and output gradients are in terms of spherical coordinates. See routines `potxc` and `modxcifc`.

REVISION HISTORY:

```
Created April 2004 (JKD)
Simplified and improved, October 2009 (JKD)
```

2.0.190 xc_xalpha (Source File: xc_xalpha.f90)

INTERFACE:

```
subroutine xc_xalpha(n,rho,exc,vxc)
```

INPUT/OUTPUT PARAMETERS:

```
n   : number of density points (in,integer)
rho : charge density (in,real(n))
exc : exchange-correlation energy density (out,real(n))
vxc : exchange-correlation potential (out,real(n))
```

DESCRIPTION:

X_α approximation to the exchange-correlation potential and energy density. See J. C. Slater, *Phys. Rev.* **81**, 385 (1951).

REVISION HISTORY:

```
Modified an ABINIT routine, September 2006 (JKD)
```

2.0.191 xc_pwca (Source File: xc_pwca.f90)

INTERFACE:

```
subroutine xc_pwca(n,rhoup,rhodn,ex,ec,vxup,vxdn,vcup,vcdn)
```

INPUT/OUTPUT PARAMETERS:

```
n     : number of density points (in,integer)
rhoup : spin-up charge density (in,real(n))
rhodn : spin-down charge density (in,real(n))
ex    : exchange energy density (out,real(n))
ec    : correlation energy density (out,real(n))
vxup  : spin-up exchange potential (out,real(n))
vxdn  : spin-down exchange potential (out,real(n))
vcup  : spin-up correlation potential (out,real(n))
vcdn  : spin-down correlation potential (out,real(n))
```

DESCRIPTION:

Spin-polarised exchange-correlation potential and energy of the Perdew-Wang parameterisation of the Ceperley-Alder electron gas: *Phys. Rev. B* **45**, 13244 (1992) and *Phys. Rev. Lett.* **45**, 566 (1980).

REVISION HISTORY:

```
Created January 2004 (JKD)
```

2.0.192 ggamt_sp_2a (Source File: ggamt_sp_2a.f90)

INTERFACE:

```
subroutine ggamt_sp_2a(is, rhoup, rhodn, g2up, g2dn, gvup, gvdn, gup2, gdn2, gupdn)
```

USES:

```
use modinput
use mod_Gvector
use mod_muffin_tin
use mod_SHT
use mod_atoms
use mod_potential_and_density
```

DESCRIPTION:

Computes the muffin-tin gradients $\nabla^2\rho^\uparrow$, $\nabla^2\rho^\downarrow$, $\nabla\rho^\uparrow$, $\nabla\rho^\downarrow$, $(\nabla\rho^\uparrow)^2$, $(\nabla\rho^\downarrow)^2$ and $\nabla\rho^\uparrow\cdot\nabla\rho^\downarrow$, which are passed in to GGA functional subroutines of type 2. The exchange-correlation energy in these routines has the functional form

$$E_{xc}[\rho^\uparrow,\rho^\downarrow]=\int d^3r\,\hat{\epsilon}_{xc}\big(\rho^\uparrow(\mathbf{r}),\rho^\downarrow(\mathbf{r}),(\nabla\rho^\uparrow(\mathbf{r}))^2,(\nabla\rho^\downarrow(\mathbf{r}))^2,\nabla\rho^\uparrow(\mathbf{r})\cdot\nabla\rho^\downarrow(\mathbf{r})\big),$$

where $\hat{\epsilon}_{xc}(\mathbf{r}) = \epsilon_{xc}(\mathbf{r})\rho(\mathbf{r})$ is the xc energy per unit volume, with ϵ_{xc} being the xc energy per electron, and $\rho = \rho^{\uparrow} + \rho^{\downarrow}$. From the gradients above, type 2 GGA routines return ϵ_{xc}, but not directly the xc potentials. Instead they generate the derivatives $\partial\hat{\epsilon}_{xc}/\partial\rho^{\uparrow}(\mathbf{r})$, $\partial\hat{\epsilon}_{xc}/\partial(\nabla\rho^{\uparrow}(\mathbf{r}))^2$, and the same for down spin, as well as $\partial\hat{\epsilon}_{xc}/\partial(\nabla\rho^{\uparrow}(\mathbf{r})\cdot\nabla\rho^{\downarrow}(\mathbf{r}))$. In a post-processing step invoked by `ggamt_sp_2b`, integration by parts is used to obtain the xc potential explicitly with

$$V_{xc}^{\uparrow}(\mathbf{r}) = \frac{\partial\hat{\epsilon}_{xc}}{\partial\rho^{\uparrow}(\mathbf{r})} - 2\left(\nabla\frac{\partial\hat{\epsilon}_{xc}}{\partial(\nabla\rho^{\uparrow})^2}\right)\cdot\nabla\rho^{\uparrow} - 2\frac{\hat{\epsilon}_{xc}}{\partial(\nabla\rho^{\uparrow})^2}\nabla^2\rho^{\uparrow}\& \quad - \left(\nabla\frac{\hat{\epsilon}_{xc}}{\partial(\nabla\rho^{\uparrow}\cdot\nabla\rho^{\downarrow})}\right)\cdot\nabla\rho^{\downarrow} - \frac{\partial\hat{\epsilon}_{xc}}{\partial(\nabla\rho^{\uparrow}\cdot\nabla\rho^{\downarrow})}\nabla^2\rho^{\downarrow},$$

and similarly for $V_{xc}^{\downarrow}$.

REVISION HISTORY:

Created November 2009 (JKD and TMcQ)

2.0.193 ggamt_sp_2b (Source File: ggamt_sp_2b.f90)

INTERFACE:

```
subroutine ggamt_sp_2b(is, g2up, g2dn, gvup, gvdn, vxup, vxdn, vcup, vcdn, dxdgu2, &
 dxdgd2, dxdgud, dcdgu2, dcdgd2, dcdgud)
```

USES:

```
use modinput
use mod_Gvector
use mod_muffin_tin
use mod_SHT
use mod_atoms
```

DESCRIPTION:

Post processing step of muffin-tin gradients for GGA type 2. See routine `ggamt_sp_2a` for full details.

REVISION HISTORY:

Created November 2009 (JKD and TMcQ)

2.0.194 ggair_sp_2a (Source File: ggair_sp_2a.f90)

INTERFACE:

```
subroutine ggair_sp_2a(rhoup, rhodn, g2up, g2dn, gvup, gvdn, gup2, gdn2, gupdn)
```

USES:

```
use modinput
use mod_Gvector
```

DESCRIPTION:

Computes the interstitial gradients $\nabla^2\rho^{\uparrow}$, $\nabla^2\rho^{\downarrow}$, $\nabla\rho^{\uparrow}$, $\nabla\rho^{\downarrow}$, $(\nabla\rho^{\uparrow})^2$, $(\nabla\rho^{\downarrow})^2$ and $\nabla\rho^{\uparrow}\cdot\nabla\rho^{\downarrow}$. These are used for GGA functionals of type 2. See `ggamt_sp_2a` for full details.

REVISION HISTORY:

```
Created November 2009 (JKD and TMcQ)
```

2.0.195 ggair_sp_2b (Source File: ggair_sp_2b.f90)

INTERFACE:

```
subroutine ggair_sp_2b(g2up, g2dn, gvup, gvdn, vxup, vxdn, vcup, vcdn, dxdgu2, dxdgd2, &
 dxdgud, dcdgu2, dcdgd2, dcdgud)
use modinput
```

USES:

```
use mod_Gvector
```

DESCRIPTION:

Post processing step of interstitial gradients for GGA type 2. See routine `ggamt_sp_2a` for full details.

REVISION HISTORY:

```
Created November 2009 (JKD and TMcQ)
```

2.0.196 ggair_2a (Source File: ggair_2a.f90)

INTERFACE:

```
subroutine ggair_2a(g2rho, gvrho, grho2)
```

USES:

```
use modinput
use mod_Gvector
use mod_potential_and_density
```

DESCRIPTION:

Spin-unpolarised version of `ggair_sp_2a`.

REVISION HISTORY:

```
Created November 2009 (JKD and TMcQ)
```

2.0.197 ggair_2b (Source File: ggair_2b.f90)

INTERFACE:

```
subroutine ggair_2b(g2rho, gvrho, vx, vc, dxdg2, dcdg2)
```

USES:

```
use modinput
use mod_Gvector
use mod_potential_and_density
```

DESCRIPTION:

Spin-unpolarised version of ggair_sp_2b.

REVISION HISTORY:

```
Created November 2009 (JKD and TMcQ)
```

2.0.198 xcifc_libxc (Source File: libxcifc.f90)

INTERFACE:

```
subroutine xcifc_libxc(xctype,n,rho,rhoup,rhodn,grho2,gup2,gdn2,gupdn,ex,ec, &
 vx,vc,vxup,vxdn,vcup,vcdn,dxdg2,dxdgu2,dxdgd2,dxdgud,dcdg2,dcdgu2,dcdgd2, &
 dcdgud)
```

INPUT/OUTPUT PARAMETERS:

```
xctype : type of exchange-correlation functional (in,integer(3))
n      : number of density points (in,integer)
rho    : spin-unpolarised charge density (in,real(n),optional)
rhoup  : spin-up charge density (in,real(n),optional)
rhodn  : spin-down charge density (in,real(n),optional)
grho2  : |grad rho|^2 (in,real(n),optional)
gup2   : |grad rhoup|^2 (in,real(n),optional)
gdn2   : |grad rhodn|^2 (in,real(n),optional)
gupdn  : (grad rhoup).(grad rhodn) (in,real(n),optional)
ex     : exchange energy density (out,real(n),optional)
ec     : correlation energy density (out,real(n),optional)
vx     : spin-unpolarised exchange potential (out,real(n),optional)
vc     : spin-unpolarised correlation potential (out,real(n),optional)
vxup   : spin-up exchange potential (out,real(n),optional)
vxdn   : spin-down exchange potential (out,real(n),optional)
vcup   : spin-up correlation potential (out,real(n),optional)
vcdn   : spin-down correlation potential (out,real(n),optional)
```

```
dxdg2  : de_x/d(|grad rho|^2) (out,real(n),optional)
dxdgu2 : de_x/d(|grad rhoup|^2) (out,real(n),optional)
dxdgd2 : de_x/d(|grad rhodn|^2) (out,real(n),optional)
dxdgud : de_x/d((grad rhoup).(grad rhodn)) (out,real(n),optional)
dcdg2  : de_c/d(|grad rho|^2) (out,real(n),optional)
dcdgu2 : de_c/d(|grad rhoup|^2) (out,real(n),optional)
dcdgd2 : de_c/d(|grad rhodn|^2) (out,real(n),optional)
dcdgud : de_c/d((grad rhoup).(grad rhodn)) (out,real(n),optional)
```

DESCRIPTION:

Interface to the `libxc` exchange-correlation functional library:
`http://www.tddft.org/programs/octopus/wiki/index.php/Libxc`. The second and third integers in `xctype` define the exchange and correlation functionals in `libxc`, respectively.

REVISION HISTORY:

```
Created April 2009 (Tyrel McQueen)
Modified September 2009 (JKD and TMQ)
Modified September 2012 (UW)
Modified Januar 2013(UW)
Modified Oktober 2013 (UW)
```

2.0.199 xcifc (Source File: modxcifc.f90)

INTERFACE:

```
subroutine xcifc(xctype,n,rho,rhoup,rhodn,grho,gup,gdn,g2rho,g2up,g2dn,g3rho, &
 g3up,g3dn,grho2,gup2,gdn2,gupdn,ex,ec,vx,vc,vxup,vxdn,vcup,vcdn,dxdg2,dxdgu2, &
 dxdgd2,dxdgud,dcdg2,dcdgu2,dcdgd2,dcdgud)
```

INPUT/OUTPUT PARAMETERS:

```
xctype : type of exchange-correlation functional (in,integer(3))
n      : number of density points (in,integer)
rho    : spin-unpolarised charge density (in,real(n),optional)
rhoup  : spin-up charge density (in,real(n),optional)
rhodn  : spin-down charge density (in,real(n),optional)
grho   : |grad rho| (in,real(n),optional)
gup    : |grad rhoup| (in,real(n),optional)
gdn    : |grad rhodn| (in,real(n),optional)
g2rho  : grad^2 rho (in,real(n),optional)
g2up   : grad^2 rhoup (in,real(n),optional)
g2dn   : grad^2 rhodn (in,real(n),optional)
g3rho  : (grad rho).(grad |grad rho|) (in,real(n),optional)
g3up   : (grad rhoup).(grad |grad rhoup|) (in,real(n),optional)
g3dn   : (grad rhodn).(grad |grad rhodn|) (in,real(n),optional)
grho2  : |grad rho|^2 (in,real(n),optional)
gup2   : |grad rhoup|^2 (in,real(n),optional)
```

```
gdn2   : |grad rhodn|^2 (in,real(n),optional)
gupdn  : (grad rhoup).(grad rhodn) (in,real(n),optional)
ex     : exchange energy density (out,real(n),optional)
ec     : correlation energy density (out,real(n),optional)
vx     : spin-unpolarised exchange potential (out,real(n),optional)
vc     : spin-unpolarised correlation potential (out,real(n),optional)
vxup   : spin-up exchange potential (out,real(n),optional)
vxdn   : spin-down exchange potential (out,real(n),optional)
vcup   : spin-up correlation potential (out,real(n),optional)
vcdn   : spin-down correlation potential (out,real(n),optional)
dxdg2  : de_x/d(|grad rho|^2) (out,real(n),optional)
dxdgu2 : de_x/d(|grad rhoup|^2) (out,real(n),optional)
dxdgd2 : de_x/d(|grad rhodn|^2) (out,real(n),optional)
dxdgud : de_x/d((grad rhoup).(grad rhodn)) (out,real(n),optional)
dcdg2  : de_c/d(|grad rho|^2) (out,real(n),optional)
dcdgu2 : de_c/d(|grad rhoup|^2) (out,real(n),optional)
dcdgd2 : de_c/d(|grad rhodn|^2) (out,real(n),optional)
dcdgud : de_c/d((grad rhoup).(grad rhodn)) (out,real(n),optional)
```

DESCRIPTION:

Interface to the exchange-correlation routines.

REVISION HISTORY:

```
Created October 2002 (JKD)
Modiefied Januar 2013 (UW)
```

2.0.200 getxcdata (Source File: modxcifc.f90)

INTERFACE:

```
subroutine getxcdata(xctype,xcdescr,xcspin,xcgrad,ex_coef)
```

INPUT/OUTPUT PARAMETERS:

```
xctype  : type of exchange-correlation functional (in,integer(3))
xcdescr : description of functional (out,character(256))
xcspin  : spin treatment (out,integer)
xcgrad  : gradient treatment (out,integer)
```

DESCRIPTION:

Returns data on the exchange-correlation functional labeled by `xctype`. The character array `xcdescr` contains a short description of the functional including journal references. The variable `xcspin` is set to 1 or 0 for spin-polarised or -unpolarised functionals, respectively. For functionals which require the gradients of the density `xcgrad` is set to 1, otherwise it is set to 0.

REVISION HISTORY:

```
Created October 2002 (JKD)
```

2.0.201 spline (Source File: splline4.f90)

INTERFACE:

```
subroutine spline4(n,x,ld,f,cf)
```

INPUT/OUTPUT PARAMETERS:

```
n  : number of points (in,integer)
x  : abscissa array (in,real(n))
ld : leading dimension (in,integer)
f  : input data array (in,real(ld,n))
cf : cubic spline coefficients (1,2,3) and work space (4) (out,real(4,n))
```

DESCRIPTION:

Calculates the coefficients of a cubic spline fitted to input data. In other words, given a set of data points f_i defined at x_i, where $i = 1 \ldots n$, the coefficients c_j^i are determined such that

$$y_i(x) = f_i + c_1^i(x - x_i) + c_2^i(x - x_i)^2 + c_3^i(x - x_i)^3,$$

is the interpolating function for $x \in [x_i, x_{i+1})$. This is done by determining the end-point coefficients c_2^1 and c_2^n from the first and last three points, and then solving the tridiagonal system

$$d_{i-1}c_2^{i-1} + 2(d_{i-1} + d_i)c_2^i + d_i c_2^{i+1} = 3\left(\frac{f_{i+1} - f_i}{d_i} - \frac{f_i - f_{i-1}}{d_{i-1}}\right),$$

where $d_i = x_{i+1} - x_i$, for the intermediate coefficients.

REVISION HISTORY:

```
Created October 2004 (JKD)
Improved speed and accuracy, April 2006 (JKD)
Optimisations and improved end-point coefficients, February 2008 (JKD)
```

2.0.202 ggamt_1 (Source File: ggamt_1.f90)

INTERFACE:

```
subroutine ggamt_1(is, ia, grho, g2rho, g3rho)
```

USES:

```
use modinput
use mod_Gvector
use mod_muffin_tin
use mod_SHT
use mod_atoms
use mod_potential_and_density
```

DESCRIPTION:

Spin-unpolarised version of `ggamt_sp_1`.

REVISION HISTORY:

```
Created November 2009 (JKD)
Modified Mai 2012 (UW)
```

2.0.203 ggair_sp_1 (Source File: ggair_sp_1.f90)

INTERFACE:

```
subroutine ggair_sp_1(rhoup, rhodn, grho, gup, gdn, g2up, g2dn, g3rho, g3up, g3dn)
```

INPUT/OUTPUT PARAMETERS:

```
use mod_Gvector
    rhoup : spin-up density (in,real(ngrtot))
    rhodn : spin-down density (in,real(ngrtot))
    grho  : |grad rho| (out,real(ngrtot))
    gup   : |grad rhoup| (out,real(ngrtot))
    gdn   : |grad rhodn| (out,real(ngrtot))
    g2up  : grad^2 rhoup (out,real(ngrtot))
    g2dn  : grad^2 rhodn (out,real(ngrtot))
    g3rho : (grad rho).(grad |grad rho|) (out,real(ngrtot))
    g3up  : (grad rhoup).(grad |grad rhoup|) (out,real(ngrtot))
    g3dn  : (grad rhodn).(grad |grad rhodn|) (out,real(ngrtot))
```

DESCRIPTION:

Computes $|\nabla\rho|$, $|\nabla\rho^{\uparrow}|$, $|\nabla\rho^{\downarrow}|$, $\nabla^2\rho^{\uparrow}$, $\nabla^2\rho^{\downarrow}$, $\nabla\rho\cdot(\nabla|\nabla\rho|)$, $\nabla\rho^{\uparrow}\cdot(\nabla|\nabla\rho^{\uparrow}|)$ and $\nabla\rho^{\downarrow}\cdot(\nabla|\nabla\rho^{\downarrow}|)$ for the interstitial charge density, as required by the generalised gradient approximation functionals of type 1 for spin-polarised densities. See routines `ggair_1` `potxc` and `modxcifc`.

REVISION HISTORY:

```
Created October 2004 (JKD)
Simplified and improved, October 2009 (JKD)
Modified third order gradients: improved numerical stability, April 2011
(S. Sagmeister)
```

2.0.204 xc_pzca (Source File: xc_pzca.f90)

INTERFACE:

```
subroutine xc_pzca(n,rho,ex,ec,vx,vc)
```

INPUT/OUTPUT PARAMETERS:

```
    n   : number of density points (in,integer)
    rho : charge density (in,real(n))
    ex  : exchange energy density (out,real(n))
    ec  : correlation energy density (out,real(n))
    vx  : exchange potential (out,real(n))
    vc  : correlation potential (out,real(n))
```

DESCRIPTION:

Spin-unpolarised exchange-correlation potential and energy of the Perdew-Zunger parameterisation of Ceperley-Alder electron gas: *Phys. Rev. B* **23**, 5048 (1981) and *Phys. Rev. Lett.* **45**, 566 (1980).

REVISION HISTORY:

```
Created October 2002 (JKD)
```

2.0.205 loadinputDOM (Source File: modinputdom.f90)

INTERFACE:

```
Subroutine loadinputDOM (deffilename)
```

INPUT/OUTPUT PARAMETERS:

```
deffilename: default filename if no file name argument is in  argv()
    implicit none

    character(*),intent(in)::deffilename
```

DESCRIPTION:

Loads the contents of the inputfile into the DOM and initializes some data that is used by getstructinput(inputnp)

2.0.206 handleunknownnodes (Source File: modinputdom.f90)

INTERFACE:

```
Subroutine handleunknownnodes (np)
```

INPUT/OUTPUT PARAMETERS:

```
np: node pointer type containing node with all
    known nodes removed, thus containing only unknown that need to be reported.
```

DESCRIPTION:

This writes the error message when the getstruct... function sees an unknown (illegal) entry

---

INTERFACE:

```
Subroutine gensymcmut (eps, maxsymcrys, nsymcrys, symlat, lsplsymc, &
& vtlsymc, scmut, tabel, tspainvsym)
```

DESCRIPTION:

Sets up the group multiplication table. The table is checked for consistency in a way that it is required that every elements occurs once and only once in each row and column of the table. The first row and colmuns must consist of the indentity since the first symmetry element is the identity by convention.

REVISION HISTORY:

```
Created July 2008 (Sagmeister)
```

2.0.207 calc_vnlmat (Source File: calc_vnlmat.f90)

INTERFACE:

```
subroutine calc_vnlmat
```

USES:

```
use modmain
use modgw
use modfvsystem
use mod_hybrids
use modmpi
```

DESCRIPTION:

Calculates the APW matrix elements of the non-local potential

---

2.0.208 updateradial (Source File: updateradial.f90)

INTERFACE:

```
Subroutine updateradial
```

USES:

```
Use modmain
```

DESCRIPTION:

Update of radial functions during the Hartree-Fock hybrids run.

REVISION HISTORY:

```
Created July 2015 (UW)
```

2.0.209 calc_vxnl (Source File: calc_vxnl.f90)

INTERFACE:

```
subroutine calc_vxnl()
```

USES:

```
use modmain
use modgw
use modfvsystem
use mod_hybrids
use modmpi
```

DESCRIPTION:

Calculates the non-local exchange potential and the non-local exchange energy for Hartree-Fock based hybrid functionals.

2.0.210 hybrids (Source File: hybrids.f90)

INTERFACE:

```
Subroutine hybrids
```

USES:

```
Use modinput
Use modmain
Use modmpi
Use scl_xml_out_Module
Use mod_hybrids
```

DESCRIPTION:

Main routine for Hartree-Fock based hybrid functionals.
REVISION HISTORY:

```
Created September 2013 (DIN)
Modified October 2013
Modified Februar 2014
```

2.0.211 putvnlmat (Source File: putvnlmat.f90)

INTERFACE:

```
subroutine putvnlmat()
```

USES:

```
use modmain
use mod_hybrids
use modmpi
```

DESCRIPTION:

Writes the APW matrix elements of the non-local potential into the file VNLMAT.OUT
REVISION HISTORY:

```
Created March 2015 (UW)
Changed May 2016 (DIN)
```

!ROUTINE: setumix
INTERFACE:

```
subroutine setumix(ia,is)
```

DESCRIPTION:

Set up the radial part of the mixed basis functions (υ_{aNL})used for the matrix expansion of the non local operators (Polarization, Bare and Screened Coulomb potential, and the Self energy) for atom `ias`. The procedure is as follows:

- For each L we take the product of radial functions $u_{lm}(r)u_{l'm'}(r)$ which fullfil the condition $|l-l'| \leq L \leq l+l'$.
- We calculate the overlap matrix of the products of radial functions:

$$\mathbb{O}_{(l,l');(l_1,l'_1)} = \int_0^{R^a_{MT}} u_{al}u_{al'}(\vec{r})u_{al_1}u_{al'_1}(\vec{r})r^2dr \tag{11}$$

- Diagonalize the matrix $\mathbb{O}_{(lm,l'm');(l_1m_1,l'_1m'_1)}$
- Discard the eigenvectors corresponding to eigenvalues with absolute value lower than a given tolerance (usually 10^{-5})
- The rest of the eigenvectors are normalized and stored for a grid of $\vec{r}$ that constitute the new basis $\{\upsilon_{NL}\}$

So defined the set of functions $\{\gamma_{aNLM} = \upsilon_{aNL}Y_{LM}\}$ constitute and orthonormal basis set, that is:

$$\int_{V^a_{MT}} \gamma_{aNLM}(\vec{r})\gamma_{aN'L'M'}(\vec{r})d^3r = \delta_{N,N'}\delta_{L,L'}\delta_{M,M'} \tag{12}$$

USES:

```
use modinput
use modmain
use modmpi
use modgw
```

```
    use reallocate

  !INPUT VARIABLES:
    implicit none
    integer(4), intent(in) :: ia
    integer(4), intent(in) :: is
```

LOCAL VARIABLES:

```
    integer(4) :: ias
    integer(4) :: ir,i,j,k,l
    integer(4) :: nwf
    integer(4) :: nl_,nli,nor
    integer(4) :: lammax,lammin
    integer(4) :: i1,i2,info,j2
    integer(4) :: nl(0:2*maxlapw)
    real(8) :: fr(nrmtmax)
    real(8) :: gr(nrmtmax)
    real(8) :: cf(3,nrmtmax)

    integer(4), allocatable :: ind(:)
    real(8),    allocatable :: uol(:), work(:)
    real(8),    allocatable :: uml(:,:)

  !EXTERNAL ROUTINES:
    external dsyev
```

REVISION HISTORY:

```
  Created May. 19th. 2004 by RGA
  Last Modified May 25th 2004 by RGA
  Revisited 28.04.2011 by DIN
  Modified Dec 2013 by DIN
```

!ROUTINE: calcinveps
INTERFACE:

```
subroutine calcinveps(iomstart,iomend)
```

DESCRIPTION:

This subroutine calculates the inverse of the dielectric function
USES:

```
    use modmain
    use modgw
    use mod_mpi_gw, only : myrank
```

INPUT PARAMETERS:

```
implicit none
integer(4), intent(in) :: iomstart, iomend
```

LOCAL VARIABLES:

```
integer(4) :: i, j, iom
integer(4) :: im, jm
integer(4) :: info, lwork

real(8)    :: tstart, tend

complex(8), allocatable :: b(:,:)
integer(4), allocatable :: ipiv(:)
complex(8), allocatable :: work(:)

!EXTERNAL ROUTINES:
external zgetrf, zgetri
external zhetrf, zhetri
```

REVISION HISTORY:

```
Created 31.01.2007 by JH
Revisited Dec 2011 by DIN
```

!ROUTINE: `task_gw`
INTERFACE:

```
subroutine task_gw()
```

DESCRIPTION:

This subroutine performs one GW cycle and calculates the corresponding quasiparticle energies.
USES:

```
use modinput
use modmain,               only : zzero, evalsv, efermi
use modgw
use mod_mpi_gw
use m_getunit
use mod_hdf5
```

LOCAL VARIABLES:

```
implicit none
integer(4) :: ikp, iq, fid, ik
real(8)    :: t0, t1
integer(4) :: recl
real(8)    :: ab_plane, ab_norm(3), q0_vol
integer    :: im
complex(8) :: vc
```

REVISION HISTORY:

Created Nov 2013 by (DIN)

!ROUTINE: calcevalqp
INTERFACE:

```
subroutine calcevalqp
```

DESCRIPTION:

Given the matrix elements $\langle\Psi_{n\vec{k}}|\Sigma(\vec{k},\omega)|\Psi_{n\vec{k}}\rangle$, $\langle\Psi_{n\vec{k}}|V^{xc}|\Psi_{n\vec{k}}\rangle$ and $\varepsilon^{DFT}_{n\vec{k}}$. this subroutine calculates the quasi-particle energies $\varepsilon^{qp}_{n\vec{k}}$
USES:

```
use modinput
use modmain, only : evalsv, efermi, zzero
use modgw,   only : ibgw, nbgw, kset, evalqp, eferqp, &
&                   sigc, znorm, selfex, selfec, vxcnn, &
&                   nbandsgw, nvelgw, &
&                   sigsx, sigch, fgw

implicit none
integer :: nb, ie, ik
real(8) :: egap, df
```

REVISION HISTORY:

Created: 16.08.05 by RGA
Revisited June 2011 by DIN

!ROUTINE: calcgauntcoef **INTERFACE:**

```
subroutine calcgauntcoef(maxj)
```

DESCRIPTION:

This subroutine calculates the gaunt coefficients:

$$\mathcal{G}^{LM}_{ll',mm'} = \int_0^{2\pi}\int_0^{\pi} Y_{lm}(\theta,\phi)Y_{l'm'}(\theta,\phi)Y^*_{LM}(\theta,\phi)\sin(\theta)d\theta d\phi \tag{13}$$

for l and $l' = 0, 1, ...$ `maxj`. The integral is done numerically on a special grid (see gaunt.f90). The values are calculated only for $l \geq l'$ and $m' \geq 0$.
The storage is optimized by saving the values in a vector (`gauntcoef`) only for those coefficients that are different from zero. The size of the vector is:

$$n = \tfrac{1}{60}(l_{max}+1)(l_{max}+2)(l_{max}+3)(16l^2_{max}+29l_{max}+10) \tag{14}$$

The gaunt coefficient $\mathcal{G}^{LM}_{ll',mm'}$ can be accesed directly at `gauntcoef(i)` by applying the function:

$$\begin{aligned} i =& \tfrac{1}{60}(16l^2-3l-3)(l+2)(l+1)l + \tfrac{1}{3}ll'(l'+1)(4l'-1) + \tfrac{1}{6}l'(l'-1)(4l'+7) + \\ & (2l+1)(l'+1)(L-l+l') + (l'+1)(m+l) + m' + l' + 1 \end{aligned} \tag{15}$$

of course, $M = m + m'$ is already taken into account. For the cases $l' > l$ and $m' < 0$ see the `getcgcoef` subroutine.

INPUT PARAMETERS:

```
implicit none
integer(4), intent(in) :: maxj
```

LOCAL VARIABLES:

```
integer(4) :: i
integer(4) :: l1, l2,  l3
integer(4) :: m1, m2, m3
integer(4) :: ntot, ngrid

!EXTERNAL ROUTINES:
real(8), external :: gaunt
```

REVISION HISTORY:

```
Created: Apr. 2004 by RGA
Last modified: May 21st. 2004 by RGA
Adapted: Nov 2013 by DIN
```

!ROUTINE: getgauntcoef **INTERFACE:**

```
real(8) function getgauntcoef(l1,l2,l3,m1,m2)
```

DESCRIPTION:

This function gets the gaunt coefficient $G^{l3,m1+m2}_{l1,l2,m1,m2}$ from the vector `gauntcoef` by:

$$G^{l3,m1+m2}_{l1,l2,m1,m2} = \alpha \texttt{cgcoef}(i) \tag{16}$$

calculating i by:

$$\begin{aligned} i =& \tfrac{1}{60}(16l^2 - 3l - 3)(l+2)(l+1)l + \tfrac{1}{3}ll'(l'+1)(4l'-1) + \tfrac{1}{6}l'(l'-1)(4l'+7) + \\ &+ (2l+1)(l'+1)(L-l+l') + (l'+1)(m+l) + m' + l' + 1 \end{aligned} \tag{17}$$

where

$$l = l1 \quad l' = l2 \quad m = m1 \quad m' = m2 \quad \alpha = 1 \quad \text{if } l1 \geq l2 \tag{18a}$$

$$l = l2 \quad l' = l1 \quad m = m2 \quad m' = m1 \quad \alpha = 1 \quad \text{if } l1 < l2 \tag{18b}$$

$$m = m1 \quad m' = m2 \quad \alpha = 1 \quad \text{if } m2 \geq 0 \tag{18c}$$

$$m = -m1 \quad m' = -m2 \quad \alpha = (-1)^{l+l'-L} \quad \text{if } m2 < 0 \tag{18d}$$

INPUT PARAMETERS:

```
implicit none
integer(4), intent(in) :: l1
integer(4), intent(in) :: l2
integer(4), intent(in) :: l3
integer(4), intent(in) :: m1
integer(4), intent(in) :: m2
```

LOCAL VARIABLES:

```
integer(4) :: j1, j2, mj1, mj2
integer(4) :: par, ing
integer(4) :: ind1, ind2, ind3, ind4
real(8) :: fact
logical :: trcond
```

REVISION HISTORY:

```
Created: Apr. 2004 by RGA
Last modified  May 21st. 2004 by RGA
Adapted: Nov 2013 by DIN
```

!ROUTINE: `task_gw`
INTERFACE:

```
subroutine task_eph()
```

DESCRIPTION:

This subroutine performs one GW cycle and calculates the corresponding quasiparticle energies.

USES:

```
  Use modinput
  Use modmain
  use modgw
  use mod_mpi_gw
  use mod_wannier
  Use FoX_wxml
  use m_wsweight
  use m_getunit
  use m_plotmat
  use mod_wfint
  use mod_symmetry
  use mod_dynmat
  use mod_constants, only: zzero
```

LOCAL VARIABLES:

```
    implicit none
    integer(4) :: ikp, iq, fid, ik, iw, ib, ik1
    real(8)    :: t0, t1
integer(4) :: recl
    real(8)    :: ab_plane, ab_norm(3), q0_vol, vtmp(3)
    integer    :: im, iknr, imode
    complex(8) :: vc
    real   (8) :: efold, evtmp, cbmold, evtmp1
    type( k_set) :: int_kset ! interpolation k-set (path)

    real(8), allocatable :: eval1     (:,:) ! original eigenvalues on the KS mesh
    real(8), allocatable :: eval2     (:,:) ! Wannier interpolated eigenvalues
    real(8), allocatable :: evalpath  (:,:) ! Wannier interpolated eigenvalues on a path
    real(8), allocatable :: evalfv    (:,:) ! auxiliary eigenvalue storage
    real(8), allocatable :: phfreq    (:,:) ! original eigenvalues on the KS mesh

    logical    :: l_speceph, l_sigmaeph
    integer    :: nqgrid
    integer    :: ngridkq(3)
     integer    :: ngridkqtot

 ------------------------------------------------!
  Calculate bandstructure by Wannier interpolation !
 ------------------------------------------------!
  if( .not. associated( input%properties%wannier)) then
    write(*,*) " Error (bandstr): Wannier functions have not been calculated."
```

```
    call terminate
  end if
  call readfermi                !saves fermi energy in variable 'efermi'


 -------------------------------------------
  Get the KS eigenvalues on the coarse k grid
 -------------------------------------------
  allocate( eval1( nstfv, nkptnr), evalfv( nstfv, nspinor))
  do iknr = 1, wf_kset%nkpt
    call getevalfv( wf_kset%vkl( :, iknr), evalfv)
    eval1( :, iknr) = evalfv( :, 1)
  enddo

 -----------------------------------------------
  Assign the variables read from the input file
 -----------------------------------------------
  ibeph=input%eph%ibeph
  nbeph=input%eph%nbeph
  ibsumeph=input%eph%ibsumeph
  nbsumeph=input%eph%nbsumeph
  nomegeph=input%eph%freqgrideph%nomegeph
  ngridkq=input%eph%ngridqeph
  ngridkqtot=ngridkq(1)*ngridkq(2)*ngridkq(3)
 print*, "input%eph%ibeph",                input%eph%ibeph
 print*, "input%eph%nbeph",                input%eph%nbeph
 print*, "input%eph%freqgrideqh%nomegeph", input%eph%freqgrideph%nomegeph

 ----------------------------
  Generate the frequency grid
 ----------------------------
  call generate_freqgrid(freq, &
  &                      'eqdist', &
  &                      'refreq', &
  &                      input%eph%freqgrideph%nomegeph, &
  &                      input%eph%freqgrideph%freqmaxeph)
  do iw = 1, nomegeph
    freq%freqs(iw)= freq%freqs(iw) - input%eph%freqgrideph%freqmaxeph / 2
  enddo
  if (.false. .and. rank==0) call print_freqgrid(freq,fgw)

 -----------------------
  Generate the G vectors
 -----------------------
  call generate_G_vectors(gset,  &
  &                       bvec,  &
  &                       intgv, &
  &                       input%groundstate%gmaxvr)
```

```
 if ((.false.).and.(rank==0)) call print_G_vectors(gset,fid)

-----------------------------------------
 Generate the dense q mesh for the phonons
-----------------------------------------
 call generate_k_vectors(qsetd,bvec,ngridkq, &
                         input%eph%vqloffeph,.false.,.false.) ! generate a dense reducible k-p
 if (.true.) then
   call getunit(fid)
   open(fid,file='PH_QPOINTS.OUT',action='Write',status='Unknown')
   call print_k_vectors(qsetd,fid)
 endif

test the transformation from cartesian to lattice coordinates
do iq = 1, ngridkqtot
  call r3mv  (binv, qsetd%vkcnr (:,iq), vtmp )
  write(6,104), iq, qsetd%vkcnr(:,iq), qsetd%vklnr(:,iq), vtmp
enddo
104 format(i4,3x,3f8.4,4x,3f8.4,4x,3f8.4)

-----------------------------------------------
 Generate the k-point path for final quantities
-----------------------------------------------
 input%properties%bandstructure%wannier = .FALSE.
 task=20
 call init1        ! initialize k-points along path

 if (.false.)then
   print*, "nkpt path = " , nkpt
   do ik = 1 , nkpt
     print*, vkl(:, ik)
   enddo
   print*,  eval1
 endif

--------------------------------------
 Import the dynamical matrix from file
--------------------------------------
 call read_dyn()

-------------------------------------------------------
 convert the reducible coarse grid from cartesian
 to lattice coord (read from the dynamical matrix files)
-------------------------------------------------------
 if(allocated(xqlred)) deallocate(xqlred)
 allocate(xqlred(3,nqredtot))
 xqlred = 0.d0
 do iq =1, nqredtot
```

```
    call r3mv (binv, xqcred(:,iq), xqlred(:,iq))
    if(.false.) write(6,103), iq, xqcred (:,iq), xqlred(:,iq)
 enddo
 103 format(i4,3x,3f8.4,4x,3f8.4)
CALL print_ev3 (ev,nqredtot,xqlred,'c')

-----------------------------------------------------------------
 Wannier interpolation of dynamical matrix on the dense BZ grid
-----------------------------------------------------------------
 if (.true.)then
 interpolation on the dense q-mesh on the whole BZ
   if(allocated(dynmatD)) deallocate(dynmatD)
   allocate(dynmatD(ndynmat,ndynmat,ngridkqtot))
   dynmatD=zzero
   if(allocated(wphdense)) deallocate(wphdense)
   allocate(wphdense(ndynmat,ngridkqtot))
   wphdense=0.d0

   CALL wann_ph (nqredtot,xqlred,ngridkqtot,qsetd%vkl,nqred,wphdense)
   CALL print_ev3 (wphdense,ngridkqtot,qsetd%vkl,'c')
 endif

-------------------------------------------------------------------
 Wannier interpolation of dynamical matrix on a path (for plotting)
-------------------------------------------------------------------
 if (.true.)then
 interpolate phonons on a path for visualization
   if(allocated(dynmatD)) deallocate(dynmatD)
   allocate(dynmatD(ndynmat,ndynmat,nkpt))
   dynmatD=zzero
   if(allocated(wphdense)) deallocate(wphdense)
   allocate(wphdense(ndynmat,nkpt))
   wphdense=0.d0

   CALL wann_ph (nqredtot,xqlred,nkpt,vkl,nqred,wphdense)
   CALL print_ev3 (wphdense,nkpt,vkl,'d')
 endif

----------------------------------------
 interpolate KS eigenvalues on the path
----------------------------------------
 This vector will store the Wannier-interpolated eigenvalues
 if( allocated(evalpath)) deallocate( evalpath)
 allocate( evalpath( nstfv, nkpt))
 evalpath = 0.d0
 I have changed this in accordance with the new interface to Wannier interpolation. (Sebastia
 call generate_k_vectors( int_kset, bvec, (/1, 1, nkpt/), (/0.d0, 0.d0, 0.d0/), .false.)
 int_kset%vkl = vkl
```

```
 int_kset%vkc = vkc
 call wfint_init( int_kset, eval1( wf_fst:wf_lst, :))
 evalpath( wf_fst:wf_lst, :) = wfint_eval
call wannier_interpolate_eval ( eval1( wf_fst:wf_lst,:), nkpt, vkl, evalpath( wf_fst:wf_lst,

 Print out the band on the path as a test
 if (.true.) then
   call getunit(fid)
   open(fid,file='band_on_path.OUT',action='Write',status='Unknown')
   do ib = wf_fst, wf_lst
     do ik = 1, nkpt
        write(fid,*) ik, evalpath( ib, ik)
     enddo
   enddo
   close(fid)
 endif

 if(allocated(selfeph0))deallocate(selfeph0)
 allocate(selfeph0(ibeph:nbeph,nkpt))
 selfeph0(:,:)=0.d0
 if(allocated(selfeph))deallocate(selfeph)
 allocate(selfeph(ibeph:nbeph,nomegeph,nkpt))
 selfeph(:,:,:)=0.d0
 if(allocated(speceph))deallocate(speceph)
 allocate(speceph(ibeph:nbeph,nomegeph,nkpt))
 speceph(:,:,:)=0.d0

 if (myrank==0) then
   call boxmsg(fgw,'=','EPH self-energy calculations starts ')
   call flushifc(fgw)
 end if

------------------------------------------------
 generate the unshifted dense q mesh for BZ integral
------------------------------------------------

print*, " Considering following grids for Wannier interpolation:"
print*, " Coarse grid: ", input%groundstate%ngridk
print*, " Dense  grid: ", ngridkq
print*, " Offset     : ", input%eph%vqloffeph
call generate_k_vectors(qsetd,bvec,ngridkqtot, &
                        input%eph%vqloffeph,.false.,.false.) ! generate a dense reducible k-p

---------------------------------------
 LOOP over all the k-points on the path
---------------------------------------
 do ik = 1 , nkpt
```

```
   print*, " ik = ", ik , " / ", nkpt

 ------------------------------------------
  for each k generate the dense k+q mesh for BZ integral
 ------------------------------------------

 print*, " Considering following grids for Wannier interpolation:"
 print*, " Coarse grid: ", input%groundstate%ngridk
 print*, " Dense  grid: ", ngridkqtot
 print*, " Offset     : ", input%eph%vqloffeph

 call generate_k_vectors(kqsetd,bvec,ngridkqtot, &
                         input%eph%vqloffeph,.false.,.false.) ! generate a dense reducible k-po

  call generate_kkqmt_vectors (kqsetd,gset,bvec,ngridkq, &
                           input%eph%vqloffeph, .false., vkl(:,ik),.false.)
 call getunit(fid)
 call print_kkqmt_vectors (kqsetd,gset,fid)

 print to file the k- and q- mesh used in the interpolation
  if (.false.) then
    call getunit(fid)
    open(fid,file='EPH_QPOINTS-ik'//ik//'.OUT',action='Write',status='Unknown')
    call print_k_vectors(kqsetd%kqmtset,fid)
    call getunit(fid)
    open(fid,file='EPH_KPOINTS-ik'//ik//'.OUT',action='Write',status='Unknown')
    call print_k_vectors(wf_kset,fid)
  endif
 print*,  "Fitting from ", wf_kset%nkpt, " points to ", ngridkqtot

 ------------------------------------------------------------------------
  Wannier interpolate the eigenvalues on the new mesh for BZ integration
 ------------------------------------------------------------------------

  This vector will store the Wannier-interpolated eigenvalues
  if( allocated(eval2)) deallocate(eval2)
  allocate( eval2( nstfv, ngridkqtot))
  eval2 = 0.d0
 print*, 'ngridkqtot = ' , ngridkqtot
 print*, 'kqsetd%kqmtset%vkl', kqsetd%kqmtset%vkl

   call wannier_interpolate_eval( eval1( wf_fst:wf_lst,:), wf_kset%nkpt, wf_kset%vkl, &
                                  eval2( wf_fst:wf_lst,:), ngridkqtot, kqsetd%kqmtset%vkl, wf_
 call wannier_interpolate_eval( eval1( wf_fst:wf_lst,:), ngridkqtot, kqsetd%kqmtset%vkl,eval2
 eval2( wf_fst:wf_lst,:), kqsetd%kset%nkpt, kqsetd%kqmtset%vkl, wf_fst, wf_lst)

  I have changed this in accordance with the new interface to Wannier interpolation. (Sebastia
  call wfint_init( kqsetd%kqmtset, eval1( wf_fst:wf_lst, :))
```

```
  eval2( wf_fst:wf_lst, :) = wfint_eval

  find the Fermi energy on the new mesh
  if (ik == 1) then
    efnew = -10000.d0
    cbm   =  10000.d0
    do ib = wf_fst, wf_lst
      do ik1 = 1, ngridkqtot

        efold  = efnew
        cbmold = cbm
        evtmp  = eval2(ib,ik1)
        if (evtmp < efermi .and. evtmp > efold ) efnew = evtmp
        if (evtmp > efermi .and. evtmp < cbmold) cbm   = evtmp
      enddo
    enddo
    print*, "Calculated Fermi energy : ", efnew * 27.21139 , ' eV'
    print*, "Conduction band minimum : ", cbm   * 27.21139 , ' eV'
 efnew = cbm + 0.100 / 27.21139
  endif

  Print interpolated (and non-interpolated) eigenvalues on a aplane in the BZ
  if (.true. .and. ik .eq.1 ) then
    call getunit(fid)
    open(fid,file='band_plane-KS.OUT',action='Write',status='Unknown')
      ib=24
      do ik1 = 1, wf_kset%nkpt
        if ( abs(wf_kset%vkl(3,ik1)) < 1.d-4) then
          write(fid,*) wf_kset%vkl(1,ik1), wf_kset%vkl(2,ik1), eval1 ( ib, ik1)-efnew
        endif
      enddo
    close(fid)

    call getunit(fid)
    open(fid,file='band_plane-W.OUT',action='Write',status='Unknown')
      ib=24
      do ik1 = 1, ngridkqtot
        if ( abs(kqsetd%kset%vkl(3,ik1)) < 1.d-4) then
          write(fid,*) kqsetd%kset%vkl(1,ik1), kqsetd%kset%vkl(2,ik1), eval2 ( ib, ik1)-efnew
        endif
      enddo
    close(fid)
  endif

  if (.true.) then
   if (ik.eq.1)then
    call getunit(fid)
    open(fid,file='EPH_KS-ev.OUT',action='Write',status='Unknown')
```

```
    do ik1 = 1 , wf_kset%nkpt
      write(fid,100) ik1, wf_kset%vkl(1:3,ik1), (eval1(24, ik1) -efnew)* 27.21139

    enddo
    close(fid)

    call getunit(fid)
    open(fid,file='EPH_Wann-ev.OUT',action='Write',status='Unknown')
    do ik1 = 1 , ngridkqtot
      write(fid,100) ik1, kqsetd%kqmtset%vkl(1:3,ik1), (eval2(24, ik1) -efnew)* 27.21139
    100 format(i4,3x,3f8.4,1x,3f10.6)
    enddo
    close(fid)
   endif
  endif

 ================================================================================
  Main loop: BZ integration
 ================================================================================

#ifdef MPI
  call set_mpi_group(kqsetd%kqmtset%nkpt)
  call mpi_set_range(nproc_row, &
  &                  myrank_row, &
  &                  kqsetd%kqmtset%nkpt, 1, &
  &                  iqstart, iqend)
  call mpi_set_range(nproc_col, &
  &                  myrank_col, &
  &                  freq%nomeg, 1, &
  &                  iomstart, iomend, &
  &                  iomcnt, iomdsp)
  write(*,*) "myrank_row, iqstart, iqend =", myrank_row, iqstart, iqend
  write(*,*) "myrank_col, iomstart, iomend =", myrank_col, iomstart, iomend
  write(*,*) 'iomcnt: ', iomcnt(0:nproc_col-1)
  write(*,*) 'iomdsp: ', iomdsp(0:nproc_col-1)
#else
  iqstart = 1
 iqend = kqset%kqmtset%nkpt
  iomstart = 1
 iomend = freq%nomeg
#endif

  l_speceph=.true.
  if (l_speceph)then
    call calcspeceph(eval2, evalpath, ik)
  endif

  l_sigmaeph=.false.
```

```
  if (l_sigmaeph)then
    call calcselfeph(eval2, evalpath, ik)
  endif

  enddo

  if(l_sigmaeph)then
    call getunit(fid)
    open(fid,file='ev-eph.OUT',action='Write',status='Unknown')
    do ib = ibeph, nbeph
      do ik = 1, nkpt
          write(fid,102) ik, ib, (evalpath(ib,ik)-efnew)* 27.21139, &
                    real(selfeph0(ib,ik))* 27.21139, aimag(selfeph0(ib,ik))* 27.21139
 write(fid,*) " "
      enddo
    enddo
    close(fid)
  endif
  102 format(i4,2x,i4,3x,3f15.6)


  if(l_speceph)then
    call getunit(fid)
    open(fid,file='SIGMA.OUT',action='Write',status='Unknown')
    do ib = ibeph, nbeph
      do ik = 1, nkpt
        do iw = 1, nomegeph ! loop over frequency
          write(fid,101) ik, ib, (freq%freqs(iw))* 27.21139, (evalpath(ib,ik)-efnew)* 27.2113
                         real(selfeph(ib,iw,ik))* 27.21139, aimag(selfeph(ib,iw,ik))* 27.2113
                         speceph(ib,iw,ik)
        enddo
        write(fid,*) " "
      enddo
    enddo
    close(fid)
  endif
  101 format(i4,2x,i4,3x,7f15.6)

  compute the spectral function from the self energy
 call spec (selfeph,)


  print*, "Stopping here "
  call terminate

 #ifdef MPI
     if ((nproc_row>1).and.(myrank_col==0)) then
       write(*,*) "sum self-energy from different q-points"
```

```
      call mpi_sum_array(0,selfex,nbandsgw,kset%nkpt,mycomm_row)
      write(*,*) "sum selfex done"
      if (input%gw%taskname.ne.'g0w0_x') then
        ! G0W0 and GW0 approximations
        call mpi_sum_array(0,selfec,nbandsgw,freq%nomeg,kset%nkpt,mycomm_row)
        if (input%gw%selfenergy%secordw) then
          call mpi_sum_array(0,selfecw2,nbandsgw,freq%nomeg,kset%nkpt,mycomm_row)
          call mpi_sum_array(0,selfecSR,nbandsgw,freq%nomeg,kset%nkpt,mycomm_row)
        end if
        if (input%gw%taskname=='cohsex') then
          ! COHSEX
          call mpi_sum_array(0,sigsx,nbandsgw,kset%nkpt,mycomm_row)
          call mpi_sum_array(0,sigch,nbandsgw,kset%nkpt,mycomm_row)
        end if ! cohsex
        write(*,*) "sum selfec done"
      end if ! selfec
    endif
#endif

   if (myrank==0) then

================================
 Write self-energies to files
================================
      call timesec(t0)
#ifdef _HDF5_
      do ik = 1, kset%nkpt
        write(cik,'(I4.4)') ik
        path = "/kpoints/"//trim(adjustl(cik))
        if (.not.hdf5_exist_group(fgwh5,"/kpoints",cik)) &
        &  call hdf5_create_group(fgwh5,"/kpoints",cik)
        ! k-point
        call hdf5_write(fgwh5,path,"vkl",kset%vkl(1,ik),(/3/))
        ! exchange
        call hdf5_write(fgwh5,path,"selfex",selfex(1,ik),(/nbandsgw/))
        ! correlation
        if (input%gw%taskname.ne.'g0w0_x') &
        &  call hdf5_write(fgwh5,path,"selfec",selfec(1,1,ik),(/nbandsgw,freq%nomeg/))
      end do
#else
call write_selfenergy(ibgw,nbgw,kset%nkpt,freq%nomeg)
#endif
      call timesec(t1)
      time_io = time_io+t1-t0

 KS band structure
     evalks(ibgw:nbgw,:) = evalsv(ibgw:nbgw,:)
call bandanalysis('KS',ibgw,nbgw,evalks(ibgw:nbgw,:),efermi)
```

```
      call bandstructure_analysis('KS', &
      &  ibgw,nbgw,kset%nkpt,evalks(ibgw:nbgw,:),efermi)

  ======================================
   Calculate the quasiparticle energies
  ======================================
  call calcevalqp

  ------------------------------------------------------
   Write quasi-particle energies to file
  ------------------------------------------------------
        call timesec(t0)
  call write_qp_energies('EVALQP.DAT')
        call timesec(t1)
        time_io = time_io+t1-t0

   G0W0 QP band structure
       select case (input%gw%taskname)

         case('g0w0_x')
  call bandanalysis('G0W0_X',ibgw,nbgw,evalqp(ibgw:nbgw,:),eferqp)
           call bandstructure_analysis('G0W0_X', &
           &  ibgw,nbgw,kset%nkpt,evalqp(ibgw:nbgw,:),eferqp)

         case('cohsex')
  call bandanalysis('COHSEX',ibgw,nbgw,evalqp(ibgw:nbgw,:),eferqp)
           call bandstructure_analysis('COHSEX', &
           &  ibgw,nbgw,kset%nkpt,evalqp(ibgw:nbgw,:),eferqp)

         case('g0w0','gw0')
  call bandanalysis('G0W0',ibgw,nbgw,evalqp(ibgw:nbgw,:),eferqp)
           call bandstructure_analysis('G0W0', &
           &  ibgw,nbgw,kset%nkpt,evalqp(ibgw:nbgw,:),eferqp)

       end select

        if (input%gw%selfenergy%secordw) then
        !---------------------------------
        ! Second order screened exchange
        !---------------------------------
          call selfcGW_vs_selfcW2
        end if

     end if ! myrank

  ------------------------------------------------------
   Calculate quasiparticle energies in GW0 approximation
  ------------------------------------------------------
```

```
    if (input%gw%taskname=='gw0') then

  self-consistent cycle
      call calcscgw0

  print GW0 QP band structure
      if (myrank==0) then
         call timesec(t0)
 ----------------------------------------
  Write quasi-particle energies to file
 ----------------------------------------
        call write_qp_energies('EVALQP-GW0.DAT')
 call bandanalysis('GW0',ibgw,nbgw,evalqp(ibgw:nbgw,:),eferqp)
        call bandstructure_analysis('GW0', &
        &  ibgw,nbgw,kset%nkpt,evalqp(ibgw:nbgw,:),eferqp)
         call timesec(t1)
         time_io = time_io+t1-t0
      end if
    end if

    if (myrank==0) then
 ----------------------------------------
  Save QP energies into binary file
 ----------------------------------------
       call timesec(t0)
      call putevalqp()
#ifdef _HDF5_
      call hdf5_write(fgwh5,"/","efermi",efermi)
      call hdf5_write(fgwh5,"/","eferqp",eferqp)
      do ik = 1, kset%nkpt
        write(cik,'(I4.4)') ik
        path = "/kpoints/"//trim(adjustl(cik))
  KS energies
        call hdf5_write(fgwh5,path,"evalks",evalks(1,ik),(/nbandsgw/))
  QP energies
        call hdf5_write(fgwh5,path,"evalqp",evalqp(1,ik),(/nbandsgw/))
      end do
#endif
    end if ! myrank

    if (allocated(evalsv)) deallocate(evalsv)
    call delete_selfenergy

    call delete_freqgrid(freq)
    call delete_k_vectors(kset)
    call delete_G_vectors(Gset)
    call delete_Gk_vectors(Gkset)
    call delete_kkqmt_vectors(kqsetd)
```

```
     call delete_Gk_vectors(Gqset)
     call delete_Gk_vectors(Gqbarc)

     return
 end subroutine
  EOC
\markboth{Left}{Source File: gw\_main.f90,  Date: Fri 21 Dec 16:51:20 CET 2018
}



 subroutine gw_main()

     use modinput
     use modmain
     use modgw
     use modmpi
     use mod_mpi_gw
     use m_getunit
     use mod_hdf5

     implicit none
     character(80) :: fname
     real(8) :: tstart, tend

  ------------------------------------------
   Skip initializing and running any GW task
  ------------------------------------------
     if (trim(input%gw%taskname)=='skip') return

   Initialize timing and memory usage variables
     call timesec(tstart)
     call init_timing


  ---------------------
   Main GW output file
  ---------------------
     if (rank==0) then
         call getunit(fgw)
         open(fgw,File='GW_INFO.OUT')
         if (input%gw%debug) then
             call getunit(fdebug)
             open(fdebug,File='debug.info',Action='Write')
         end if
         call boxmsg(fgw,'=','Main GW output file')
         call flushifc(fgw)
     end if
```

```
 ------------------------------------------
  initialize GW MPI environment
 ------------------------------------------
    call init_mpi_gw

 ------------------------------------------------------------
  Parse and check the validity of some GW input parameters
 ------------------------------------------------------------
    call parse_gwinput

 ------------------------------------------
  Store all important results to the hdf5 file
 ------------------------------------------
#ifdef _HDF5_
    call hdf5_initialize()
    fgwh5 = "gw_output.h5"
    if (rank==0) then
      select case (input%gw%taskname)

        case('acon','band', 'sepl')
          continue

        case default
          call hdf5_create_file(fgwh5)
          call hdf5_create_group(fgwh5,"/","parameters")
          call hdf5_create_group(fgwh5,"/","kpoints")
          call write_gw_parameters_hdf5

      end select
    end if
#endif

 ---------------
  Task selector
 ---------------
    select case(input%gw%taskname)

  GW calculations
       case('g0w0','g0w0_x','gw0','cohsex')
           call task_gw

  Calculate the QP band structure
       case('band')
           if (rank==0) call task_band

  Calculate QP DOS
       case('dos')
```

```
          if (rank==0) call task_dos

   Calculate the macroscopic dielectric function
         case('emac')
          call task_emac

   test option: q-dependent \epsilon along a k-path
   case('emac_q')
       call task_emac_q

   Calculate diagonal matrix elements of the exchange-correlation potential
         case('vxc')
          call init_gw
          call calcvxcnn

   Calculate matrix elements of the momentum operator
         case('pmat')
          call init_gw
          call calcpmatgw

   Perform analytic continuation of the correlation self-energy and
   calculate QP energies
         case('acon')
          if (rank==0) call task_analytic_continuation

   Calculate and store the (q,\omega)-dependent dielectric function
         case('epsilon')
          call task_epsilon

   Calculate and store the (q,\omega)-dependent dielectric function
         case('chi0_r')
          call task_chi0_r

   Calculate and store the (q,\omega)-dependent dielectric function
         case('chi0_q')
          call task_chi0_q

   Calculate and store the (q,\omega)-dependent dielectric function
         case('eps_r')
          call task_eps_r

   Calculate the eigenvalues the LDA dielectric function and its inverse
   case('epsev')
      call task_epsev

   Calculate the eigenvalues the GW dielectric function and its inverse
   case('epsgw')
      call task_epsgw
```

```
Calculate the eigenvalues of the screened coulomb potential
case('wev')
   call task_wev

(testing option) Check generation of k-, q-, k+G, q+G, etc. sets
      case('kqgen')
          if (rank==0) call test_kqpts

(testing option) Test generation of k- k/q-dependent BZ integration weights
      case('bzintw')
          if (rank==0) call test_bzintw

(testing option) Calculate LAPW basis functions for plotting
      case('lapw')
          call init_gw
          if (rank==0) call plot_lapw

(testing option) Calculate LAPW eigenvectors for plotting (test option)
      case('evec')
          call init_gw
          if (rank==0) call plot_evec

(testing option) Calculate LAPW eigenvectors products for plotting (test option)
      case('prod')
          call init_gw
          if (rank==0) call test_prodfun

(testing option) Calculate eigenvectors products compared
with mix basis expansion for plotting
      case('mixf')
          call init_gw
          if (rank==0) call test_mixfun

(testing option) Integrate eigenvector products directly and
as a sum of the Minm matrix elements
      case('comp')
          call init_gw
          if (rank==0) call test_mixcomp

(testing option) Test the bare coulomb matrix for various q-points
case('coul')
   if (rank==0) call test_coulpot

(testing option) Plot the self-energy
      case('sepl')
          if (rank==0) call plot_selfenergy
```

```
        case('wannier')
  call init_gw
            if( associated( input%properties%wannier)) then
              input%properties%wannier%input = "gw"
  input%properties%wannier%fst = ibgw
  input%properties%wannier%lst = nbgw
              call wannierlauncher
  call task_eph ()
              call task_band
            end if

   (testing option) Check the rotational matrix for MB functions
   case('rotmat')
   if (rank==0) call test_mbrotmat

     end select

   output timing info
     call timesec(tend)
     time_total = time_total+tend-tstart
     call print_timing

   close GW output file
     if (rank==0) then
       close(fgw)
       if (input%gw%debug) close(fdebug)
     end if

 #ifdef _HDF5_
     call hdf5_finalize()
 #endif

     return
 end subroutine
\markboth{Left}{Source File: setrindex.f90,  Date: Fri 21 Dec 16:51:20 CET 2018
}


  ------------------------------------------------------------
%//////////////////////////////////////////////////////////////

\mbox{}\hrulefill\



  !MODULE: fouri

 module fouri
```

```
   This module declares the R-vectors and stars for the Fourier transform


{\bf PUBLIC TYPES:}
\begin{verbatim}     integer(4) :: nrr                        ! Number of R vectors
     integer(4) :: nst                       ! Number of stars
     integer(4) :: nirk                      ! Number of irreducible k-points
     integer(4) :: irdivk                    ! Division of irred. k-points
     integer(4), allocatable :: irk(:)       ! Integer index of irred. k-points
     integer(4), allocatable :: rindex(:,:)
     integer(4), allocatable :: rst(:,:)
     integer(4), allocatable :: kir(:,:)

     real(8)    :: rmax
     real(8)    :: rbas(3,3)                 ! basis vectors

     complex(8), allocatable :: ek(:,:)
     complex(8), allocatable :: er(:,:)
     complex(8), allocatable :: ekpl(:,:)

     logical :: setrindex_done = .false.

 end module fouri
```

!ROUTINE: setrindex

INTERFACE:

```
 subroutine setrindex
```

DESCRIPTION:

Sets the indexes of the real space lattice vectors used for Fourier interpolation ordered by increasing length

USES:

```
     use modinput
     use modmain
     use fouri
```

LOCAL VARIABLES:

```
     implicit none
     integer(4) :: i         ! (Counter): runs over coordinates
     integer(4) :: ippw      ! (Counter): runs over plane waves
     integer(4) :: ir1       ! (Counter): run over x-coord of G
```

```
integer(4) :: ir2       ! (Counter): run over y-coord of G
integer(4) :: ir3       ! (Counter): run over z-coord of G
integer(4) :: jppw      ! (Counter): runs over plane waves
integer(4) :: nr1       ! Max. ir1
integer(4) :: nr2       ! Max. ir2
integer(4) :: nr3       ! Max. ir3
integer(4) :: nr       ! Maximum numb,er of plane waves in intipw
integer(4) :: nrmin,nrmax,ist,j,isym,lspl
integer(4), dimension(3) :: irvec ! Integer coordinates of the G-vector
integer(4), allocatable  :: invrindex(:,:,:)

real(8) :: rr
real(8), dimension(3) :: rvec     ! Cartesian coordinates of the R-vector
integer(4), allocatable :: rind(:,:) ! Temporary storage for rindex
real(8), allocatable :: rlen(:)   ! Temporary storage for all the  qpg's

integer :: ierr
character(len=10)::sname="setrindex"
```

REVISION HISTORY:

```
Created May 2004 by RGA
Last Modified 9. Nov. 2005 by RGA
Revisited July 2011 by DIN
```

!ROUTINE: calcbarcmb
INTERFACE:

```
subroutine calcbarcmb(iq)
```

DESCRIPTION:

This subroutine calculates the matrix of the bare coulomb potential
USES:

```
use modinput
use modgw
use mod_mpi_gw, only: myrank
```

INPUT PARAMETERS:

```
implicit none
integer, intent(in) :: iq ! index of the q-point
```

LOCAL VARIABLES:

```
    integer :: imix, jmix, igq, jgq
    real(8) :: tstart, tend, t0, t1
  for diagonalization subroutine
    real(8) :: vl, vu, abstol
    integer :: il, iu, neval, lwork, info, lrwork, liwork
    complex(8), allocatable :: work(:)
    real(8),    allocatable :: rwork(:)
    integer,    allocatable :: iwork(:), ifail(:), isuppz(:)

    real(8), external :: dlamch
```

REVISION HISTORY:

```
  Created Jan 2014 by DIN
```

!ROUTINE: setuprod
INTERFACE:

```
subroutine setuprod(ia,is)
```

DESCRIPTION:

This subroutine calculates the radial product functions and their overlap matrix
USES:

```
    use modinput
    use modmain
    use modgw
    use reallocate
    use mod_mpi_gw

 !INPUT VARIABLES:
    implicit none
    integer(4), intent(in) :: ia
    integer(4), intent(in) :: is
```

LOCAL VARIABLES:

```
    integer(4) :: ias
    integer(4) :: io1, io2
    integer(4) :: ilo1, ilo2
    integer(4) :: ipr1, ipr2
    integer(4) :: ir, l1, l2, ist
    integer(4) :: nupcore
    real(8) :: fr(nrmtmax)
    real(8) :: gr(nrmtmax)
    real(8) :: cf(3,nrmtmax)
```

REVISION HISTORY:

```
Created 17. May 2006 by RGA
Revisited 5.05.2011 by DIN
Modified Dec 2013 by DIN
```

2.0.212 ylm (Source File: ylm.f90)

INTERFACE:

```
subroutine ylm(v,lmax,y)
```

DESCRIPTION:

This subroutine calculates spherical harmonics up to l=lmax for a given vector in cartesian coordinates
The spherical harmonics (Condon and Shortley convention) $Y_{0,0}, Y_{1,-1}, Y_{1,0}, Y_{1,1}, Y_{2,-2}...Y_{LMAX,LMAX}$ for vector V (given in Cartesian coordinates) are calculated. In the Condon Shortley convention the spherical harmonics are defined as

$$Y_{l,m} = (-1)^m \sqrt{\frac{1}{2\pi}} P_l^m(cos(\theta)) e^{im\phi} \tag{19}$$

where $P_l^m(cos(\theta))$ is the normalized Associated Legendre function. Thus,

$$Y_{l,-m} = (-1)^m Y_{l,m}^* \tag{20}$$

The output is writen to the vector Y(:) such that Y(k)= $Y_{l,m}$ with $k = l^2 + l + m + 1$, thus,
Y(1)= $Y_{0,0}$, Y(2)= $Y_{1,-1}$, Y(3)= $Y_{1,0}$, Y(4)= $Y_{1,1}$... Y(lmax*lmax+1)= $Y_{lmax,-lmax}$...Y((lmax+1)*(lmax+1))= $Y_{lmax,lmax}$
The basic algorithm used to calculate the spherical harmonics for vector V is as follows:

$$Y_{0,0} = \sqrt{\frac{1}{4\pi}} \tag{21a}$$

$$Y_{1,0} = \sqrt{\frac{3}{4\pi}} cos(\theta) \tag{21b}$$

$$Y_{1,1} = -\sqrt{\frac{3}{8\pi}} sin(\theta) e^{i\phi} \tag{21c}$$

$$Y_{1,-1} = -Y_{1,1} \tag{21d}$$

$$Y_{l,l} = -\sqrt{\frac{2l+1}{2l}} sin(\theta) e^{i\phi} Y_{l-1,l-1} \tag{21e}$$

$$Y_{l,m} = \sqrt{\frac{(2l-1)(2l+1)}{(l-m)(l+m)}} cos(\theta) Y_{l-1,m} - \sqrt{\frac{(l-1+m)(l-1-m)(2l+1)}{(2l-3)(l-m)(l+m)}} Y_{l-2,m} \tag{21f}$$

INSTITUT FUER THEORETISCHE CHEMIE – TU VIENNA
INPUT PARAMETERS:

```
implicit none

integer(4) :: lmax ! maximum l for which the spherical
                   harmonics are calculated

real(8) :: v(3)    ! vector (cartesian coordinates) for which
                   the spherical harmonics are calculated
```

OUTPUT PARAMETERS:

```
complex(8) :: y(*) ! the values of the spherical harmonics
                   dimension (lmax+1)^2
```

!ROUTINE: calcmicm
INTERFACE:

```
subroutine calcmicm(ik,iq,cstart,cend,mstart,mend,micm)
```

DESCRIPTION:

This subroutine calculates the matrix elements $M^i_{cm}(\vec{k},\vec{q})$ (c stands for core-state)
USES:

```
use modinput
use modmain,           only : pi, idxas, idxlm, idxlo, nlorb, apword, &
&                             lorbl, zzero
use modgw,             only : kqset, Gkset, fdebug, time_micm
use mod_bands,         only : eveckp, eveckpalm
use mod_product_basis, only : locmatsiz, nmix, bigl, locmixind, bradketc
use mod_core_states,   only : corind, ncg
use mod_misc_gw,       only : atposl
use mod_gaunt_coefficients
```

INPUT PARAMETERS:

```
implicit none
integer(4), intent(in)  :: ik   ! k-point
integer(4), intent(in)  :: iq   ! q-point
integer(4), intent(in)  :: cstart, cend  ! range of core states
integer(4), intent(in)  :: mstart, mend  ! range of m states
complex(8), intent(out) :: micm(locmatsiz,ncg,mstart:mend)
```

LOCAL VARIABLES:

```
    integer(4) :: jk
    integer(4) :: bl, bm
    integer(4) :: ia, is, ias
    integer(4) :: igk2
    integer(4) :: io2, ilo2
    integer(4) :: icg, ic, ie2
    integer(4) :: imix, irm, im
    integer(4) :: l1, m1, l2, m2, l2m2
    integer(4) :: l2min, l2max

    real(8) :: arg, kvec(3)
    real(8) :: tstart,tend

    complex(8) :: phs, angint, sum

  !EXTERNAL ROUTINES:
    external :: zgemm
    real(8), external :: gaunt
```

REVISION HISTORY:

```
Created  23th. Feb. 2004 by RGA
Last modified 20th. July 2004 by RGA
Revisited: May 2011 by DIN
```

2.0.213 combin (Source File: combin.f90)

INTERFACE:

```
real(8) function combin(num,denom1,denom2)
```

DESCRIPTION:

Calculates the factorial quotients $\frac{n_1!}{n_2!n_3!}$

INPUT PARAMETERS:

```
implicit none

integer(4), intent(in) :: num,denom1,denom2
```

LOCAL VARIABLES:

```
integer(4) :: i
integer(4) :: dmin,dmax
real(8)    :: cm
```

```
!EXTERNAL ROUTINES:

    real(8), external :: factr

!INTRINSIC ROUTINES:

    intrinsic min
    intrinsic max
```

REVISION HISTORY:

```
Created March  2004 by RGA
Last modified July 20th 2004 by RGA
```

!ROUTINE: barcq0
INTERFACE:

```
subroutine barcq0()
```

DESCRIPTION:

This subroutine calculates the matrix of the bare coulomb potential for q=0 and atomic functions with L=0
USES:

```
use modinput
use modmain,               only : nspecies, natoms, idxas, zzero, pi, &
&                                 gkmax, natmtot, nrmt, spr, bvec
use mod_product_basis,     only : nmix, bigl, maxnmix, umix, mbindex, &
                                  locmatsiz
use mod_coulomb_potential, only : barc
use mod_misc_gw,           only : vi, atposl, pia
use mod_kpointset
```

LOCAL VARIABLES:

```
implicit none

type(G_set) :: Gset

integer :: ngrid(3), grid(3,2)
integer :: ng, ngl, igl, ig
integer :: ia, is, ias, ja, js, jas
```

```
integer :: nr, ir, irm, jrm
integer :: l1, m1, l2, m2
integer :: imix, jmix

integer, allocatable :: igl2ig(:), ig2igl(:)


real(8) :: gmax, gl0
real(8) :: dpos(3), gvec(3)
real(8) :: x, gpr

real(8), allocatable :: sinf(:), sing(:,:,:)
real(8), allocatable :: fr(:), gr(:), cf(:,:)

complex(8) ::  expg, sum
complex(8), allocatable :: phase(:,:,:)
```

REVISION HISTORY:

```
Created 16th. March 2004 by RGA
Last modified 31. March 2005 by RGA
Revisited: June 2011 by DIN
```

!ROUTINE: calcmicm
INTERFACE:

```
subroutine calcminc(ik,iq,nstart,nend,cstart,cend,minc)
```

DESCRIPTION:

This subroutine calculates the matrix elements $M^i_{nc}(\vec{k},\vec{q})$ (c stands for core-state)
USES:

```
use modinput
use modmain,           only : pi, idxas, idxlm, idxlo, nlorb, apword, &
&                             lorbl, zzero
use modgw,             only : kqset, Gkset, fdebug, time_minc
use mod_bands,         only : eveck, eveckp, eveckalm, eveckpalm
use mod_product_basis, only : locmatsiz, nmix, bigl, locmixind, bradketc
use mod_core_states,   only : corind, ncg
use mod_misc_gw,       only : vi, atposl
use mod_gaunt_coefficients
```

INPUT PARAMETERS:

```
    implicit none
    integer(4), intent(in) :: ik   ! k-point
    integer(4), intent(in) :: iq   ! q-point
    integer(4), intent(in) :: nstart, nend  ! range of n states
    integer(4), intent(in) :: cstart, cend  ! range of c states
    complex(8), intent(out) :: minc(locmatsiz,nstart:nend,cstart:cend)
```

LOCAL VARIABLES:

```
    integer(4) :: bl, bm
    integer(4) :: ia, is, ias
    integer(4) :: igk2
    integer(4) :: io2, ilo2
    integer(4) :: icg, ic, ie2
    integer(4) :: imix, irm, im
    integer(4) :: l1, m1, l2, m2, l2m2
    integer(4) :: l2min, l2max

    real(8) :: arg, kvec(3)
    real(8) :: tstart, tend

    complex(8) :: phs, angint, sum

  !EXTERNAL ROUTINES:
    external :: zgemm
    real(8), external :: gaunt
```

REVISION HISTORY:

```
  Created  23th. Feb. 2004 by RGA
  Last modified 20th. July 2004 by RGA
  Revisited: May 2011 by DIN
```

!ROUTINE: calcminm
INTERFACE:

```
subroutine calcminm(ik,iq,nstart,nend,mstart,mend,minm)
```

DESCRIPTION:

This subroutine calculates the matrix elements $M^i_{nm}(\vec{k},\vec{q})$
USES:

```
    use modinput
    use modmain,              only : nspecies, natoms, idxas, idxlm, idxlo, &
    &                                zzero, zone, intgv, apword, nlorb, lorbl, &
```

```
&                                   nlomax, pi, apwordmax, nmatmax
use modgw,                  only : kqset, Gkset, Gqbarc, Gqset, Gset, fdebug, time_minm
use mod_bands,              only : eveck, eveckp, eveckalm, eveckpalm
use mod_product_basis,      only : nmix, bigl, bradketa, bradketlo, mpwipw, &
&                                  matsiz, locmatsiz, mbindex
use mod_misc_gw,            only : vi, atposl
use mod_gaunt_coefficients
```

INPUT PARAMETERS:

```
implicit none
integer(4), intent(in) :: ik   ! the index of the first k-point
integer(4), intent(in) :: iq   ! the index of the q-point
integer(4), intent(in) :: nstart, nend  ! range of n states
integer(4), intent(in) :: mstart, mend  ! range of m states
complex(8), intent(out):: minm(matsiz,nstart:nend,mstart:mend)
```

LOCAL VARIABLES:

```
integer(4) :: jk
integer(4) :: bl, bm
integer(4) :: i, ia, is, ias
integer(4) :: igk1, igk2
integer(4) :: io1, io2, ilo1, ilo2
integer(4) :: imix, irm
integer(4) :: ie1, ie2
integer(4) :: l1, m1, l2, m2, l1m1, l2m2
integer(4) :: l2min, l2max
integer(4) :: ig, igq
integer(4), allocatable :: igqk12(:,:)
integer(4), dimension(3):: ikv, ig0 ! Indexes of G_1+G'-G
integer(4) :: ngk1, ngk2
integer(4) :: ndim, mdim, nmdim

real(8) :: sqvi, x, arg, angint
real(8) :: qvec(3)
real(8) :: tstart, tend, tmt, tir

real(8) :: t1, t2

complex(8) :: phs
complex(8) :: sumterms
complex(8) :: apwterm(apwordmax)
complex(8) :: loterm(nlomax)

complex(8), allocatable :: tmat(:,:), tmat2(:,:), mnn(:,:)
```

```
!EXTERNAL ROUTINES:
   external :: zgemm
   real(8), external :: gaunt
```

REVISION HISTORY:

```
Created  23th. Feb. 2004 by RGA
Last modified 20th. Jul. 2004 by RGA
Revisited 29.04.2011 by DIN
```

!ROUTINE: kintw
INTERFACE:

```
subroutine kintw()
```

DESCRIPTION:

This subroutine calculates the occupancy dependent k-point weights for BZ integrations.
USES:

```
use modinput
use modmain, only: evalsv, evalcr, nstsv, efermi, &
&                  natmtot, nspecies, natoms, idxas
use modgw,   only: kset, kqset, kiw, ciw, kwfer, metallic, fdebug
use mod_core_states, only: ncmax, ncore
```

LOCAL VARIABLES:

```
implicit none
integer(4) :: ia
integer(4) :: ias
integer(4) :: is
integer(4) :: ist
integer(4) :: ik, ikp
real(8), allocatable :: bandpar(:,:)
real(8), allocatable :: cwpar(:,:)
```

!ROUTINE: fourintp
INTERFACE:

```
subroutine fourintp(f1,nk1,kvecs1,f2,nk2,kvecs2,nb)
```

DESCRIPTION:

This subroutine interpolate function $f1_n(k)$ defined on the kmesh 1, to kmesh 2 using 3D Smooth Fourier transform according to PRB 38, 2721 (1988).
USES:

```
use modinput
use modmain, only: nsymcrys, pi, zzero, zone
use fouri
```

LOCAL VARIABLES:

```
implicit none
integer(4), intent(in) :: nk1,nk2,nb
real(8),    intent(in) :: kvecs1(3,nk1),kvecs2(3,nk2)
complex(8), intent(in) :: f1(nk1,1:nb)
complex(8), intent(out):: f2(nk2,1:nb)

integer(4) :: i, ist, ib, ik, jk, ir
integer(4) :: info
integer(4), allocatable  :: ipiv(:)

real(8) :: den, pref, kdotr
real(8) :: rmin,rlen,x2,x6,c1,c2
real(8), dimension(3) :: r,rvec,kvec
real(8), allocatable  :: rho(:)

complex(8) :: expkr
complex(8), allocatable :: dele(:,:)
complex(8), allocatable :: h(:,:)
complex(8), allocatable :: coef(:,:)
complex(8), allocatable :: smat1(:,:),smat2(:,:)
complex(8), allocatable :: sm2(:,:)
```

REVISION HISTORY:

```
Created 02.03.06 by RGA
Revisited July 2011 by DIN
```

2.0.214 derfc (Source File: derfc.f90)

INTERFACE:

```
real(8) function derfc(x)
```

INPUT PARAMETERS:

```
real(8) :: x
```

DESCRIPTION:

Error function in double precision. Some compilers include it as an internal procedure, but since some of them don't, it is better to have the external one

LOCAL VARIABLES:

```
implicit none

real(8) :: t
real(8) :: u
real(8) :: x
real(8) :: y
```

DEFINED PARAMETERS:

```
 real(8), parameter :: &
&    pa = 3.97886080735226000d+00, &
&    p0 = 2.75374741597376782d-01, &
&    p1 = 4.90165080585318424d-01, &
&    p2 = 7.74368199119538609d-01, &
&    p3 = 1.07925515155856677d+00, &
&    p4 = 1.31314653831023098d+00, &
&    p5 = 1.37040217682338167d+00, &
&    p6 = 1.18902982909273333d+00, &
&    p7 = 8.05276408752910567d-01, &
&    p8 = 3.57524274449531043d-01, &
&    p9 = 1.66207924969367356d-02, &
&    p10 = -1.19463959964325415d-01, &
&    p11 = -8.38864557023001992d-02
 real(8), parameter :: &
&    p12 = 2.49367200053503304d-03, &
&    p13 = 3.90976845588484035d-02, &
&    p14 = 1.61315329733252248d-02, &
&    p15 = -1.33823644533460069d-02, &
&    p16 = -1.27223813782122755d-02, &
&    p17 = 3.83335126264887303d-03, &
&    p18 = 7.73672528313526668d-03, &
&    p19 = -8.70779635317295828d-04, &
&    p20 = -3.96385097360513500d-03, &
&    p21 = 1.19314022838340944d-04, &
&    p22 = 1.27109764952614092d-03
```

2.0.215 higam (Source File: higam.f90)

INTERFACE:

```
recursive function higam(n) result(tgam)
```

DESCRIPTION:

This function calculates the gamma function for half integer values $\Gamma(n+\frac{1}{2})$ recursively, using the relation: $\Gamma(n+\frac{1}{2}) = \frac{2n-1}{2}\Gamma(n-\frac{1}{2})$, and $\Gamma(\frac{1}{2}) = \sqrt{\pi}$.
INPUT PARAMETERS:

```
implicit none
integer(4) :: n
```

OUTPUT PARAMETERS:

```
real(8) :: tgam
```

DEFINED PARAMETERS:

```
real(8) :: pi
pi = 4.0d+0*datan(1.0d+0)
```

2.0.216 fint (Source File: fint.f90)

INTERFACE:

```
subroutine fint(np,n,x,f,g)
```

INPUT/OUTPUT PARAMETERS:

```
np : order of fitting polynomial (in,integer)
n  : number of points (in,integer)
x  : abscissa array (in,real(n))
f  : function array (in,real(n))
g  : integrated function (out,real(n))
```

DESCRIPTION:

Calculates the integrals $g(x_i)$ of a function f defined on a set of points x_i as follows

$$g(x_i) = \int_0^{x_i} f(x)\,dx.$$

This is performed by piecewise fitting the function to polynomials of order $n_p - 1$ and performing the integrations analytically.
REVISION HISTORY:

```
Created May 2002 (JKD)
```

!ROUTINE: calcmpwipw
INTERFACE:

```
subroutine calcmpwipw(iq)
```

DESCRIPTION:

This subroutine calculates the matrix elements between PW's and orthogonalized IPW's.
USES:

```
use modinput
use modmain, only : cfunig, zzero, zone, ngrtot, ngrid, igfft
use modgw,   only : Gset, Gqset, Gqbarc, sgi, sgi_fft, mpwipw, &
&                   fdebug, time_mpwipw
use modmpi,  only : rank
```

INPUT PARAMETERS:

```
implicit none
integer(4), intent(in) :: iq
```

LOCAL VARIABLES:

```
integer(4) :: ig, npw, ngq, ipw, igq
integer(4), dimension(3) :: igv ! integer coodintates of G-G'
complex(8), allocatable  :: tmat(:,:)
real(8) :: tstart, tend

!EXTERNAL ROUTINES:
  external zgemm
```

REVISION HISTORY:

```
Created: May 2006 by RGA
Revisited 10.05.2011 by DIN
```

2.0.217 besk0 (Source File: gencoulcut.f90)

INTERFACE:

```
Real (8) Function besk0 (x)
```

INPUT/OUTPUT PARAMETERS:

```
x : real argument (in,real)
```

DESCRIPTION:

besk0 evaluates the Bessel K0(X) function. Returns the error function erf(x) using a rational function approximation. This procedure is numerically stable and accurate to near machine precision. This routine computes approximate values for the modified Bessel function of the second kind of order zero for arguments 0.0 ¡ ARG ¡= XMAX. See comments heading CALCK0.
REVISION HISTORY:

2.0.218 besk1 (Source File: gencoulcut.f90)

INTERFACE:

```
Real (8) Function besk1 ( x )
```

INPUT/OUTPUT PARAMETERS:

```
x : real argument (in,real)
```

DESCRIPTION:

besk1 evaluates the Bessel K1(X) function. This routine computes approximate values for the modified Bessel function of the second kind of order one for arguments XLEAST ¡= ARG ¡= XMAX.

REVISION HISTORY:

2.0.219 calck0 (Source File: gencoulcut.f90)

INTERFACE:

```
subroutine calck0 ( arg, result, jint )
```

INPUT/OUTPUT PARAMETERS:

```
arg : real argument (in,real).
 0 < ARG is
 always required.  If JINT = 1, then the argument must also be
 less than XMAX.

result : real argument (in,real)
 Input, real ( kind = 8 ) ARG, the argument.  0 < ARG is
 always required.  If JINT = 1, then the argument must also be
 less than XMAX.

 Output, real ( kind = 8 ) RESULT, the value of the function,
 which depends on the input value of JINT:
 1, RESULT = K0(x);
 2, RESULT = exp(x) * K0(x);

jint : real argument (in,real)
 Input, integer ( kind = 4 ) JINT, chooses the function to be computed.
 1, K0(x);
 2, exp(x) * K0(x);
```

DESCRIPTION:

CALCK0 computes various K0 Bessel functions. This routine computes modified Bessel functions of the second kind and order zero, K0(X) and EXP(X)*K0(X), for real arguments X.
The main computation evaluates slightly modified forms of near minimax rational approximations generated by Russon and Blair, Chalk River (Atomic Energy of Canada Limited) Report AECL-3461, 1969. **REVISION HISTORY:**

2.0.220 calck0 (Source File: gencoulcut.f90)

INTERFACE:

```
subroutine calck1 ( arg, result, jint )
```

INPUT/OUTPUT PARAMETERS:

```
arg : real argument (in,real).
 0 < ARG is
 always required.  If JINT = 1, then the argument must also be
 less than XMAX.

result : real argument (in,real)
 Input, real ( kind = 8 ) ARG, the argument.  XLEAST < ARG is
 always required.  If JINT = 1, then the argument must also be
 less than XMAX.

 Output, real ( kind = 8 ) RESULT, the value of the function,
 which depends on the input value of JINT:
 1, RESULT = K1(x);
 2, RESULT = exp(x) * K1(x);

jint : real argument (in,real)
 Input, integer ( kind = 4 ) JINT, chooses the function to be computed.
 1, K1(x);
 2, exp(x) * K1(x);
```

DESCRIPTION:

CALCK1 computes various K1 Bessel functions. This routine computes modified Bessel functions of the second kind and order one, K1(X) and EXP(X)*K1(X), for real arguments X.
The main computation evaluates slightly modified forms of near minimax rational approximations generated by Russon and Blair, Chalk River (Atomic Energy of Canada Limited) Report AECL-3461, 1969. **REVISION HISTORY:**

2.0.221 gaulag (Source File: gaulag.f90)

INTERFACE:

```
subroutine gaulag(x,w,n,alfa)
```

DESCRIPTION:

Given `alfa`, the parameter α of the `Laguerre` polynomials, this routine returns arrays `x(1:n)` and `w(1:n)` containing the abscissas and weights of the `n`-point Gauss-Laguerre quadrature formula. The smallest abscissa is returned in `x(1)`, the largest in `x(n)`.

INPUT PARAMETERS:

```
implicit none

integer(4), intent(in) :: n    ! Order of the quadrature

real(8),    intent(in) :: alfa ! alpha parameter of the Laguerre
                                 polynomials
```

OUTPUT PARAMETERS:

```
real(8),    intent(out) :: x(n) ! abscissas
real(8),    intent(out) :: w(n) ! weights
```

LOCAL VARIABLES:

```
integer(4) :: i
integer(4) :: its
integer(4) :: j

real(8) :: ai
real(8) :: dj
real(8) :: p1
real(8) :: p2
real(8) :: p3
real(8) :: pp
real(8) :: z
real(8) :: z1

logical(1) :: conv
```

DEFINED PARAMETERS:

```
integer(4), parameter :: maxit = 10

real(8),    parameter :: eps = 3.0d-14
```

```
!EXTERNAL ROUTINES:


    real(8), external :: gammln


!INTRINSIC ROUTINES:


    intrinsic dble
    intrinsic exp
```

REVISION HISTORY:

```
Original subroutine: gauleg.for (C) copr. 1986-92 copr. 1986-92
numerical recipes software &146i..
Last modified: 20.06.05 by RGA.
```

!ROUTINE: calcwmix0
INTERFACE:

```
subroutine calcwmix0
```

DESCRIPTION:

This subroutine calculate the matrix elements $\mathcal{W}_0^i$
USES:

```
use modinput
use modmain,               only : pi, zzero, idxas
use modgw,                 only : Gqset, fdebug
use mod_product_basis,     only : locmatsiz, matsiz, mpwipw, mbindex, rtl
use mod_coulomb_potential, only : wi0
use mod_misc_gw,           only : vi
```

LOCAL VARIABLES:

```
implicit none
integer(4) :: imix, l1, irm
integer(4) :: ia, is, ias
integer(4) :: igq
real(8)    :: fact
```

REVISION HISTORY:

```
Created 11.02.05 by RGA
Revisited June 2011 by DIN
```

2.0.222 gauleg (Source File: gauleg.f90)

INTERFACE:

```
subroutine gauleg(x1,x2,x,w,n)
```

DESCRIPTION:

Given the lower and upper limits of integration `x1` and `x2`, and given `n`, this routine returns arrays `x(1:n)` and `w(1:n)` of length `n`, containing the abscissas and weights of the Gauss-Legendre `n`-point quadrature formula.

INPUT PARAMETERS:

```
implicit none


integer(4), intent(in) :: n  ! Order of the quadrature

real(8),    intent(in) :: x1 ! Lower integration limit
real(8),    intent(in) :: x2 ! Upper integration limit
```

OUTPUT PARAMETERS:

```
real(8),    intent(out) :: x(n) ! abscissas
real(8),    intent(out) :: w(n) ! weights
```

LOCAL VARIABLES:

```
integer(4) :: i
integer(4) :: j
integer(4) :: m

real(8) :: dj
real(8) :: p1
real(8) :: p2
real(8) :: p3
real(8) :: pp
real(8) :: xl
real(8) :: xm
real(8) :: z
real(8) :: z1
```

```
      logical(1) :: conv
```

DEFINED PARAMETERS:

```
      real(8), parameter :: eps = 3.0d-14

      real(8), parameter :: pi  = 3.141592653589793d+0


  !INTRINSIC ROUTINES:


      intrinsic abs
      intrinsic cos
      intrinsic dble
```

REVISION HISTORY:

```
  Original subroutine: gauleg.for (C) copr. 1986-92 copr. 1986-92
  numerical recipes software &145i..
  Last modified: 20.06.05 by RGA.
```

2.0.223 readingw (Source File: parse_gwinput.f90)

INTERFACE:

```
subroutine parse_gwinput
```

DESCRIPTION:

This subroutine check important GW input parameters
USES:

```
    use modinput
    use modmain
    use modgw
    use modmpi
    implicit none
```

LOCAL VARIABLES:

```
      integer :: idum
      real(8) :: rdum
```

!ROUTINE: diagsgi **INTERFACE:**

```
 subroutine diagsgi(iq)
```

DESCRIPTION:

This subroutine generates the overlap matrix of the product functions in the interstitial region and diagonalizes it. The output is the matrix $S_{\vec{G}i}$.

USES:

```
      use modinput
      use modmain, only : cfunig
      use modgw,   only : Gset, Gqset, sgi, &
      &                   fdebug, time_diagsgi
```

INPUT PARAMETERS:

```
      implicit none
      integer(4), intent(in) :: iq
```

LOCAL VARIABLES:

```
      integer(4) :: ig, ngq
      integer(4) :: igq   ! Counter: Runs over igq's
      integer(4) :: jgq   ! Counter: Runs over igq's
      integer(4), dimension(3) :: iig    ! integer coordinates of G-G'
      real(8)   :: tstart, tend

      complex(8) :: cfact
      real(8), allocatable :: epsipw(:)
      complex(8), allocatable :: zmat(:,:)

    for diagonalization subroutine
      real(8) :: vl, vu, abstol
      integer :: il, iu, neval, lwork, info, lrwork, liwork
      complex(8), allocatable :: work(:)
      real(8),    allocatable :: rwork(:)
      integer,    allocatable :: iwork(:), ifail(:), isuppz(:)
      real(8), external :: dlamch

   !EXTERNAL ROUTINES:
      external zheev      ! Lapack diagonalization subroutine
```

REVISION HISTORY:

```
Created Dec. 2003. by RGA
Last Modification May. 2006 by RGA
Revisited 10.05.2011 by DIN
```

!ROUTINE: calchead **INTERFACE:**

```
subroutine calchead(ik,iomstart,iomend,ndim)
```

DESCRIPTION:

This subroutine calculate the head of the dielectric matrix at the Γ point.
USES:

```
    use modinput
    use modmain, only : zzero, zone, pi, evalsv, evalcr, idxas
    use modgw

!INPUT VARIABLES:
    implicit none
    integer(4), intent(in) :: ik
    integer(4), intent(in) :: iomstart, iomend
    integer(4), intent(in) :: ndim
```

LOCAL VARIABLES:

```
    integer(4) :: ic, icg
    integer(4) :: ia, is, ias
    integer(4) :: ie1, ie2, dimtk
    integer(4) :: ikp, iop, jop
    integer(4) :: iom, fid
    real(8) :: edif    ! energy difference
    real(8) :: edsq    ! edif^2
    real(8) :: tstart, tend
    complex(8) :: coefh
    complex(8) :: pnm, zsum
```

REVISION HISTORY:

```
Created 11.02.05 by RGA
Revisited July 2011 by DIN
```

2.0.224 calctildeg (Source File: calctildeg.f90)

INTERFACE:

```
subroutine calctildeg(lmax)
```

DESCRIPTION:

Calculates $\tilde{g}_{lm,l'm'}$ according to equation **??**, that is:

$$\tilde{g}_{lm,l'm'} = \sqrt{4\pi}(-1)^l \sqrt{\frac{(l+l'+m+m')!(l+l'-m-m')!}{(2l+1)(2l'+1)[2(l+l')+1]!(l+m)!(l-m)!(l'+m')!(l'-m')!}} \tag{22}$$

needed for the calculation of the structure constants, for l and l' = 0 ... lmax and stores them in memory.

USES:

```
use gtil
```

INPUT PARAMETERS:

```
implicit none

integer(4), intent(in) :: lmax
```

LOCAL VARIABLES:

```
integer(4) :: l1,l2,m1,m2,tsize,i


!EXTERNAL ROUTINES:

real(8), external :: tildeg
```

REVISION HISTORY:

```
Created May 2004 by RGA
Last modified 30. June 2004 by RGA
```

2.0.225 gettildeg (Source File: calctildeg.f90)

INTERFACE:

```
real(8) function gettildeg(l1,l2,m1,m2)
```

DESCRIPTION:

Gets the value of $\tilde{g}_{l1,m1,l2,m2}$ from the vector `tilg(i)`, where:

$$\tilde{g}_{l_1 m_1, l_2 m_2} = (-1)^{l_1 - l_2} \texttt{tilg(i)} \tag{23}$$

being

$$\texttt{i} = \frac{1}{12} j_1(j_1+1)(j_1+2)(3j_1-1) + j_2(j_2+1)j_1 + \frac{1}{2} j_2(j_2-1) + (j_2+1)(m_1'+j_1) + m_2' + j_2 + 1 \tag{24}$$

where we defined:

$$\begin{aligned} j_1 &= \max(l_1, l_2) \\ j_2 &= \min(l_1, l_2) \\ m_1' &= \mathrm{sgn}(m_2) m_1 \\ m_2' &= |m_2| \end{aligned} \tag{25}$$

USES:

```
use gtil
```

INPUT PARAMETERS:

```
implicit none

integer(4), intent(in) :: l1
integer(4), intent(in) :: l2
integer(4), intent(in) :: m1
integer(4), intent(in) :: m2
```

LOCAL VARIABLES:

```
integer(4) :: j1,j2,mj1,mj2
integer(4) :: index1, index2, index3, index4
integer(4) :: par,tind
real(8) :: fact


!INTRINSIC ROUTINES:


intrinsic mod
```

REVISION HISTORY:

```
Creeated May 2004 by RGA
Last modified: 30. June 2004 by RGA
```

2.0.226 doreallocate_r8_d1

INTERFACE:

```
subroutine doreallocate_r8_d1(tf, newdimension)
```

INPUT PARAMETERS:

```
implicit none

real(8), pointer :: tf(:)  ! vector to reallocate

integer(4) :: newdimension ! new dimension of the vector
```

DESCRIPTION:

Reallocates a real(8) pointer vector of range 1

LOCAL VARIABLES:

```
real(8), pointer :: hilfsfeld(:)

integer(4) :: min1
```

2.0.227 doreallocate_r8_d2

INTERFACE:

```
subroutine doreallocate_r8_d2(tf, newdimension1, newdimension2)
```

INPUT PARAMETERS:

```
implicit none

real(8), pointer :: tf(:,:) ! the matrix to reallocate

integer(4) :: newdimension1, newdimension2 ! new dimensions
```

DESCRIPTION:

Reallocates a real(8) pointer matrix of range 2

LOCAL VARIABLES:

```
real(8), pointer :: hilfsfeld(:,:)

integer(4) :: min1, min2
```

2.0.228 doreallocate_r8_d3

INTERFACE:

```
      subroutine doreallocate_r8_d3(tf, newdim1, newdim2,          &
     &                                newdim3)
```

INPUT PARAMETERS:

```
        implicit none

        real(8), pointer :: tf(:,:,:)

        integer(4) :: newdim1, newdim2, newdim3
```

LOCAL VARIABLES:

```
        real(8), pointer :: hilfsfeld(:,:,:)

        integer(4) :: min1, min2, min3
```

2.0.229 doreallocate_r8_d4

INTERFACE:

```
      subroutine doreallocate_r8_d4(tf, newdim1, newdim2, newdim3,&
     &    newdim4)
```

INPUT PARAMETERS:

```
        implicit none

        integer(4) :: newdim1,newdim2,newdim3,newdim4

        real(8), pointer :: tf(:,:,:,:)
```

LOCAL VARIABLES:

```
        real(8), pointer :: hilfsfeld(:,:,:,:)

        integer(4) :: min1, min2, min3, min4
```

2.0.230 doreallocate_r8_d5

INTERFACE:

```
      subroutine doreallocate_r8_d5(tf, newdim1, newdim2, newdim3,&
&     newdim4,newdim5)
```

INPUT PARAMETERS:

```
      implicit none

      real(8), pointer :: tf(:,:,:,:,:)

      integer(4) :: newdim1,newdim2,newdim3,newdim4, newdim5
```

LOCAL VARIABLES:

```
      integer(4) :: min1, min2, min3, min4, min5

      real(8), pointer :: hilfsfeld(:,:,:,:,:)
```

2.0.231 doreallocate_r8_d6

INTERFACE:

```
      subroutine doreallocate_r8_d6(tf, newdim1, newdim2, newdim3,&
&     newdim4,newdim5,newdim6)
```

INPUT PARAMETERS:

```
      implicit none

      real(8), pointer :: tf(:,:,:,:,:,:)

      integer(4) :: newdim1, newdim2, newdim3
      integer(4) :: newdim4, newdim5, newdim6
```

LOCAL VARIABLES:

```
      real(8), pointer :: hilfsfeld(:,:,:,:,:,:)

      integer(4) :: min1, min2, min3, min4, min5, min6
```

2.0.232 doreallocate_i4_d1

INTERFACE:

```
subroutine doreallocate_i4_d1(tf, newdimension)
```

INPUT PARAMETERS:

```
implicit none

integer(4), pointer :: tf(:)

integer(4) :: newdimension
```

LOCAL VARIABLES:

```
integer(4), pointer :: hilfsfeld(:)

integer(4) :: min1
```

2.0.233 doreallocate_I4_d2

INTERFACE:

```
subroutine doreallocate_i4_d2(tf, newdimension1, newdimension2)
```

INPUT PARAMETERS:

```
implicit none

integer(4), pointer :: tf(:,:)

integer(4) :: newdimension1, newdimension2
```

LOCAL VARIABLES:

```
integer(4), pointer :: hilfsfeld(:,:)

integer(4) :: min1, min2
```

2.0.234 doreallocate_i4_d3

INTERFACE:

```
subroutine doreallocate_i4_d3(tf, newdim1, newdim2, newdim3)
```

INPUT PARAMETERS:

```
implicit none

integer(4), pointer :: tf(:,:,:)

integer(4) :: newdim1, newdim2, newdim3
```

LOCAL VARIABLES:

```
integer(4), pointer :: hilfsfeld(:,:,:)

integer(4) :: min1, min2, min3
```

2.0.235 doreallocate_z8_d1

INTERFACE:

```
subroutine doreallocate_z8_d1(tf, newdimension)
```

INPUT PARAMETERS:

```
implicit none

complex(8), pointer :: tf(:)  ! vector to reallocate

integer(4) :: newdimension ! new dimension of the vector
```

DESCRIPTION:

Reallocates a complex(8) pointer vector of range 1

LOCAL VARIABLES:

```
complex(8), pointer :: hilfsfeld(:)

integer(4) :: min1
```

2.0.236 doreallocate_z8_d2

INTERFACE:

```
subroutine doreallocate_z8_d2(tf, newdim1,newdim2)
```

DESCRIPTION:

Reallocates a complex(8) pointer vector of range 2
INPUT PARAMETERS:

```
implicit none

integer(4), intent(in) :: newdim1 ! new dimension of the vector
integer(4), intent(in) :: newdim2 ! new dimension of the vector
```

INPUT/OUTPUT PARAMETERS:

```
complex(8),  pointer :: tf(:,:)  ! vector to reallocate
```

LOCAL VARIABLES:

```
complex(8), pointer :: hilfsfeld(:,:)

integer(4) :: min1,min2
```

2.0.237 doreallocate_z8_d3

INTERFACE:

```
subroutine doreallocate_z8_d3(tf,newdim1,newdim2,newdim3)
```

DESCRIPTION:

Reallocates a complex(8) pointer vector of range 3
INPUT PARAMETERS:

```
implicit none

integer(4), intent(in) :: newdim1 ! new dimension of the vector
integer(4), intent(in) :: newdim2 ! new dimension of the vector
integer(4), intent(in) :: newdim3 ! new dimension of the vector
```

INPUT/OUTPUT PARAMETERS:

```
complex(8), pointer :: tf(:,:,:)  ! vector to reallocate
```

LOCAL VARIABLES:

```
complex(8), pointer :: hilfsfeld(:,:,:)

integer(4) :: min1,min2,min3
```

!MODULE: incgamma module incgamma
LOCAL VARIABLES:

```
    real(8), private :: etx ! exp(-x), declared here because it is used
by all the calls of the function, to avoid
recalculating it every time.

    real(8), private :: sqx ! sqrt(x), declared here because it is used
by all the calls of the function, to avoid
recalculating it every time.
```

INTERFACE:

```
interface gammaincc
  module procedure gammaincc_int
  module procedure gammaincc_hin
end interface gammaincc
```

DESCRIPTION:

This module calculates the incomplete gamma function, defined by:

$$\Gamma(a,x) = \int_x^\infty e^{-t}t^{a-1}dt \tag{26}$$

for values of a integer and halfinteger recursively, according to the equation:

$$\Gamma(a+1,x) = a\Gamma(a,x) + x^a e^{-x} \tag{27}$$

!USAGE:
result=incgam(n,x)
where:

- n (real(8)): order of the gamma function, the number must be integer or half integer
- x (real(8)): point where the gamma function is to be calculated. x must be positive

!EXTERNAL ROUTINES:

- `e1xb`: calculates the exponential integral function
- `erf`: calculates the complementary error function. Depending on the compiler this can also be an internal procedure (ifc 6.0 has it). If not, the file erf.f90 is included in the package.

PUBLIC MEMBER FUNCTIONS:

```
real(8) :: incgam(n,x)
```

!IROUTINE: `gammaincc_int`
INTERFACE:

```
recursive function gammaincc_int(n,x) result(gmi)
```

INPUT PARAMETERS:

```
implicit none
real(8) :: x  ! value at which the gamma function is calculated
integer :: n  ! order of the gamma function
```

OUTPUT PARAMETERS:

```
real(8) :: gmi ! value of the gamma function
```

DESCRIPTION:

Calculates the incomplete gamma function of integer order

!IROUTINE: `gammaincc_hin`
INTERFACE:

```
recursive function gammaincc_hin(in,x,n) result(gmh)
```

INPUT PARAMETERS:

```
implicit none
integer :: in  ! order of the gamma function - 1/2
real(8) :: x   ! value at which the gamma function is calculated
real(8) :: n   ! order of the gamma function
```

OUTPUT PARAMETERS:

```
real(8) :: gmh ! value of the gamma function
```

DESCRIPTION:

Calculates the incomplete gamma function of halfinteger order

---

!IROUTINE: incgam
INTERFACE:

```
function incgam(n,x)
```

INPUT PARAMETERS:

```
implicit none
real(8) :: x     ! Argument of the gamma function
real(8) :: n     ! order of the gamma function
```

OUTPUT PARAMETERS:

```
real(8) :: incgam ! value of the gamma function (result)
```

DESCRIPTION:

The function works as an interface, it checks whether the index is integer or half integer and calls the corresponding procedure within the module
LOCAL VARIABLES:

```
integer(4) :: in ! integer part of n
```

REVISION HISTORY:

```
Created Jan. 2004
```

!ROUTINE: rcutoff
INTERFACE:

```
function rcutoff(tol,eta,lambdamax) result(rcf)
```

DESCRIPTION:

Estimates the cutoff radius of the sums in real space for the calculation of the structure constants by the solving the equation:

$$\mathfrak{E}^{\mathrm{tol}}_{R,\lambda} = \begin{cases} \frac{4\pi}{(\lambda-2)\Gamma(\lambda+\frac{1}{2})}\left(\frac{\Gamma[\frac{\lambda}{2}+\frac{3}{2},\left(\frac{R_c}{\eta}\right)^2]}{\eta^{\lambda-2}} - \frac{\Gamma[\lambda+\frac{1}{2},\left(\frac{R_c}{\eta}\right)^2]}{R_c^{\lambda-2}}\right) & \lambda \neq 2 \\ \frac{4\pi}{\Gamma(\frac{5}{2})}\left[\frac{\eta}{R_c}\Gamma[3,\left(\frac{R_c}{\eta}\right)^2] - \Gamma[\frac{5}{2},\left(\frac{R_c}{\eta}\right)^2]\right] & \lambda = 2 \end{cases} \tag{28}$$

and taking the maximum value of R_c obtained for $\lambda = 1...$ `lambdamax`.
USES:

```
use modmain,     only: pi
use mod_misc_gw, only: avec
use incgamma,    only: incgam
```

INPUT PARAMETERS:

```
implicit none
real(8), intent(in) :: tol ! The tolerance for the convergence of the lattice sum
real(8), intent(in) :: eta
integer, intent(in) :: lambdamax
```

OUTPUT PARAMETERS:

```
real(8) :: rcf ! The maximum cutoff radius
```

LOCAL VARIABLES:

```
integer(4) :: l1
integer(4) :: i
real(8) :: rl
real(8) :: x
real(8) :: gmm
real(8) :: aleng(3)
real(8), allocatable :: eps(:)
real(8) :: rnot
real(8) :: gaml12
real(8) :: gaml32
real(8) :: prefac
real(8), allocatable :: rct(:)

!EXTERNAL ROUTINES:
  real(8), external :: higam
```

REVISION HISTORY:

```
Created: January 2004 by RGA
Last Modified: 6. August 2004 by RGA
Revisited: 23 May 2011 by DIN
```

!ROUTINE: gcutoff
INTERFACE:

```
function gcutoff(tol,eta,lambdamax) result(rcf)
```

DESCRIPTION:

Estimates the cutoff radius of the sums in reciprocal space for the calculation of the structure constants by the solving the equation:

$$\mathfrak{E}^{\mathrm{tol}}_{G,\lambda} = \frac{8(\pi)^{\frac{5}{2}}}{\Omega\Gamma(\lambda+\frac{1}{2})\eta^{\lambda+1}}\Gamma\left[\frac{\lambda+1}{2},\left(\frac{\eta G_c}{2}\right)^2\right] \tag{29}$$

and taking the maximum value of G_c obtained for $\lambda = 1...$ `lambdamax`.

USES:

```
use modmain,  only: pi, omega, bvec
use incgamma, only: incgam
```

INPUT PARAMETERS:

```
implicit none
real(8), intent(in) :: tol ! The tolerance for the convergence of the lattice sum
real(8), intent(in) :: eta
integer, intent(in) :: lambdamax
```

OUTPUT PARAMETERS:

```
real(8) :: rcf             ! The maximum cutoff radius
```

LOCAL VARIABLES:

```
integer(4) :: l1
integer(4) :: i
real(8) :: gaml32
real(8) :: gmm
real(8) :: prefac
real(8) :: rl
real(8) :: rnot
real(8) :: x
real(8) :: bleng(3)
real(8), allocatable :: eps(:)
real(8), allocatable :: rct(:)

!EXTERNAL ROUTINES:
real(8), external :: higam
```

REVISION HISTORY:

```
Created: January 2004 by RGA
Last Modified: 6. August 2004 by RGA
Revisited: 23 May 2011 by DIN
```

!ROUTINE: calcbradket
INTERFACE:

```
subroutine calcbradket(ia,is)
```

DESCRIPTION:

This subroutine calculates the set of integrals:

$$\langle LM\tilde{\lambda}|\tilde{\lambda}'\rangle_a \equiv \int_0^{R_{MT}^a} \upsilon_{aLM}(r^a)\tilde{u}_\lambda^*(r^a,E_\lambda)\tilde{u}_{\lambda'}(r^a,E_{\lambda'})\,(r^a)^2\,dr^a \tag{30}$$

where $\tilde{u}$ can be u, $\dot{u}$ or u^{lo}
USES:

```
    use modinput
    use modmain
    use modgw

!INPUT VARIABLES:
    implicit none
    integer(4), intent(in) :: ia
    integer(4), intent(in) :: is
```

LOCAL VARIABLES:

```
    integer(4) :: ias
    integer(4) :: ilo1, ilo2
    integer(4) :: io1, io2
    integer(4) :: ir, irm
    integer(4) :: ist1, ist2
    integer(4) :: l1, l2
    real(8) :: fr(nrmtmax)
    real(8) :: gr(nrmtmax)
    real(8) :: cf(3,nrmtmax)
```

DEFINED PARAMETERS:

```
    character(4) :: ftype(3)
    character(4) :: utype(3)
    data ftype /'core','apw ','lo  '/
    data utype /'    ','dot ','ddot'/
```

REVISION HISTORY:

```
Created Dic 2003
Last Modified: May. 2006
Revisited August 2012 by DIN
```

!ROUTINE: calcpmatgw **INTERFACE:**

```
subroutine calcpmatgw
```

USES:

```
use modinput
use modmain
use modgw
use m_getunit
use modmpi
use mod_pmat
use mod_hdf5
```

DESCRIPTION:

Calculates the momentum matrix elements using routine `genpmat` and writes them to direct access file `PMATVV.OUT` or `PMATCV.OUT`.

REVISION HISTORY:

```
Created October 2013 (DIN)
```

!ROUTINE: calcselfc
INTERFACE:

```
subroutine calcselfc(iq)
```

DESCRIPTION:

This subroutine calculates the q-dependent correaltion self-energy

USES:

```
use modinput
use modmain,    only : nstsv, apwordmax, lmmaxapw, natmtot, nspnfv, &
&                      zzero, nmatmax
use modgw
use mod_mpi_gw, only : myrank, myrank_col
use m_getunit
```

INPUT PARAMETERS:

```
implicit none
integer(4), intent(in) :: iq
```

LOCAL VARIABLES:

```
integer(4) :: ik, ikp, jk, ispn
integer(4) :: mdim
integer(4) :: fid
character(120) :: fname_mwm
real(8) :: tstart, tend, t0, t1
complex(8), allocatable :: evecsv(:,:,:)
character(len=10), external :: int2str
```

REVISION HISTORY:

Created Nov 2013 by DIN

!ROUTINE: expand_evec
INTERFACE:

```
subroutine expand_evec(ik,trans)
```

DESCRIPTION:

Calculate the product of an eigenvector with the corresponding matching coefficients
USES:

```
use modinput
use modmain, only : ngkmax, apwordmax, lmmaxapw, natmtot, &
&                   nspecies, natoms, idxas, idxlm, apword, nstsv, &
&                   lsplsymc, isymlat, symlatc
use modgw,   only : kqset, Gkset, eveckalm, eveckpalm, eveck, eveckp
```

INPUT PARAMETERS:

```
implicit none
integer(4),   intent(in) :: ik    ! index of the k-point
character(1), intent(in) :: trans
```

LOCAL VARIABLES:

```
integer(4) :: ia, is, ias
integer(4) :: io
integer(4) :: ist
integer(4) :: l, m, lm, m1
integer(4) :: igk, ngk
integer(4) :: isym, lspl, ilspl
real(8)    :: c(3,3)
complex(8), allocatable :: apwalm(:,:,:,:), alm(:,:,:,:)
```

```
!EXTERNAL ROUTINES:
   complex(8), external :: zdotc
   complex(8), external :: zdotu
```

REVISION HISTORY:

```
Created Dic. 2003 by RGA
Last Modification: July. 20th. 2004 by RGA
Revision: 5.05.2011 by DIN
```

2.0.238 shelsort (Source File: shelsort.f90)

INTERFACE:

```
      subroutine shelsort(n,v,len)
```

DESCRIPTION:

Sorts a set of vectors (v) by increasing length (len) using the shell algorithm.

INPUT PARAMETERS:

```
      implicit none
      integer(4), intent(in) :: n       ! number of vectors to sort
```

INPUT/OUTPUT PARAMETERS:

```
      integer(4), intent(inout) :: v(3,n) ! set of vectors
      real(8),    intent(inout) :: len(n) ! set of lengths
```

LOCAL VARIABLES:

```
      integer(4) :: gap
      integer(4) :: i
      integer(4) :: vtemp(3)

      real(8) :: ltemp

      logical :: done
```

REVISION HISTORY:

```
Created Nov 2006 by RGA
```

2.0.239 genpmat (Source File: genpmatcor.f90)

INTERFACE:

```
subroutine genpmatcor(vpl,ngp,apwalm,evecfv,evecsv,pmc)
```

USES:

```
use modinput
use modmain
use modgw
```

INPUT/OUTPUT PARAMETERS:

```
vpl    : k-vector (in,real(3))
ngp    : number of G+p-vectors (in,integer)
apwalm : APW matching coefficients
         (in,complex(ngkmax,apwordmax,lmmaxapw,natmtot))
evecfv : first-variational eigenvector (in,complex(nmatmax,nstfv))
evecsv : second-variational eigenvectors (in,complex(nstsv,nstsv))
pmc    : momentum matrix elements (out,complex(3,nstsv,nstsv))
```

DESCRIPTION:

Calculates the momentum matrix elements

$$p_{ij} = \langle \Psi_{i,\mathbf{k}} | -i\nabla | \Psi_{j,\mathbf{k}} \rangle.$$

REVISION HISTORY:

```
Created August 2006 (RGA)
Revisited June 2011 (DIN)
```

!ROUTINE:
INTERFACE:

```
subroutine genmbrotmat(iq,isym)
```

DESCRIPTION:

The space group symmetry operations in MB representation
USES:

```
use modinput
use modmain
use modgw
```

INPUT PARAMETERS:

```
implicit none
integer(4), intent(in) :: iq
integer(4), intent(in) :: isym
```

LOCAL VARIABLES:

```
integer(4) :: ia, is, ias
integer(4) :: ja, jas
integer(4) :: lspl
integer(4) :: irm, jrm
integer(4) :: l1, l2, m1, m2
integer(4) :: imix,jmix
integer(4) :: im,jm
integer(4) :: iqp
integer(4) :: ig, igq1, igq2
integer(4) :: s(3,3)
integer(4) :: g0(3), g1(3), g2(3), g3(3)
real(8)    :: c(3,3)
real(8)    :: t1
real(8)    :: tstart, tend
real(8)    :: v(3), v1(3)
complex(8) :: zt1
complex(8), allocatable :: tmat1(:,:), tmat2(:,:)
complex(8), allocatable :: sgip(:,:)

fortran function
  complex(8) :: getdlmm
```

REVISION HISTORY:

```
Created October 2011 by DIN
```

2.0.240 tildeg (Source File: tildeg.f90)

INTERFACE:

```
real(8) function tildeg(l1,l2,m1,m2)
```

DESCRIPTION:

This subroutine calculates the coefficients $\tilde{g}_{lm,l'm'}$ according to equation **??**. The factor $(4\pi)^{\frac{3}{2}}$ is included into $\tilde{g}_{lm,l'm'}$.

INPUT PARAMETERS:

```
implicit none

integer(4) :: l1,l2,m1,m2
```

LOCAL VARIABLES:

```
      integer(4), dimension(4) :: jm        ! value of l1+m1
      integer(4) :: tjpo    ! value of 2(l1+l2)+1
      real(8) :: combjpm    ! value of (l1+l2+m1+m2)!/(l1+m1)!(l2+m2)!
      real(8) :: combjmm    ! value of (l1+l2-m1-m2)!/(l1-m1)!(l2-m2)!
      real(8) :: denom
      real(8), parameter :: pi = 3.1415926536d0

  !EXTERNAL ROUTINES:

      real(8), external :: factr
      real(8), external :: combin
```

!ROUTINE: calcselfx
INTERFACE:

```
subroutine calcselfx(iq)
```

DESCRIPTION:

This subroutine calculates the q-dependent self energy contribution
USES:

```
    use modinput
    use modmain,    only : nstfv, apwordmax, lmmaxapw, natmtot, nspnfv, &
    &                      pi, idxas, zzero, nmatmax, zone, occsv
    use modgw
    use mod_mpi_gw, only : myrank
    use mod_hdf5
```

INPUT PARAMETERS:

```
    implicit none
    integer(4), intent(in) :: iq
```

LOCAL VARIABLES:

```
    integer(4) :: ik, ikp, jk
    integer(4) :: mdim, nmdim
    real(8)    :: tstart, tend, t0, t1
    integer(4) :: ie1, ie2, im
    integer(4) :: ia, is, ias, ic, icg
    real(8)    :: wkq, sxs2
    complex(8) :: sx, vc
    complex(8) :: mvm     ! Sum_ij{M^i*V^c_{ij}*conjg(M^j)}
```

```
    complex(8), allocatable :: evecsv(:,:,:)

    integer :: k, l, ispn, ist, jst
    complex(8) :: zsum

  external routines
    complex(8), external :: zdotc
```

REVISION HISTORY:

```
  Created Nov 2013 by DIN
```

2.0.241 calceta (Source File: calceta.f90)

INTERFACE:

```
function calceta() result(neta)
```

DESCRIPTION:

This function calculates the optimal value of η for the lattice summations needed to obtain the structure constants.

USES:

```
    use modmain,     only: bvec, pi
    use mod_misc_gw, only: avec
```

LOCAL VARIABLES:

```
    implicit none
    integer(4) :: i
    real(8) :: mlg
    real(8) :: mlr
    real(8) :: neta
    real(8), dimension (3)   :: lgbs
    real(8), dimension (3)   :: lrbs
```

REVISION HISTORY:

```
  Created 7th. January 2004 by RGA
  Last Modified: 30th. March 2004 by RGA
  Revisited Oct 2013 by DIN
```

2.0.242 writeqpts (Source File: gw_writeqpts.f90)

INTERFACE:

```
subroutine gw_writeqpts
```

USES:

```
use modmain
use modgw
```

DESCRIPTION:

Writes the **q**-points in lattice coordinates, weights and number of $\mathbf{G}+\mathbf{q}$-vectors to the file `QPOINTS.OUT`.

REVISION HISTORY:

```
Created May 2006 (RGA)
```

!ROUTINE: setbarcev

INTERFACE:

```
subroutine setbarcev(evtol)
```

DESCRIPTION:

This subroutine reset the eigenvectors and eigenvalues of bare coulomb matrix in terms of evtol

USES:

```
use modmain,              only : zone, zzero
use modgw
use mod_mpi_gw,           only : myrank
```

INPUT PARAMETERS:

```
implicit none
real(8), intent(in) :: evtol
```

LOCAL VARIABLES:

```
integer(4) :: im, jm
integer(4) :: immax
real(8) :: test1, test2
complex(8) :: vc
integer(4), allocatable :: im_kept(:) ! indicate which barc eigenvectors
  are kept as basis functions
complex(8), allocatable :: wi0new(:)
```

REVISION HISTORY:

```
Created July 31,2009 by Hong Jiang
Readjusted Jan, 2012 by DIN
```

!ROUTINE: sigma
subroutine sigma(iq,lambdamax)

This subroutine calculates the lattice sums $\Sigma^{a,a'}_{\lambda,\mu}(\vec{q}) = \sum_{\vec{R}} \frac{e^{i\vec{q}\cdot\left(\vec{R}+\vec{r}_{aa'}\right)}}{|\vec{R}+\vec{r}_{aa'}|^{(\lambda+1)}} Y_{\lambda\mu}\left(\hat{R}_{aa'}\right)$ using the method described in appendix **??**, in particular equation **??**.
USES:

```
  use modinput
  use modmain, only : nspecies, natoms, idxas, atposc, &
  &                   zzero, bvec, pi, natmtot, zi
  use modgw,   only : kqset, fdebug
  use mod_coulomb_potential, only : sgm
  use mod_misc_gw, only : avec, vi
  use strconst
```

INPUT PARAMETERS:

```
  implicit none
  integer(4), intent(in) :: iq        ! index of the q-point for which
sigma is calculated
  integer(4), intent(in) :: lambdamax ! Maximum value of lambda
```

LOCAL VARIABLES:

```
  integer(4) :: np ! Number of points for the real space
summation
  integer(4) :: ng ! Number of points for the reciprocal
space summation
  integer(4) :: ia, ias, is, ja, jas, js
  integer(4) :: i1
  integer(4) :: lmbd
  integer(4) :: mu
  integer(4) :: lmuind

  real(8) :: eta
  real(8) :: rcf
  real(8) :: gcf
  real(8) :: rleng              ! the length of rpaa
  real(8) :: qtraa              ! the scalar product qvec.rpaa
  real(8) :: gausr              ! value of the gaussian function e^(-(rleng/eta)^2)
  real(8) :: pref               ! prefactor for the reciprocal lattice sum = 4 pi^(3/2)/v
  real(8) :: gleng              ! the length of gqv
```

```
  real(8) :: gqtra               ! the scalar product gqv.raa
  real(8) :: gausg               ! value of the gaussian function e^(-(gleng*eta/2)^2)
  real(8) :: gtolam              ! value of gleng^(lambda-2)/2^(lambda-0.5)
  real(8) :: gamlam              ! gamlam = Gamma[lambda+1/2]
  real(8) :: erfr
  real(8) :: gammaor

  real(8), dimension(3) :: qvec  ! cartesian coords. of the q point
  real(8), dimension(3) :: raa   ! Vector going from atom 1 to atom 2.
  real(8), dimension(3) :: rpaa  ! the corresponding sum R+raa
  real(8), dimension(3) :: qtemp
  real(8), dimension(3) :: g     ! vector belonging to the reciprocal space lattice
  real(8), dimension(3) :: gqv   ! the corresponding sum G+qvec

  complex(8) :: ylam((lambdamax+1)*(lambdamax+1)) ! the values of the spherical harmonics
  complex(8) :: stmp1((lambdamax+1)*(lambdamax+1)) ! temporary allocation of the values of s
  complex(8) :: stmp2((lambdamax+1)*(lambdamax+1)) ! temporary allocation of the values of s
  complex(8) :: expqdr           ! e^(qvec.rpaa)
  complex(8) :: itolam           ! i^lambda
  complex(8) :: term1

!EXTERNAL ROUTINES:
  real(8), external :: derfc
  real(8), external :: calceta
  real(8), external :: gcutoff
  real(8), external :: rcutoff
```

REVISION HISTORY:

```
Created 21. Jan. 2004 by RGA
Last Modified 3. Nov 2005 by RGA
Revisited: May 2011 by DIN
```

!ROUTINE: genrstr
INTERFACE:

```
subroutine genrstr(rmax,rshift,rbas,nr)
```

DESCRIPTION:

Generates the indexes of the lattice vectors to be included in the calculation of the structure constants, under the condition:

$$|\vec{R} + \vec{r}_{aa'}| \leq R_{cutoff} \tag{31}$$

USES:

```
use strconst
```

INPUT PARAMETERS:

```
implicit none
real(8), intent(in) :: rmax      ! Maximum radius
real(8), intent(in) :: rshift(3) ! Shift of the origin
real(8), intent(in) :: rbas(3,3) ! Bravais lattice basis
```

OUTPUT PARAMETERS:

```
integer(4), intent(out) :: nr    ! number of vectors
```

LOCAL VARIABLES:

```
integer(4) :: i, imax, ir, rdim, i1, i2, i3, gap
real(8) :: lrmin                 ! minimum length of the basis vectors.
real(8) :: rleng
real(8), dimension(3) :: r       ! vector belonging to the real space lattice
real(8), dimension(3) :: lrbs    ! length of the basis vectors
real(8), dimension(3) :: rps
real(8), dimension(4) :: rtmp
logical :: done
```

REVISION HISTORY:

```
Created 5. Aug. 2004 by RGA
Revisited Oct. 2013 by DIN
```

!ROUTINE: qdepwtet
INTERFACE:

```
subroutine qdepwtet(iq,iomstart,iomend,ndim)
```

DESCRIPTION:

This subroutine calculates the weights for q dependent BZ integration using LIBBZINT
USES:

```
use modinput
use modmain, only : natmtot, nstsv, zzero, nspecies, natoms, idxas, &
&                   evalcr, efermi, evalsv
use modgw,   only : fnm, freq, ncmax, kset, kqset, nomax, numin, &
&                   ncg, corind, fdebug, time_bzinit
```

INPUT PARAMETERS:

```
implicit none
integer(4), intent(in) :: iq
integer(4), intent(in) :: iomstart, iomend
integer(4), intent(in) :: ndim
```

LOCAL VARIABLES:

```
integer(4) :: ik, ikp, ib, ic, icg
integer(4) :: ia, is, ias
integer(4) :: iom
integer(4) :: fflg, sgw

real(8) :: emaxb ! maximum energy of the second band
real(8) :: edif, edsq, omsq

real(8), allocatable :: eval(:,:)
real(8), allocatable :: cwpar(:,:,:)
real(8), allocatable :: cwparsurf(:,:,:)

real(8) :: tstart, tend
```

2.0.243 e1xb (Source File: eixb.f90)

INTERFACE:

```
function e1xb(x) result(e1)
```

DESCRIPTION:

compute the exponential integral function e1(x)

INPUT PARAMETERS:

```
implicit none

real(8) :: x
```

OUTPUT PARAMETERS:

```
real(8) :: e1
```

LOCAL VARIABLES:

```
integer(4) :: k
integer(4) :: m

real(8)    :: r
real(8)    :: ga
real(8)    :: t0
real(8)    :: t
```

!ROUTINE: calcmpwmix

INTERFACE:

```
subroutine calcmpwmix(iq)
```

DESCRIPTION:

This subroutine calculates the matrix elements between mixed basis functions and plane waves.

USES:

```
use modmain,                only : pi, nspecies, nrmt, spr, natoms, &
&                                  idxas, idxlm, zi, zzero, zone
use modgw,                  only : Gset, Gamma, kqset, Gqset, Gqbarc
use mod_product_basis,      only : matsiz, maxbigl, nmix, bigl, umix, &
&                                  locmatsiz, mpwipw, mpwmix
use mod_coulomb_potential, only : wi0
use mod_misc_gw,            only : vi, atposl, alat
```

INPUT PARAMETERS:

```
implicit none
integer(4), intent(in) :: iq
```

LOCAL VARIABLES:

```
integer :: npw, ipw, ipw0  ! PW
integer :: ngq, igq        ! IPW
integer :: imix, is, ia, ias, irm, l1, m1, l1m1
integer :: ir, nr
real(8) :: const, x
real(8) :: gvec(3), gqvec(3), gqlen, gpr
real(8) :: janl
real(8), allocatable :: fr(:), gr(:), cf(:,:)
real(8), allocatable :: bessl(:,:)
complex(8) :: expg, prefac
complex(8) :: sph((maxbigl+1)*(maxbigl+1))
complex(8), allocatable :: tmat1(:,:), tmat2(:,:)
```

REVISION HISTORY:

```
Created: Mar 2014 by DIN
```

2.0.244 gammln (Source File: gammln.f90)

INTERFACE:

```
real(8) function gammln(xx)
```

DESCRIPTION:

Returns the value $\ln[\Gamma(\texttt{xx})]$ for $\texttt{xx} > 0$.

INPUT PARAMETERS:

```
implicit none

real(8), intent(in) :: xx
```

LOCAL VARIABLES:

```
integer(4) :: j

real(8) :: ser
real(8) :: tmp
real(8) :: x
real(8) :: y
```

DEFINED PARAMETERS:

```
real(8), save :: stp
real(8), save :: cof(6)

 data cof /76.18009172947146d0,-86.50532032941677d0,              &
&          24.01409824083091d0,-1.231739572450155d0,              &
&          .120865097386617 9d-2,-.5395239384953d-5/
 data stp /2.5066282746310005d0/



!INTRINSIC ROUTINES:



intrinsic log
```

REVISION HISTORY:

```
Original subroutine: gauleg.for (C) copr. 1986-92 copr. 1986-92
numerical recipes software pp 207..
Last modified: 20.06.05 by RGA.
```

2.0.245 getdlmm (Source File: getdlmm.f90)

INTERFACE:

```
complex(8) function getdlmm(rot,l,m1,m2)
```

INPUT/OUTPUT PARAMETERS:

```
rot   : rotation matrix (in,real(3,3))
l     : angular momentum (in,integer)
m1    :
m2    :
```

DESCRIPTION:

Calculates the rotational matrix $D^l_{mm'}$ for the given rotation matrix R. This is done by first the computing the Euler angles (α, β, γ) of R (see routine `euler`) and then generating the rotation matrix for spherical harmonics, $D^l_{mm'}(\alpha, \beta, \gamma)$, with which

$$Y_{lm}(\theta', \phi') = \sum_{m'} D^l_{mm'}(\alpha, \beta, \gamma) Y_{lm'}(\theta, \phi),$$

where (θ', ϕ') are the angles (θ, ϕ) rotated by R. The matrix D is given explicitly by

$$D^l_{mm'}(\alpha, \beta, \gamma) = \sum_i \frac{(-1)^i \sqrt{(l+m)!(l-m)!(l+m')!(l-m')!}}{(l-m'-i)!(l+m-i)!i!(i+m'-m)!} \times \left(\cos\frac{\beta}{2}\right)^{2l+m-m'-2i} \left(\sin\frac{\beta}{2}\right)^{2i+m'-m} e^{-i(m\alpha+m'\gamma)},$$

where the sum runs over all i which make the factorial arguments non-negative. For improper rotations, i.e. those which are a combination of a rotation and inversion, the rotation is first made proper with $R \to -R$ and D is modified with $D^l_{mm'} \to (-1)^l D^l_{mm'}$.

REVISION HISTORY:

```
Created October 2011 by DIN based on rotzflm.f90
Important changes: In rotzflm.f90 D^l_{mm'}(R^-1) is calculated
                   I need D^l_{mm'}(R)
```

2.0.246 gensmallq (Source File: gensmallq.f90)

INTERFACE:

```
SUBROUTINE gensmallq
```

USES:

```
use modinput
use modmain
use modgw
```

DESCRIPTION:

Generates the small group of q-vectors, i.e., all R's that $Rq = q+G_R$. Then the q-dependent BZ(q) is searched. As a last step the subroup of R is determined which regenerate from k=BZ(q) the full BZ.

LOCAL VARIABLES:

```
implicit none

integer :: iqp, ikp  ! q-, k-point indexes for IBZ
integer :: ik
integer :: ip, jp, iv(3)
integer :: isym, lspl, nsym

real(8) :: s(3,3), v1(3), v2(3), t1

real(8), allocatable :: vklq(:,:)
integer, allocatable :: scmapq(:)
integer, allocatable :: ivwrapq(:,:)
integer, allocatable :: iwkpq(:)

Real (8), External :: r3taxi
```

REVISION HISTORY:

```
Created October 2011 (DIN)
```

!ROUTINE: calcminm

INTERFACE:

```
subroutine calcminm2(ik,iq,nstart,nend,mstart,mend,minm)
```

DESCRIPTION:

This subroutine calculates the matrix elements $M^i_{nm}(\vec{k},\vec{q})$

USES:

```
    use modinput
    use modmain,               only : nspecies, natoms, idxas, idxlm, idxlo, &
    &                                 zzero, zone, intgv, apword, nlorb, lorbl, &
    &                                 nlomax, pi, apwordmax, nmatmax
    use modgw,                 only : kqset, Gkset, Gqbarc, Gqset, Gset, fdebug, time_minm
    use mod_bands,             only : eveck, eveckp, eveckalm, eveckpalm
    use mod_product_basis,     only : nmix, bigl, bradketa, bradketlo, mpwipw, &
    &                                 matsiz, locmatsiz, mbindex
    use mod_misc_gw,           only : vi, atposl
    use mod_gaunt_coefficients
```

INPUT PARAMETERS:

```
    implicit none
    integer(4), intent(in) :: ik   ! the index of the first k-point
    integer(4), intent(in) :: iq   ! the index of the q-point
    integer(4), intent(in) :: nstart, nend  ! range of n states
    integer(4), intent(in) :: mstart, mend  ! range of m states
    complex(8), intent(out):: minm(matsiz,nstart:nend,mstart:mend)
```

LOCAL VARIABLES:

```
    integer(4) :: jk
    integer(4) :: bl, bm
    integer(4) :: i, ia, is, ias
    integer(4) :: igk1, igk2
    integer(4) :: io1, io2, ilo1, ilo2
    integer(4) :: imix, irm
    integer(4) :: ie1, ie2
    integer(4) :: l1, m1, l2, m2, l1m1, l2m2
    integer(4) :: l2min, l2max
    integer(4) :: ig, igq
    integer(4), allocatable :: igqk12(:,:)
    integer(4), dimension(3):: ikv, ig0 ! Indexes of G_1+G'-G
    integer(4) :: ngk1, ngk2
    integer(4) :: ndim, mdim, nmdim

    real(8) :: sqvi, x, arg, angint
    real(8) :: qvec(3)
    real(8) :: tstart, tend, tmt

    real(8) :: t1, t2

    complex(8) :: phs, bk
    complex(8), allocatable :: tmat(:,:), tmat2(:,:), mnn(:,:)
    complex(8), allocatable :: lok(:,:),lokp(:,:) ,veck(:),veckp(:)
  dir$ attributes align:64 :: tmat
```

```
dir$ attributes align:64 :: lok
dir$ attributes align:64 :: lokp

!EXTERNAL ROUTINES:
   external :: zgemm
   real(8), external :: gaunt
```

REVISION HISTORY:

```
Created  23th. Feb. 2004 by RGA
Last modified 20th. Jul. 2004 by RGA
Revisited 29.04.2011 by DIN
```

2.0.247 qdepw (Source File: qdepwsum.f90)

INTERFACE:

```
subroutine qdepwsum(iq)
```

DESCRIPTION:

This subroutine calculates the weights for q dependent BZ integrations. **needs checking!!**
USES:

```
use modmain
use modgw
```

INPUT PARAMETERS:

```
implicit none

integer(4) :: iq  !ID. number of the q-point
```

LOCAL VARIABLES:

```
integer(4) :: ib  ! counter, run over bands
integer(4) :: ic  ! counter, run over core states
integer(4) :: jb  ! counter, run over bands
integer(4) :: iom ! counter, run over frequencies
integer(4) :: ik,ia,is,ias,jk
integer(4) :: ikp,jkp
integer(4) :: fflg,sgw
real(8) :: edif,edsq,omsq,wk
real(8) :: tstart,tend
```

!ROUTINE: readselfc
INTERFACE:

```
subroutine readselfc
```

DESCRIPTION:

This subroutine reads the correlation self-energy from file
USES:

```
use modgw, only : kset, freq, ibgw, nbgw, selfec
use m_getunit
```

LOCAL VARIABLES:

```
implicit none
integer(4) :: ib, nb, nk, no
integer(4) :: fid
```

REVISION HISTORY:

```
Created June 2011 by DIN
```

!ROUTINE: calcrlamint
INTERFACE:

```
subroutine calcrlamint(ia,is)
```

DESCRIPTION:

This subroutine calculates the set of integrals:

$$\texttt{rtl(ias,i)} \equiv \left\langle r^b l \right\rangle_{aNL} = \int_0^{R^a_{MT}} (r^a)^{bl+2} \upsilon_{aNL}(r^a) dr^a \tag{32}$$

and

$$\texttt{rint(ias,N,N',bl)} \equiv \left\langle \begin{array}{c} r_<^{bl} \\ r_>^{bl+1} \end{array} \right\rangle_{aNL,N'bl} = \iint_0^{R^a_{MT}} \upsilon_{aNL}(r^a_1) \frac{r_<^b l}{r_>^{bl+1}} \upsilon_{aNL}(r^a_2)(r^a_1)^2 dr^a_1 (r^a_2)^2 dr^a_2 \tag{33}$$

that will be needed for the calculation of the bare coulomb potential
USES:

```
    use modmain
    use modgw

!INPUT VARIABLES:
    implicit none
    integer(4), intent(in) :: ia
    integer(4), intent(in) :: is
```

LOCAL VARIABLES:

```
    integer(4) :: ias
    integer(4) :: irm  ! Counter: Runs over radial mixed basis functions.
    integer(4) :: jrm  ! Counter: Runs over radial mixed basis functions.
    integer(4) :: ijrm ! Joint index for (irm,jrm) pairs for compressed storage
    integer(4) :: bl   ! Angular momentum quantum number of the mixed basis functions
    integer(4) :: l1   ! just a counters
    integer(4) :: ir   ! Counter: Runs over radial mesh point.
    real(8) :: rxov    ! value of the integral

    real(8), allocatable :: rrtol(:)
    real(8), allocatable :: ui(:)
    real(8), allocatable :: uj(:)
    real(8), allocatable :: fr(:)
    real(8), allocatable :: gr(:)
    real(8), allocatable :: cf(:,:)
```

REVISION HISTORY:

```
Created Dic. 2003 by RGA
Last Modification: July. 20th. 2004 by RGA
Revisited 5.05.2011 by DIN
```

!ROUTINE: calcepsilon
INTERFACE:

```
subroutine calcepsilon(iq,iomstart,iomend)
```

DESCRIPTION:

This subroutine calculates the dielectric matrix in the RPA approximation using symmetry
USES:

```
    use modinput
    use modmain, only : zone, zzero
    use modgw
    use mod_mpi_gw, only : myrank
```

```
use modxs,      only : symt2
use m_getunit
```

INPUT PARAMETERS:

```
implicit none
integer(4), intent(in) :: iq
integer(4), intent(in) :: iomstart, iomend
```

LOCAL VARIABLES:

```
integer(4) :: ia, is, ias
integer(4) :: ie1, ie2, ie12, ic, icg
integer(4) :: iom
integer(4) :: ik, jk, ispn
integer(4) :: im, jm, iop, jop

integer(4) :: ndim, mdim, nmdim

integer(8) :: recl

real(8)    :: tstart, tend, t0, t1

complex(8) :: coefb
complex(8) :: head(3,3)
complex(8), allocatable :: minm(:,:,:)
complex(8), allocatable :: evecsv(:,:,:)

!EXTERNAL ROUTINES:
  external zgemm
```

REVISION HISTORY:

```
Created Nov 2013 by DIN
```

!ROUTINE: calcscgw0
INTERFACE:

```
subroutine calcscgw0
```

DESCRIPTION:

This subroutine performs gw calculatons with more radial MPI parallelization including parallelization with respect to both q-points and frequency mesh
USES:

```
use modinput
use modmain,    only : zzero, nstsv, evalsv, efermi
use modgw,      only : kset, kqset, freq,  &
&                      nomax, numin, nbandsgw, ncg, &
&                      ibgw, nbgw, time_selfc, time_io, fgw
use mod_selfenergy, only : evalks, evalqp, eferqp, selfec, mwm
use mod_mpi_gw
use m_getunit
```

LOCAL VARIABLES:

```
implicit none
integer(4) :: ikp, iq, mdim
integer(4) :: isc, nscmax
integer(4) :: fid
character(120) :: fname_mwm
real(8)    :: deltae
real(8)    :: egk(kset%nkpt), egkold(kset%nkpt)
real(8)    :: epsilon_sc
real(8)    :: tstart, tend, t0, t1

character(len=10), external :: int2str
```

REVISION HISTORY:

```
Created 16.09.2008 by Hong Jiang
Readjusted Dec. 2013 by DIN
```

!ROUTINE: readselfx
INTERFACE:

```
subroutine readselfx
```

DESCRIPTION:

This subroutine reads the exchange self-energy from file
USES:

```
use modgw,   only : kset, ibgw, nbgw, selfex
use m_getunit
```

LOCAL VARIABLES:

```
integer(4) :: ib, nb, nk
integer(4) :: fid
```

REVISION HISTORY:

`Created June 2011 by DIN`

2.0.248 ratfun (Source File: ratfun.f90)

INTERFACE:

```
subroutine ratfun(x,a,y,dyda,hess,n)
```

DESCRIPTION:

Given the abscissa `x`, and the parameters `a(1:4n+4)` this subroutine returns in `y` the value of the function:

$$f(x,\{c\}) = \frac{P(x,\{c\})}{Q(x,\{c\})} \tag{34}$$

where

$$P(x,\{c\}) = \sum_{k=1}^{n+1} c_k x^{k-1}, \tag{35}$$

$$Q(x,\{c\}) = 1 + \sum_{k=n+2}^{2n+2} c_k x^{k-n-1} \tag{36}$$

and $c_k =$ `a(k)` $+ i$`a(k+2n+2)`. In `dyda` the partial derivatives of the function with respect to the parameters:

$$\frac{\partial f(x,\{c\})}{\partial c_j} = \begin{cases} \dfrac{x^{j-1}}{Q(x,\{c\})} & 1 \le j \le n+1 \\ -\dfrac{P(x,\{c\})}{[Q(x,\{c\})]^2} x^{j-n-1} & n+2 \le j \le 2n+2 \end{cases} \tag{37}$$

The algorithm for calculating the derivatives is:

$$\frac{\partial f(x,\{c\})}{\partial c_1} = \frac{1}{Q(x,\{c\})} \tag{38a}$$

$$\frac{\partial f(x,\{c\})}{\partial c_{n+2}} = -\frac{f(x,\{c\})}{Q(x,\{c\})} \tag{38b}$$

$$\frac{\partial f(x,\{c\})}{\partial c_j} = \begin{cases} \dfrac{\partial f(x,\{c\})}{\partial c_{j-1}} x & 2 \le j \le n+1 \\ -f(x,\{c\}) \dfrac{\partial f(x,\{c\})}{\partial c_{j-n-1}} & n+3 \le j \le 2n+2 \end{cases} \tag{38c}$$

The hessian is returned in the array `hess`, and is calculated as:

$$\frac{\partial^2 f(x,\{c\})}{\partial c_k \partial c_j} = \begin{cases} 0 & 1 \le j,k \le n+1 \\ -\dfrac{x^{j-1+k-n-1}}{[Q(x,\{c\})]^2} & 1 \le j \le n+1 < k \le 2n+2 \\ 2\dfrac{P(x,\{c\})}{[Q(x,\{c\})]^3} x^{j+k-2n-2} & n+2 \le j,k \le 2n+2 \end{cases} \tag{39}$$

which are calculated as:

$$\frac{\partial^2 f(x,\{c\})}{\partial c_k \partial c_j} = \begin{cases} 0 & 1 \le j,k \le n+1 \\ -\dfrac{\partial f(x,\{c\})}{\partial c_j}\dfrac{\partial f(x,\{c\})}{\partial c_{k-n-1}} & 1 \le j \le n+1 < k \le 2n+2 \\ 2f(x,\{c\})\dfrac{\partial f(x,\{c\})}{\partial c_{j-n-1}}\dfrac{\partial f(x,\{c\})}{\partial c_{k-n-1}} & n+2 \le j,k \le 2n+2 \end{cases} \tag{40}$$

The derivatives with respect to the real parameters are easily calculated by using

$$\frac{\partial c_k}{\partial a_j} = \delta_{k,j} + i\delta_{k,j-2n-2} \tag{41}$$

and the chain rule.

$$\frac{\partial f(x,\{c\})}{\partial a_j} = \sum_{k=1}^{2*n+2} \frac{\partial f(x,\{c\})}{\partial c_k}\frac{\partial c_k}{\partial a_j} \tag{42}$$

INPUT PARAMETERS:

```
implicit none


integer(4), intent(in) :: n

complex(8),    intent(in) :: x
real(8),    intent(in) :: a(1:4*n+4)
```

OUTPUT PARAMETERS:

```
complex(8),    intent(out) :: dyda(1:4*n+4)
complex(8),    intent(out) :: hess(1:4*n+4,1:4*n+4)
complex(8),    intent(out) :: y
```

LOCAL VARIABLES:

```
integer(4) :: i
integer(4) :: j
integer(4) :: l
integer(4) :: m

complex(8) :: q
```

```
complex(8) :: p

complex(8) :: ca(1:2*n+2)
```

DEFINED PARAMETERS:

```
complex(8), parameter :: czero = (0.0d0,0.0d0)
complex(8), parameter :: cone = (1.0d0,0.0d0)
complex(8), parameter :: cim = (0.0d0,1.0d0)
```

REVISION HISTORY:

```
Created: 13th. Jul. 2005 by RGA
```

2.0.249 mrqcof (Source File: mrqcof.f90)

INTERFACE:

```
subroutine mrqcof(x,y,ndata,a,ma,alpha,beta,nca,          &
&                  chisq)
```

DESCRIPTION:

Used by `mrqmin` to evaluate the linearized fitting matrix `alpha`, and vector `beta` as:

$$\beta_k \equiv -\frac{1}{2}\frac{\partial \chi^2}{\partial a_k} \quad \alpha_{kl} \equiv \frac{1}{2}\frac{\partial^2 \chi^2}{\partial a_k \partial a_l}, \tag{43}$$

and calculate χ^2.

INPUT PARAMETERS:

```
implicit none

integer(4), intent(in) :: ndata ! Number of datapoints to be fitted
integer(4), intent(in) :: ma    ! Number of coefficients
integer(4), intent(in) :: nca   ! Size of working-space arrays

real(8),    intent(in) :: x(*)  ! abscisas of the datapoints
complex(8),    intent(in) :: y(*)  ! data value of the datapoint
```

INPUT/OUTPUT PARAMETERS:

```
real(8),    intent(inout) :: chisq

real(8),    intent(inout) :: a(1:ma)
real(8),    intent(inout) :: alpha(1:nca,1:nca)
real(8),    intent(inout) :: beta(1:ma)
```

LOCAL VARIABLES:

```
      integer(4) :: i
      integer(4) :: j
      integer(4) :: k
      integer(4) :: l
      integer(4) :: m

      complex(8)    :: wt
      complex(8)    :: dy
      complex(8)    :: ymod
      complex(8)    :: cx
      complex(8)    :: dyda(1:ma)
      complex(8)    :: d2yda(1:ma,1:ma)


   !EXTERNAL ROUTINES:


       external ratfun
```

REVISION HISTORY:

```
   Original subroutine: mrqcof.for (c) copr. 1986-92 numerical recipes
   software &681i..
   Last modified: 7th. Jul. 2005 by RGA
```

2.0.250 setsac (Source File: setsac.f90)

INTERFACE:

```
      subroutine setsac(iopac,nomeg,npar,en,sc,omega,apar,poles)
```

DESCRIPTION:

This subroutine set up the parameters for the selfenery analytic continuation (SAC)

USES:

```
      implicit none
      integer(4),intent(in):: iopac         ! Option for which method to use
   0  -- Thiele's reciprocal difference method as described in
         H. J. Vidberg and J. W. Serence, J. Low Temp. Phys. 29, 179 (1977)
   1  -- Rojas, Godby and Needs (PRL 74, 1827 (1996)

      integer(4),intent(in):: nomeg         ! Number of frequency points along the imaginary a
      integer(4),intent(in):: npar          ! Number of parameters (== 2*npol (npol= the numbe
```

```
      real(8),intent(in):: en               ! LDA eigen-energy around which an analyic continu
      real(8),intent(in):: omega(nomeg)     ! frequency points along the imaginary frequency
      complex(8),intent(in) :: sc(nomeg)    ! correlation selfenergy along the imaginary frequ
      complex(8),intent(out):: apar(npar)   ! fitted paramters to calculate selfenergy
      complex(8),intent(out):: poles(npar)  ! the positions of poles of fitted selfenergy
```

LOCAL VARIABLES:

```
      integer(4) :: iw,step,ipar,ierr
      integer(4) :: iwpa(npar)
      real(8) :: varsq    ! Square root of the mean square error
      real(8)    :: xin(nomeg),anl(2*npar)      ! Values of the function
      complex(8) :: yin(nomeg)             ! Values of the function
      complex(8) :: cx(npar),cy(npar)        ! input for Pade's approximation
      complex(8) :: coefs(0:npar/2)
      logical::lpolish


  !INTRINSIC ROUTINES:

      intrinsic dble

  !EXTERNAL ROUTINES:
      external setrgn
      external setpatrd
```

REVISION HISTORY:

```
  Created: Nov. 19, 2007 by  JH
```

2.0.251 stdesc (Source File: stdesc.f90)

INTERFACE:

```
      subroutine stdesc(x,y,ndata,a,ma,varsq)
```

DESCRIPTION:

This program perform one iteration of the steepest descent method.

INPUT PARAMETERS:

```
      implicit none

      integer(4), intent(in) :: ndata ! Number of datapoints to be fitted
      integer(4), intent(in) :: ma    ! Number of coefficients
```

```
real(8),    intent(in) :: x(*)  ! abscisas of the datapoints
complex(8),    intent(in) :: y(*)  ! data value of the datapoint
```

INPUT/OUTPUT PARAMETERS:

```
real(8),    intent(inout) :: varsq

real(8),    intent(inout) :: a(1:ma)
```

LOCAL VARIABLES:

```
integer(4) :: j
integer(4) :: it
integer(4) :: i

real(8) :: varsqold
real(8) :: num
real(8) :: denomi
real(8) :: denom
real(8) :: lambda
real(8) :: deltach

real(8) :: atemp(1:ma)
real(8), allocatable :: beta(:)
real(8), allocatable :: alpha(:,:)

logical :: conv
```

DEFINED PARAMETERS:

```
real(8), parameter :: tolstd = 1.0d-3
```

```
!EXTERNAL ROUTINES:


external mrqcof
```

REVISION HISTORY:

```
Created: 11th. Jul. 2005 by RGA
```

2.0.252 acrgn (Source File: acrgn.f90)

INTERFACE:

```
subroutine acrgn(npar,x,ca,y,dy)
```

DESCRIPTION:

Given the abscissa `x`, and the parameters `a(1:2n+2)` this subroutine returns in `y` the value of the function:

$$f(x,\{c\}) = \frac{P(x,\{c\})}{Q(x,\{c\})} \tag{44}$$

where

$$P(x,\{c\}) = \sum_{k=1}^{n+1} c_k x^{k-1}, \tag{45}$$

$$Q(x,\{c\}) = 1 + \sum_{k=n+2}^{2n+2} c_k x^{k-n-1} \tag{46}$$

and $c_k =$ `a(k)` $+ i$`a(k+2n+2)`. In `dy` the derivative of the function with respect to `x`:

$$\frac{\partial f(x,\{c\})}{\partial x} = \frac{1}{Q(x,\{c\})}\frac{\partial P(x,\{c\})}{\partial x} - \frac{P(x,\{c\})}{[Q(x,\{c\})]^2}\frac{\partial Q(x,\{c\})}{\partial x} \tag{47}$$

with:

$$\frac{\partial P(x,\{c\})}{\partial x} = \sum_{k=2}^{n+1} (k-1) c_k x^{k-2} \tag{48a}$$

$$\frac{\partial Q(x,\{c\})}{\partial x} = \sum_{k=n+2}^{2n+2} (k-n-1) c_k x^{k-n-2} \tag{48b}$$

$$\tag{48c}$$

INPUT PARAMETERS:

```
implicit none

integer(4), intent(in) :: npar        ! number of fitting parameters
complex(8), intent(in) :: x,ca(npar)    ! fitting coeffients
```

OUTPUT PARAMETERS:

```
complex(8),    intent(out) :: y       ! fitted selfec at real frequency
complex(8),    intent(out) :: dy      ! first order derivative of selfec with respect to
```

LOCAL VARIABLES:

```
integer(4) :: i
integer(4) :: n

complex(8) :: q
complex(8) :: p
complex(8) :: dq
complex(8) :: dp
```

DEFINED PARAMETERS:

```
complex(8), parameter :: czero = (0.0d0,0.0d0)
complex(8), parameter :: cone = (1.0d0,0.0d0)
complex(8), parameter :: cim = (0.0d0,1.0d0)
```

REVISION HISTORY:

Created: 13th. Jul. 2005 by RGA

2.0.253 nllsq (Source File: nllsq.f90)

INTERFACE:

```
subroutine nllsq(x,y,ndata,a,ma,chisq)
```

DESCRIPTION:

This subroutine fits the set for data points `x(1:ndata)`, `y(1:ndata)`, and a nonlinear function dependent on `ma` coefficients `a(1:ma)` using the Levenberg-Marquardt method.

INPUT PARAMETERS:

```
implicit none

integer(4), intent(in) :: ndata ! Number of datapoints to be fitted
integer(4), intent(in) :: ma    ! Number of coefficients
real(8),    intent(in) :: x(*)  ! abscisas of the datapoints
complex(8), intent(in) :: y(*)  ! data value of the datapoint
```

INPUT/OUTPUT PARAMETERS:

```
real(8),    intent(inout) :: a(1:ma)
```

OUTPUT PARAMETERS:

```
real(8),    intent(inout) :: chisq
```

LOCAL VARIABLES:

```
integer(4) :: it

real(8) :: alambda
real(8) :: alambdaold
real(8) :: chisqold
real(8) :: deltach
real(8) :: covar(1:ma,1:ma)
real(8) :: alpha(1:ma,1:ma)

logical :: converg
```

DEFINED PARAMETERS:

```
real(8), parameter :: tol = 1.0d-10
```

```
!EXTERNAL ROUTINES:


external mrqmin
external stdesc
```

REVISION HISTORY:

```
Created: 8th. Jul. 2005 by RGA
```

2.0.254 mrqmin (Source File: mrqmin.f90)

INTERFACE:

```
subroutine mrqmin(x,y,ndata,a,ma,covar,alpha,nca,chisq,alamda)
```

DESCRIPTION:

Levenberg-Marquardt method, attempting to reduce the value χ^2 of a fit between a set of data points `x(1:ndata)`, `y(1:ndata)`, and a nonlinear function dependent on `ma` coefficients `a(1:ma)`. The program returns current best-fit values for the parameters `a(1:ma)`, and $\chi^2 =$ `chisq`. The arrays `covar(1:nca,1:nca)`, `alpha(1:nca,1:nca)` with physical dimension `nca` ($\geq$ the number of fitted parameters) are used as working space during most iterations. On the first call provide an initial guess for the parameters `a`, and set `alamda` < 0 for initialization (which then sets `alamda` $= .001$). If a step succeeds `chisq` becomes smaller and `alamda` decreases by a factor of 10. If a step fails `alamda` grows by a factor of 10. You must call this routine repeatedly until convergence is achieved. Then, make one final call

with `alamda` = 0, so that `covar(1:ma,1:ma)` returns the covariance matrix, and `alpha` the curvature matrix. (Parameters held fixed will return zero covariances.)

INPUT PARAMETERS:

```
implicit none

integer(4), intent(in) :: ndata ! Number of datapoints to be fitted
integer(4), intent(in) :: ma    ! Number of coefficients
integer(4), intent(in) :: nca   ! Size of working-space arrays

real(8),    intent(in) :: x(*)  ! abscisas of the datapoints
complex(8),    intent(in) :: y(*)  ! data value of the datapoint
```

INPUT/OUTPUT PARAMETERS:

```
real(8),    intent(inout) :: alamda
real(8),    intent(inout) :: chisq

real(8),    intent(inout) :: a(1:ma)
real(8),    intent(inout) :: covar(1:nca,1:nca)
real(8),    intent(inout) :: alpha(1:nca,1:nca)
```

LOCAL VARIABLES:

```
integer(4) :: j
integer(4) :: k
integer(4) :: l
integer(4), parameter :: max = 50

real(8)    :: ochisq

real(8), dimension(1:max) :: atry
real(8), dimension(1:max) :: beta
real(8), dimension(1:max) :: da

save ochisq,atry,beta,da


!EXTERNAL ROUTINES:


external covsrt
external gaussj
external mrqcof
```

REVISION HISTORY:

```
Original subroutine: mrqmin.for (c) copr. 1986-92 numerical recipes
software &680i..
Last modified: 7th. Jul. 2005 by RGA
```

2.0.255 gaussj (Source File: gaussj.f90)

INTERFACE:

```
subroutine gaussj(a,n,np,b,m,mp)
```

DESCRIPTION:

Linear equation solution by Gauss-Jordan elimination, equation 49 below.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \left[\begin{pmatrix} x_{11} \\ x_{21} \\ x_{31} \\ x_{41} \end{pmatrix} \sqcup \begin{pmatrix} x_{12} \\ x_{22} \\ x_{32} \\ x_{42} \end{pmatrix} \sqcup \begin{pmatrix} x_{13} \\ x_{23} \\ x_{33} \\ x_{43} \end{pmatrix} \sqcup \begin{pmatrix} y_{11} & y_{12} & y_{13} & y_{14} \\ y_{21} & y_{22} & y_{23} & y_{24} \\ y_{31} & y_{32} & y_{33} & y_{34} \\ y_{41} & y_{42} & y_{43} & y_{44} \end{pmatrix} \right] = \left[\begin{pmatrix} b_{11} \\ b_{21} \\ b_{31} \\ b_{41} \end{pmatrix} \sqcup \begin{pmatrix} b_{12} \\ b_{22} \\ b_{32} \\ b_{42} \end{pmatrix} \sqcup \begin{pmatrix} b_{13} \\ b_{23} \\ b_{33} \\ b_{43} \end{pmatrix} \sqcup \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \right] \tag{49}$$

`a(1:n,1:n)` is an imput matrix stored in an array of physical dimensions `np` by `np`.`b(1:n,1:m)` is an input matrix containing the `m` right-hand side vectors, stored in an array of physical dimensions `np` by `mp`. On output, `a(1:n,1:n)` is replaced by its matrix inverse, and `b(1:n,1:m)` is replaced by the corresponding set of solution vectors.

INPUT PARAMETERS:

```
implicit none

integer(4), intent(in) :: m
integer(4), intent(in) :: mp
integer(4), intent(in) :: n
integer(4), intent(in) :: np
```

INPUT/OUTPUT PARAMETERS:

```
real(8), intent(inout) :: a(1:np,1:np)
real(8), intent(inout) :: b(1:np,1:mp)
```

LOCAL VARIABLES:

```
integer(4) :: i
integer(4) :: icol
integer(4) :: irow
integer(4) :: j
integer(4) :: k
integer(4) :: l
integer(4) :: ll
integer(4) :: indxc(1:np) ! Used for bookkeeping on the pivoting
integer(4) :: indxr(1:np) ! Used for bookkeeping on the pivoting
integer(4) :: ipiv(1:np)  ! Used for bookkeeping on the pivoting

real(8) :: big
real(8) :: dum
real(8) :: pivinv
```

REVISION HISTORY:

```
Original subroutine: gaussj.for (c) copr. 1986-92 numerical recipes
software &30i..
Last modified: 7th. Jul. 2005 by RGA
```