# The Elk Code Manual
# Version 0.9.279

J. K. Dewhurst, S. Sharma, L. Nordström, F. Cricchio, F. Bultmark

## Contents

# 1 Introduction

Welcome to the Elk Code! Elk is an all-electron full-potential linearised augmented-plane-wave (FP-LAPW) code for determining the properties of crystalline solids. It was developed originally at the Karl-Franzens-Universität Graz as part of the EXCITING EU Research and Training Network project [1]. The guiding philosophy during the implementation of the code was to keep it as simple as possible for both users and developers without compromising on its capabilities. All the routines are released under either the GNU General Public License (GPL) or the GNU Lesser General Public License (LGPL) in the hope that they may inspire other scientists to implement new developments in the field of density functional theory and beyond.

# 2 Acknowledgements

Lots of people contributed to the Elk code with ideas, checking and testing, writing code or documentation and general encouragement. They include Claudia Ambrosch-Draxl, Clas Persson, Christian Brouder, Rickard Armiento, Andrew Chizmeshya, Per Anderson, Igor Nekrasov, Sushil Auluck, Frank Wagner, Fateh Kalarasse, Jürgen Spitaler, Stefano Pittalis, Nektarios Lathiotakis, Tobias Burnus, Stephan Sagmeister, Christian Meisenbichler, Sébastien Lebègue, Yigang Zhang, Fritz Körmann, Alexey Baranov, Anton Kozhevnikov, Shigeru Suehara, Frank Essenberger, Antonio Sanna and Tyrel McQueen. Special mention of David Singh's very useful book *Planewaves, Pseudopotentials and the LAPW Method* [2] must also be made. Finally we would like to acknowledge the generous support of Karl-Franzens-Universität Graz, as well as the EU Marie-Curie Research Training Networks initiative.

Kay Dewhurst
Sangeeta Sharma
Lars Nordström
Francesco Cricchio
Fredrik Bultmark

Berlin and Uppsala, August 2009

# 3 Units

Unless explicitly stated otherwise, Elk uses atomic units. In this system $\hbar = 1$, the electron mass $m = 1$, the Bohr radius $a_0 = 1$ and the electron charge $e = 1$ (note that the electron charge is positive, so that the atomic numbers $Z$ are negative). Thus, the atomic unit of length is 0.52917720859(36) Å, and the atomic unit of energy is the Hartree which equals 27.21138386(68) eV. The unit of the external magnetic fields is defined such that one unit of magnetic field in `elk.in` equals 1717.2445320376 Tesla.

# 4 Compiling and running Elk

## 4.1 Compiling the code

Unpack the code from the archive file. Run the command

```
setup
```

in the `elk` directory and select the appropriate system and compiler. We highly recommend that you edit the file `make.inc` and tune the compiler options for your particular system. You can also make use of machine-optimised `BLAS/LAPACK` libraries if they are available, but make sure they are version 3.$x$. Setting the OpenMP options of your compiler will enable Elk to run in parallel mode on multiprocessor systems. Following this, run

```
make all
```

This will hopefully compile the entire code and all the libraries into one executable, `elk`, located in the `src` directory. It will also compile a few useful auxiliary programs, namely `spacegroup` for producing crystal geometries from spacegroup data, `species` for generating species files, and `eos` for fitting equations of state to energy-volume data. If you want to compile everything all over again, then run `make clean` from the `elk` directory, followed by `make all`.

## 4.2 Running the code

As a rule, all input files for the code are in lower case and end with the extension `.in`. All output files are uppercase and have the extension `.OUT`. For most cases, the user will only need to modify the file `elk.in`. In this file input parameters are arranged in blocks. Each block consists of a block name on one line and the block variables on subsequent lines. Almost all blocks are optional: the code uses reasonable default values in cases where they are absent. Blocks can appear in any order, if a block is repeated then the second instance is used. Comment lines can be included in the input file and begin with the `!` character.
The only other input files are those describing the atomic species which go into the crystal. These files are found in the `species` directory and are named with the element symbol and the extension `.in`, for example `Sb.in`. They contain parameters like the atomic charge, mass, muffin-tin radius, occupied atomic states and the type of linearisation required. Users should not have to modify these files in the majority of cases.
The best way to learn to use Elk is to run the examples included with the package. These can be found in the `examples` directory and use many of the code's capabilities. The following section which describes all the input parameters will be of invaluable assistance.

# 5 Input blocks

This section lists all the input blocks available. It is arranged with the name of the block followed by a table which lists each parameter name, what the parameter does, its type and default value. A horizontal line in the table indicates a new line in `elk.in`. Below the table is a brief overview of the block's function.

## 5.1 `atoms`

| | | | |
|---|---|---|---|
| `nspecies` | number of species | integer | 0 |
| `spfname(i)` | species filename for species `i` | string | - |
| `natoms(i)` | number of atoms for species `i` | integer | - |
| `atposl(j,i)` | atomic position in lattice coordinates for atom `j` | real(3) | - |
| `bfcmt(j,i)` | muffin-tin external magnetic field in Cartesian coordinates for atom `j` | real(3) | - |

Defines the atomic species as well as their positions in the unit cell and the external magnetic field applied throughout the muffin-tin. These fields are used to break spin symmetry and should be considered infinitesimal as they do not contribute directly to the total energy. Collinear calculations are more efficient if the field is applied in the $z$-direction. One could, for example, set up an anti-ferromagnetic crystal by pointing the field on one atom in the positive $z$-direction and in the opposite direction on another atom. If `molecule` is `.true.` then the atomic positions are assumed to be in Cartesian coordinates. See also `sppath`, `bfieldc` and `molecule`.

## 5.2 `autokpt`

| | | | |
|---|---|---|---|
| `autokpt` | `.true.` if the **k**-point set is to be determined automatically | logical | `.false.` |

See `radkpt` for details.

## 5.3 `autormt`

| | | | |
|---|---|---|---|
| `autormt` | `.true.` if muffin-tin radii should be determined automatically | logical | `.false.` |

See `rmtapm` for details.

## 5.4 `avec`

| | | | |
|---|---|---|---|
| `avec(1)` | first lattice vector | real(3) | $(1.0, 0.0, 0.0)$ |
| `avec(2)` | second lattice vector | real(3) | $(0.0, 1.0, 0.0)$ |
| `avec(3)` | third lattice vector | real(3) | $(0.0, 0.0, 1.0)$ |

Lattice vectors of the crystal in atomic units (Bohr). If `molecule` is `.true.` then these vectors are not used.

## 5.5 `beta0`

| | | | |
|---|---|---|---|
| `beta0` | adaptive mixing parameter | real | 0.05 |

This determines how much of the potential from the previous iteration is mixed with the current potential during the self-consistent loop. It should be made smaller if the calculation is unstable. See `betamax` and also the routine `mixadapt`.

### 5.6 `betamax`

| | | | |
|---|---|---|---|
| `betamax` | maximum adaptive mixing parameter | real | 0.5 |

Maximum allowed mixing parameter used in routine `mixadapt`.

### 5.7 `bfieldc`

| | | | |
|---|---|---|---|
| `bfieldc` | global external magnetic field in Cartesian coordinates | real(3) | $(0.0, 0.0, 0.0)$ |

This is a constant magnetic field applied throughout the entire unit cell and enters the second-variational Hamiltonian as

$$\frac{g_e}{4c}\,\vec{\sigma}\cdot\mathbf{B}_{\text{ext}},$$

where $g_e$ is the electron $g$-factor. This field is normally used to break spin symmetry for spin-polarised calculations and considered to be infinitesimal with no direct contribution to the total energy. In cases where the magnetic field is finite (for example when computing magnetic response) the external $\mathbf{B}$-field energy reported in `INFO.OUT` should be added to the total by hand. This field is applied throughout the entire unit cell. To apply magnetic fields in particular muffin-tins use the `bfcmt` vectors in the `atoms` block. Collinear calculations are more efficient if the field is applied in the $z$-direction.

### 5.8 `chgexs`

| | | | |
|---|---|---|---|
| `chgexs` | excess electronic charge | real | 0.0 |

This controls the amount of charge in the unit cell beyond that required to maintain neutrality. It can be set positive or negative depending on whether electron or hole doping is required.

### 5.9 `deband`

| | | | |
|---|---|---|---|
| `deband` | initial band energy step size | real | 0.0025 |

The initial step length used when searching for the band energy, which is used as the APW and local-orbital linearisation energies. 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 $E_{\text{t}}$. A downward search is now performed from $E_{\text{t}}$ until the slope of the radial wavefunction at the muffin-tin radius is zero. This energy, $E_{\text{b}}$, is at the bottom of the band. The band energy is taken as $(E_{\text{t}}+E_{\text{b}})/2$. If either $E_{\text{t}}$ or $E_{\text{b}}$ cannot be found then the band energy is set to the default value.

### 5.10 `deltaem`

| | | | |
|---|---|---|---|
| `deltaem` | the size of the **k**-vector displacement used when calculating numerical derivatives for the effective mass tensor | real | 0.025 |

See `ndspem` and `vklem`.

### 5.11 `deltaph`

| | | | |
|---|---|---|---|
| `deltaph` | the size of the atomic displacement used for calculating dynamical matrices | real | 0.03 |

Phonon calculations are performed by constructing a supercell corresponding to a particular **q**-vector and making a small periodic displacement of the atoms. The magnitude of this displacement is given by `deltaph`. This should not be made too large, as anharmonic terms could then become significant, neither should it be too small as this can introduce numerical error.

### 5.12 `dos`

| | | | |
|---|---|---|---|
| `nwdos` | number of frequency/energy points in the DOS or optics plot | integer | 500 |
| `ngrdos` | effective **k**-point mesh size to be used for Brillouin zone integration | integer | 100 |
| `nsmdos` | level of smoothing applied to DOS/optics output | integer | 0 |
| `wintdos` | frequency/energy window for the DOS or optics plot | real(2) | $(-0.5, 0.5)$ |

DOS and optics plots require integrals of the kind

$$g(\omega_i) = \frac{\Omega}{(2\pi)^3} \int_{\mathrm{BZ}} f(\mathbf{k}) \delta(\omega_i - e(\mathbf{k})) d\mathbf{k}.$$

These are calculated by first interpolating the functions $e(\mathbf{k})$ and $f(\mathbf{k})$ with the trilinear method on a much finer mesh whose size is determined by `ngrdos`. Then the $\omega$-dependent histogram of the integrand is accumulated over the fine mesh. If the output function is noisy then either `ngrdos` should be increased or `nwdos` decreased. Alternatively, the output function can be artificially smoothed up to a level given by `nsmdos`. This is the number of successive 3-point averages to be applied to the function $g$.

### 5.13 `epsband`

| | | | |
|---|---|---|---|
| `epsband` | convergence tolerance for determining band energies | real | $1 \times 10^{-6}$ |

APW and local-orbital linearisation energies are determined from the band energies. Good convergence of these energies, especially for low-lying, states is required for accurate total energies. See also `deband`.

### 5.14 `epschg`

| | | | |
|---|---|---|---|
| `epschg` | maximum allowed error in the calculated total charge beyond which a warning message will be issued | real | $1 \times 10^{-3}$ |

### 5.15 `epsforce`

| | | | |
|---|---|---|---|
| `epsforce` | convergence tolerance for the forces during a structural optimisation run | real | $5 \times 10^{-4}$ |

If the mean absolute value of the atomic forces is less than `epsforce` then the structural optimisation run is ended. See `tasks`.

### 5.16 `epslat`

| | | | |
|---|---|---|---|
| `epslat` | vectors with lengths less than this are considered zero | real | $10^{-6}$ |

Sets the tolerance for determining if a vector or its components are zero. This is to account for any numerical error in real or reciprocal space vectors.

### 5.17 `epsocc`

| | | | |
|---|---|---|---|
| `epsocc` | smallest occupancy for which a state will contribute to the density | real | $1 \times 10^{-8}$ |

### 5.18 `epspot`

| | | | |
|---|---|---|---|
| `epspot` | convergence criterion for the effective potential and field | real | $1 \times 10^{-6}$ |

If the RMS change in the effective potential and magnetic field is smaller than `epspot`, then the self-consistent loop is considered converged and exited. For structural optimisation runs this results in the forces being calculated, the atomic positions updated and the loop restarted. See also `maxscl`.

### 5.19 `evalmin`

| | | | |
|---|---|---|---|
| `evalmin` | valence eigenvalue minimum | real | $-4.5$ |

Any valence states with eigenvalues below `evalmin` are not occupied and a warning message is issued.

### 5.20 `emaxelnes`

| | | | |
|---|---|---|---|
| `emaxelnes` | maximum allowed initial-state eigenvalue for ELNES calculations | real | $-1.2$ |

### 5.21 `fixspin`

| | | | |
|---|---|---|---|
| `fixspin` | 0 for no fixed spin moment (FSM), 1 for total FSM, 2 for local muffin-tin FSM, and 3 for both total and local FSM | integer | 0 |

Set to 1, 2 or 3 for fixed spin moment calculations. To fix only the direction and not the magnitude set to $-1$, $-2$ or $-3$. See also `momfix`, `mommtfix`, `taufsm` and `spinpol`.

### 5.22 fracinr

| | | | |
|---|---|---|---|
| `fracinr` | fraction of the muffin-tin radius up to which `lmaxinr` is used as the angular momentum cut-off | real | 0.25 |

See `lmaxinr`.

### 5.23 gmaxvr

| | | | |
|---|---|---|---|
| `gmaxvr` | maximum length of $\|\mathbf{G}\|$ for expanding the interstitial density and potential | real | 12.0 |

See also `rgkmax`.

### 5.24 intraband

| | | | |
|---|---|---|---|
| `intraband` | `.true.` if the intraband (Drude-like) contribution is to be added to the dieletric tensor | logical | `.false.` |

### 5.25 isgkmax

| | | | |
|---|---|---|---|
| `isgkmax` | species for which the muffin-tin radius will be used for calculating `gkmax` | integer | $-1$ |

By default the smallest muffin-tin radius is used for determining `gkmax` from `rgkmax`. This can be changed by setting `isgkmax` to the desired species number.

### 5.26 kstlist

| | | | |
|---|---|---|---|
| `kstlist(i)` | $i$th **k**-point and state pair | integer(2) | $(1, 1)$ |

This is a user-defined list of **k**-point and state index pairs which are those used for plotting wavefunctions and writing **L**, **S** and **J** expectation values. Only the first pair is used by the aforementioned tasks. The list should be terminated by a blank line.

### 5.27 lda+u

| | | | |
|---|---|---|---|
| `ldapu` | type of LDA+$U$ calculation | integer | 0 |
| `inptypelu` | type of input for LDA+U calculation | integer | 1 |
| `is` | species number | integer | - |
| `l` | angular momentum value | integer | -1 |
| `u` | the desired $U$ value | real | 0.0 |
| `j` | the desired $J$ value | real | 0.0 |

This block contains the parameters required for an LDA+$U$ calculation, with the list of parameters for each species terminated with a blank line. The type of double counting required is set with the parameter `ldapu`. Currently implemented are:

0 No LDA+$U$ calculation
1 Fully localised limit (FLL)
2 Around mean field (AFM)
3 An interpolation between FLL and AFM

The type of input parameters is set with the parameter `inptypelu`. The current possibilities are:

1 U and J
2 Slater parameters
3 Racah parameters
4 Yukawa screening length
5 U and determination of corresponding Yukawa screening length

See (amongst others) *Phys. Rev. B* **67**, 153106 (2003), *Phys. Rev. B* **52**, R5467 (1995), *Phys. Rev. B* **60**, 10673 (1999), and *Phys. Rev. B* **80**, 035121 (2009).

### 5.28 `lmaxapw`

| | | | |
|---|---|---|---|
| `lmaxapw` | angular momentum cut-off for the APW functions | integer | 8 |

### 5.29 `lmaxinr`

| | | | |
|---|---|---|---|
| `lmaxinr` | angular momentum cut-off for the muffin-tin density and potential on the inner part of the muffin-tin | integer | 2 |

Close to the nucleus, the density and potential is almost spherical and therefore the spherical harmonic expansion can be truncated a low angular momentum. See also `fracinr`.

### 5.30 `lmaxmat`

| | | | |
|---|---|---|---|
| `lmaxmat` | angular momentum cut-off for the outer-most loop in the hamiltonian and overlap matrix setup | integer | 5 |

### 5.31 `lmaxvr`

| | | | |
|---|---|---|---|
| `lmaxvr` | angular momentum cut-off for the muffin-tin density and potential | integer | 7 |

### 5.32 `lmirep`

| | | | |
|---|---|---|---|
| `lmirep` | `.true.` if the $Y_{lm}$ basis is to be transformed into the basis of irreducible representations of the site symmetries for DOS plotting | logical | `.false.` |

When lmirep is set to .true., the spherical harmonic basis is transformed into one in which the site symmetries are block diagonal. Band characters determined from the density matrix expressed in this basis correspond to irreducible representations, and allow the partial DOS to be resolved into physically relevant contributions, for example $e_g$ and $t_{2g}$.

### 5.33 `lradstp`

| | | | |
|---|---|---|---|
| `lradstp` | radial step length for determining coarse radial mesh | integer | 4 |

Some muffin-tin functions (such as the density) are calculated on a coarse radial mesh and then interpolated onto a fine mesh. This is done for the sake of efficiency. `lradstp` defines the step size in going from the fine to the coarse radial mesh. If it is too large, loss of precision may occur.

### 5.34 maxitoep

| | | | |
|---|---|---|---|
| maxitoep | maximum number of iterations when solving the exact exchange integral equations | integer | 120 |

See tau0oep.

### 5.35 maxscl

| | | | |
|---|---|---|---|
| maxscl | maximum number of self-consistent loops allowed | integer | 200 |

This determines after how many loops the self-consistent cycle will terminate if the convergence criterion is not met. If maxscl is 1 then the density and potential file, STATE.OUT, will **not** be written to disk at the end of the loop. See epspot.

### 5.36 mixtype

| | | | |
|---|---|---|---|
| mixtype | type of mixing required for the potential | integer | 1 |

1 Adaptive linear mixing
2 Pulay mixing, *Chem. Phys. Lett.* **73**, 393 (1980)

### 5.37 molecule

| | | | |
|---|---|---|---|
| molecule | .true. if the system is an isolated molecule | logical | .false. |

If molecule is .true., then the atomic positions, **a**, given in the atoms block are assumed to be in Cartesian coordinates. The lattice vectors are also set up automatically with the $i$th lattice vector given by

$$\mathbf{A}^i = A_i \hat{\mathbf{e}}^i,$$

where

$$A_i = \max_{\alpha,\beta} \left| \mathbf{a}_i^\alpha - \mathbf{a}_i^\beta \right| + d_{\rm vac}$$

with $\alpha$ and $\beta$ labeling atoms, and $d_{\rm vac}$ determines the size of the vacuum around the molecule. The last variable is set by the input parameter vacuum.

### 5.38 momfix

| | | | |
|---|---|---|---|
| momfix | the desired total moment for a FSM calculation | real(3) | (0.0, 0.0, 0.0) |

Note that all three components must be specified (even for collinear calculations). See fixspin, taufsm and spinpol.

### 5.39 mommtfix

| | | | |
|---|---|---|---|
| is | species number | integer | 0 |
| ia | atom number | integer | 0 |
| mommtfix | the desired muffin-tin moment for a FSM calculation | real(3) | (0.0, 0.0, 0.0) |

The local muffin-tin moments are specified for a subset of atoms, with the list terminated

with a blank line. Note that all three components must be specified (even for collinear calculations). See `fixspin`, `taufsm` and `spinpol`.

### 5.40 `mustar`

| | | | |
|---|---|---|---|
| `mustar` | Coulomb pseudopotential, $\mu^*$, used in the McMillan-Allen-Dynes equation | real | 0.15 |

This is used when calculating the superconducting critical temperature with the formula [Phys. Rev. B 12, 905 (1975)]

$$T_c = \frac{\omega_{\log}}{1.2k_B} \exp\left[\frac{-1.04(1+\lambda)}{\lambda - \mu^*(1+0.62\lambda)}\right],$$

where $\omega_{\log}$ is the logarithmic average frequency and $\lambda$ is the electron-phonon coupling constant.

### 5.41 `ndspem`

| | | | |
|---|---|---|---|
| `ndspem` | the number of **k**-vector displacements in each direction around `vklem` when computing the numerical derivatives for the effective mass tensor | integer | 1 |

See `deltaem` and `vklem`.

### 5.42 `nempty`

| | | | |
|---|---|---|---|
| `nempty` | the number of empty states | integer | 5 |

Defines the number of eigenstates beyond that required for charge neutrality. When running metals it is not known *a priori* how many states will be below the Fermi energy for each **k**-point. Setting `nempty` greater than zero allows the additional states to act as a buffer in such cases. Furthermore, magnetic calculations use the first-variational eigenstates as a basis for setting up the second-variational Hamiltonian, and thus `nempty` will determine the size of this basis set. Convergence with respect to this quantity should be checked.

### 5.43 `ngridk`

| | | | |
|---|---|---|---|
| `ngridk` | the **k**-point mesh sizes | integer(3) | (1, 1, 1) |

The **k**-vectors are generated using

$$\mathbf{k} = (\frac{i_1}{n_1}, \frac{i_2}{n_2}, \frac{i_3}{n_3}) + \mathbf{v}_{\text{off}},$$

where $i_j$ runs from 0 to $n_j - 1$ and $0 \leq \mathbf{v}_{\text{off};j} < 1$ for $j = 1, 2, 3$. See also `reducek` and `vkloff`.

### 5.44 `ngridq`

| | | | |
|---|---|---|---|
| `ngridq` | the phonon **q**-point mesh sizes | integer(3) | (1, 1, 1) |

Same as `ngridk`, except that this mesh is for the phonon **q**-points. See also `reduceq`.

### 5.45 nosource

| | | | |
|---|---|---|---|
| `nosource` | when set to `.true.`, source fields are projected out of the exchange-correlation magnetic field | logical | `.false.` |

Experimental feature.

### 5.46 nosym

| | | | |
|---|---|---|---|
| `nosym` | when set to `.true.` no symmetries, apart from the identity, are used anywhere in the code | logical | `.false.` |

### 5.47 notes

| | | | |
|---|---|---|---|
| `notes(i)` | the $i$th line of the notes | string | - |

This block allows users to add their own notes to the file `INFO.OUT`. The block should be terminated with a blank line, and no line should exceed 80 characters.

### 5.48 nprad

| | | | |
|---|---|---|---|
| `nprad` | radial polynomial order | integer | 4 |

This sets the polynomial order for the predictor-corrector method when solving the radial Dirac and Schrödinger equations, as well as for performing radial interpolation in the plotting routines.

### 5.49 nseqit

| | | | |
|---|---|---|---|
| `nseqit` | number of iterations per self-consistent loop using the iterative first-variational secular equation solver | integer | 6 |

See `tseqit`.

### 5.50 nstfsp

| | | | |
|---|---|---|---|
| `nstfsp` | number of states to be included in the Fermi surface plot file | integer | 6 |

### 5.51 nwrite

| | | | |
|---|---|---|---|
| `nwrite` | number of iterations after which `STATE.OUT` is to be written | integer | 0 |

Normally, the density and potentials are written to the file `STATE.OUT` only after completion of the self-consistent loop. By setting `nwrite` to a positive integer the file will be written during the loop every `nwrite` iterations.

### 5.52 optcomp

| | | | |
|---|---|---|---|
| `optcomp` | the components of the first- or second-order optical tensor to be calculated | integer(3) | $(1,1,1)$ |

This selects which components of the optical tensor you would like to plot. Only the first two are used for the first-order tensor.

### 5.53 phwrite

| | | | |
|---|---|---|---|
| nphwrt | number of **q**-points for which phonon modes are to be found | integer | 1 |
| vqlwrt(i) | the $i$th **q**-point in lattice coordinates | real(3) | $(0.0, 0.0, 0.0)$ |

This is used in conjunction with task=230. The code will write the phonon frequencies and eigenvectors to the file PHONON.OUT for all the **q**-points in the list. The **q**-points can be anywhere in the Brillouin zone and do not have to lie on the mesh defined by ngridq. Obviously, all the dynamical matrices have to be computed first using task=200.

### 5.54 plot1d

| | | | |
|---|---|---|---|
| nvp1d | number of vertices | integer | 2 |
| npp1d | number of plotting points | integer | 200 |
| vvlp1d(i) | lattice coordinates for vertex i | real(3) | $(0.0, 0.0, 0.0) \rightarrow (1.0, 1.0, 1.0)$ |

Defines the path in either real or reciprocal space along which the 1D plot is to be produced. The user should provide nvp1d vertices in lattice coordinates.

### 5.55 plot2d

| | | | |
|---|---|---|---|
| vclp2d(1) | first corner (origin) | real(3) | $(0.0, 0.0, 0.0)$ |
| vclp2d(2) | second corner | real(3) | $(1.0, 0.0, 0.0)$ |
| vclp2d(3) | third corner | real(3) | $(0.0, 1.0, 0.0)$ |
| np2d | number of plotting points in both directions | integer(2) | $(40, 40)$ |

Defines the corners of a parallelogram and the grid size used for producing 2D plots.

### 5.56 plot3d

| | | | |
|---|---|---|---|
| vclp3d(1) | first corner (origin) | real(3) | $(0.0, 0.0, 0.0)$ |
| vclp3d(2) | second corner | real(3) | $(1.0, 0.0, 0.0)$ |
| vclp3d(3) | third corner | real(3) | $(0.0, 1.0, 0.0)$ |
| vclp3d(4) | fourth corner | real(3) | $(0.0, 0.0, 1.0)$ |
| np3d | number of plotting points each direction | integer(3) | $(20, 20, 20)$ |

Defines the corners of a box and the grid size used for producing 3D plots.

### 5.57 primcell

| | | | |
|---|---|---|---|
| primcell | .true. if the primitive unit cell should be found | logical | .false. |

Allows the primitive unit cell to be determined automatically from the conventional cell. This is done by searching for lattice vectors among all those which connect atomic sites, and using the three shortest which produce a unit cell with non-zero volume.

### 5.58 radkpt

| | | | |
|---|---|---|---|
| radkpt | radius of sphere used to determine k-point density | real | 40.0 |

Used for the automatic determination of the **k**-point mesh. If `autokpt` is set to `.true.` then the mesh sizes will be determined by $n_i = \lambda/|\mathbf{A}_i| + 1$.

### 5.59 `readalu`

| | | | |
|---|---|---|---|
| `readalu` | set to `.true.` if the interpolation constant for LDA+$U$ should be read from file rather than calculated | logical | `.false.` |

When `ldapu`=3, the LDA+$U$ energy and potential are interpolated between FLL and AFM. The interpolation constant, $\alpha$, is normally calculated from the density matrix, but can also be read in from the file `ALPHALU.OUT`. This allows the user to fix $\alpha$, but is also necessary when calculating forces, since the contribution of the potential of the variation of $\alpha$ with respect to the density matrix is not computed. See `lda+u`.

### 5.60 `reducebf`

| | | | |
|---|---|---|---|
| `reducebf` | reduction factor for the external magnetic fields | real | 1.0 |

After each iteration the external magnetic fields are multiplied with `reducebf`. This allows for a large external magnetic field at the start of the self-consistent loop to break spin symmetry, while at the end of the loop the field will be effectively zero, i.e. infinitesimal. See `bfieldc` and `atoms`.

### 5.61 `reducek`

| | | | |
|---|---|---|---|
| `reducek` | set to `.true.` if the **k**-point set is to be reduced with the crystal symmetries | logical | `.true.` |

See also `ngridk` and `vkloff`.

### 5.62 `reduceq`

| | | | |
|---|---|---|---|
| `reduceq` | set to `.true.` if the **q**-point set is to be reduced with the crystal symmetries | logical | `.true.` |

See also `ngridq`.

### 5.63 `rgkmax`

| | | | |
|---|---|---|---|
| `rgkmax` | $R_{\mathrm{min}}^{\mathrm{MT}} \times \max(\|\mathbf{G}+\mathbf{k}\|)$ | real | 7.0 |

This sets the maximum length for the $\mathbf{G}+\mathbf{k}$ vectors, defined as `rgkmax` divided by the smallest muffin-tin radius.

### 5.64 `rmtapm`

| | | | |
|---|---|---|---|
| `rmtapm` | parameters governing the automatic generation of the muffin-tin radii | real(2) | $(0.25, 0.95)$ |

When `autormt` is set to true, the muffin-tin radii are found automatically from the formula

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

where $Z_i$ is the atomic number of the $i$th species, $\zeta$ 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.

### 5.65 scale

| | | | |
|---|---|---|---|
| `scale` | lattice vector scaling factor | real | 1.0 |

Scaling factor for all three lattice vectors. Applied in conjunction with `scale1`, `scale2` and `scale3`.

### 5.66 scale1/2/3

| | | | |
|---|---|---|---|
| `scale1/2/3` | separate scaling factors for each lattice vector | real | 1.0 |

### 5.67 scissor

| | | | |
|---|---|---|---|
| `scissor` | the scissors correction | real | 0.0 |

This is the scissors shift applied to states above the Fermi energy. Affects DOS, optics and band structure plots.

### 5.68 scrpath

| | | | |
|---|---|---|---|
| `scrpath` | scratch space path | string | `./` |

This is the path to scratch space where the eigenvector file `EIGVEC.OUT` will be written. If the local directory is accessed via a network then `scrpath` can be set to a directory on the local disk, for example `/tmp/`. Note that the forward slash `/` at the end of the string must be included.

### 5.69 spinorb

| | | | |
|---|---|---|---|
| `spinorb` | set to `.true.` if a spin-orbit coupling is required | logical | `.false.` |

If `spinorb` is `.true.`, then a $\boldsymbol{\sigma}\cdot\mathbf{L}$ term is added to the second-variational Hamiltonian. See `spinpol`.

### 5.70 spinpol

| | | | |
|---|---|---|---|
| `spinpol` | set to `.true.` if a spin-polarised calculation is required | logical | `.false.` |

If `spinpol` is `.true.`, then the spin-polarised Hamiltonian is solved as a second-variational step using two-component spinors in the effective magnetic field. The first variational scalar wavefunctions are used as a basis for setting this Hamiltonian.

### 5.71 spinsprl

| | | | |
|---|---|---|---|
| `spinsprl` | set to `.true.` if a spin-spiral calculation is required | logical | `.false.` |

Experimental feature for the calculation of spin-spiral states. See `vqlss` for details.

### 5.72 `sppath`

| | | | |
|---|---|---|---|
| `sppath` | path where the species files can be found | string | `./` |

Note that the forward slash `/` at the end of the string must be included.

### 5.73 `stype`

| | | | |
|---|---|---|---|
| `stype` | integer defining the type of smearing to be used | integer | 0 |

A smooth approximation to the Dirac delta function is needed to compute the occupancies of the Kohn-Sham states. The variable `swidth` determines the width of the approximate delta function. Currently implemented are

0 Gaussian
1 Methfessel-Paxton order 1, Phys. Rev. B **40**, 3616 (1989)
2 Methfessel-Paxton order 2
3 Fermi-Dirac

### 5.74 `swidth`

| | | | |
|---|---|---|---|
| `swidth` | width of the smooth approximation to the Dirac delta function | real | 0.01 |

See `stype` for details.

### 5.75 `tasks`

| | | | |
|---|---|---|---|
| `task(i)` | the $i$th task | integer | $-1$ |

A list of tasks for the code to perform sequentially. The list should be terminated with a blank line. Each task has an associated integer as follows:

| | |
|---|---|
| -1 | Write out the version number of the code. |
| 0 | Ground state run starting from the atomic densities. |
| 1 | Resumption of ground state run using density in `STATE.OUT`. |
| 2 | Structural optimisation run starting from the atomic densities, with atomic positions written to `GEOMETRY.OUT`. |
| 3 | Resumption of structural optimisation run using density in `STATE.OUT` but with positions from `elk.in`. |
| 5 | Ground state Hartree-Fock run. |
| 10 | Total, partial and interstitial density of states (DOS). |
| 14 | Plots the smooth Dirac delta and Heaviside step functions used by the code to calculate occupancies. |
| 15 | Output **L**, **S** and **J** total expectation values. |
| 16 | Output **L**, **S** and **J** expectation values for each **k**-point and state in `kstlist`. |
| 20 | Band structure plot. |
| 21 | Band structure plot which includes angular momentum characters for every atom. |
| 25 | Compute the effective mass tensor at the **k**-point given by `vklem`. |
| 31, 32, 33 | 1/2/3D charge density plot. |
| 41, 42, 43 | 1/2/3D exchange-correlation and Coulomb potential plots. |
| 51, 52, 53 | 1/2/3D electron localisation function (ELF) plot. |
| 61, 62, 63 | 1/2/3D wavefunction plot: $\lvert\Psi_{i\mathbf{k}}(\mathbf{r})\rvert^2$. |
| 72, 73 | 2/3D plot of magnetisation vector field, $\mathbf{m}(\mathbf{r})$. |
| 82, 83 | 2/3D plot of exchange-correlation magnetic vector field, $\mathbf{B}_{\rm xc}(\mathbf{r})$. |
| 91, 92, 93 | 1/2/3D plot of $\nabla\cdot\mathbf{B}_{\rm xc}(\mathbf{r})$. |
| 100 | 3D Fermi surface plot using the scalar product $p(\mathbf{k})=\Pi_i(\epsilon_{i\mathbf{k}}-\epsilon_{\rm F})$. |
| 101 | 3D Fermi surface plot using separate bands (minus the Fermi energy). |
| 110 | Calculation of Mössbauer contact charge densities and magnetic fields at the nuclear sites. |
| 115 | Calculation of the electric field gradient (EFG) at the nuclear sites. |
| 120 | Output of the momentum matrix elements $\langle\Psi_{i\mathbf{k}}\lvert-i\nabla\rvert\Psi_{j\mathbf{k}}\rangle$. |
| 121 | Linear optical response tensor. |
| 122 | Magneto optical Kerr effect (MOKE) angle. |
| 130 | Output matrix elements of the type $\langle\Psi_{i\mathbf{k}+\mathbf{q}}\rvert\exp[i(\mathbf{G}+\mathbf{q})\cdot\mathbf{r}]\rvert\Psi_{j\mathbf{k}}\rangle$. |
| 140 | Energy loss near edge structure (ELNES). |
| 142, 143 | 2/3D plot of the electric field $\mathbf{E}(\mathbf{r})\equiv\nabla V_{\rm C}(\mathbf{r})$. |
| 152, 153 | 2/3D plot of $\mathbf{m}(\mathbf{r})\times\mathbf{B}_{\rm xc}(\mathbf{r})$. |
| 162 | Scanning-tunneling microscopy (STM) image. |
| 190 | Write the atomic geometry to file for plotting with XCrySDen and V_Sim. |
| 200 | Calculation of dynamical matrices on a **q**-point set defined by `ngridq`. |
| 210 | Phonon density of states. |
| 220 | Phonon dispersion plot. |
| 230 | Phonon frequencies and eigenvectors for an arbitrary **q**-point. |
| 240 | Generate the **q**-dependent phonon linewidths and electron-phonon coupling constants and write them to file. |
| 245 | Phonon linewidths plot. |
| 250 | Eliashberg function $\alpha^2F(\omega)$, electron-phonon coupling constant $\lambda$, and the McMillan-Allen-Dynes critical temperature $T_c$. |
| 300 | Reduced density matrix functional theory (RDMFT) calculation. |
| 400 | Calculation of tensor moments and corresponding LDA+U Hartree-Fock energy contributions. |

### 5.76 `tau0atm`

| | | | |
|---|---|---|---|
| `tau0atm` | the step size to be used for structural optimisation | real | 0.2 |

The position of atom $\alpha$ is updated on step $m$ of a structural optimisation run using

$$\mathbf{r}_\alpha^{m+1} = \mathbf{r}_\alpha^m + \tau_\alpha^m \left( \mathbf{F}_\alpha^m + \mathbf{F}_\alpha^{m-1} \right),$$

where $\tau_\alpha$ is set to `tau0atm` for $m = 0$, and incremented by the same amount if the atom is moving in the same direction between steps. If the direction changes then $\tau_\alpha$ is reset to `tau0atm`.

### 5.77 `tauoep`

| | | | |
|---|---|---|---|
| `tauoep` | step length initial value and scaling factors for the OEP iterative solver | real(3) | $(1.0, 0.2, 1.5)$ |

The optimised effective potential is determined using an interative method. [Phys. Rev. Lett. 98, 196405 (2007)]. At the first iteration the step length is set to `tauoep(1)`. During subsequent iterations, the step length is scaled by `tauoep(2)` or `tauoep(3)`, when the residual is increasing or decreasing, respectively. See also `maxitoep`.

### 5.78 `taufsm`

| | | | |
|---|---|---|---|
| `taufsm` | the step size to be used when finding the effective magnetic field in fixed spin moment calculations | real | 0.01 |

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

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

where $\lambda$ is proportional to `taufsm`. See also `fixspin`, `momfix` and `spinpol`.

### 5.79 `tfibs`

| | | | |
|---|---|---|---|
| `tfibs` | set to `.true.` if the IBS correction to the force should be calculated | logical | `.true.` |

Because calculation of the incomplete basis set (IBS) correction to the force is fairly time-consuming, it can be switched off by setting `tfibs` to `.false.` This correction can then be included only when necessary, i.e. when the atoms are close to equilibrium in a structural relaxation run.

### 5.80 `tforce`

| | | | |
|---|---|---|---|
| `tforce` | set to `.true.` if the force should be calculated at the end of the self-consistent cycle | logical | `.false.` |

This variable is automatically set to `.true.` when performing structural optimisation.

### 5.81 `tmomlu`

| | | | |
|---|---|---|---|
| `tmomlu` | set to `.true.` if the tensor moments and the corresponding decomposition of LDA+U energy should be calculated at every iteration of the self-consistent cycle | logical | `.false.` |

This variable is useful to check the convergence of the tensor moments in LDA+U caculations. Alternatively, with `task` equal to 400, one can calculate the tensor moments and corresponding LDA+U energy contributions from a given density matrix and set of Slater parameters at the end of the self-consistent cycle.

### 5.82 `tseqit`

| | | | |
|---|---|---|---|
| `tseqit` | set to `.true.` if the first-variational secular equation should be solved iteratively | logical | `.false.` |

See also `nseqit`.

### 5.83 `tshift`

| | | | |
|---|---|---|---|
| `tshift` | set to `.true.` if the crystal can be shifted so that the atom closest to the origin is exactly at the origin | logical | `.true.` |

### 5.84 `vacuum`

| | | | |
|---|---|---|---|
| `vacuum` | the size of the vacuum region around a molecule | real | 8.05 |

See `molecule`.

### 5.85 `vklem`

| | | | |
|---|---|---|---|
| `vklem` | the **k**-point in lattice coordinates at which to compute the effective mass tensors | real(3) | $(0.0, 0.0, 0.0)$ |

See `deltaem` and `ndspem`.

### 5.86 `vqlss`

| | | | |
|---|---|---|---|
| `vqlss` | the **q**-vector of the spin-spiral state in lattice coordinates | real(3) | $(0.0, 0.0, 0.0)$ |

Spin-spirals arise from spinor states assumed to be of the form

$$\Psi^{\mathbf{q}}_{\mathbf{k}}(\mathbf{r}) = \begin{pmatrix} U^{\mathbf{q}\uparrow}_{\mathbf{k}}(\mathbf{r}) e^{i(\mathbf{k}+\mathbf{q}/\mathbf{2})\cdot\mathbf{r}} \\ U^{\mathbf{q}\downarrow}_{\mathbf{k}}(\mathbf{r}) e^{i(\mathbf{k}-\mathbf{q}/\mathbf{2})\cdot\mathbf{r}} \end{pmatrix}.$$

These are determined using a second-variational approach, and give rise to a magnetisation density of the form

$$\mathbf{m}^{\mathbf{q}}(\mathbf{r}) = (m_x(\mathbf{r})\cos(\mathbf{q}\cdot\mathbf{r}), m_y(\mathbf{r})\sin(\mathbf{q}\cdot\mathbf{r}), m_z(\mathbf{r})),$$

where $m_x$, $m_y$ and $m_z$ are lattice periodic. See also `spinprl`.

### 5.87 vkloff

| | | | |
|---|---|---|---|
| vkloff | the **k**-point offset vector in lattice coordinates | real(3) | (0.0, 0.0, 0.0) |

See ngridk.

### 5.88 xctype

| | | | |
|---|---|---|---|
| xctype | integer defining the type of exchange-correlation functional to be used | integer | 3 |

Currently implemented are:

1 No exchange-correlation funtional ($E_{\rm xc} \equiv 0$)
2 LDA, Perdew-Zunger/Ceperley-Alder, *Phys. Rev. B* **23**, 5048 (1981)
3 LSDA, Perdew-Wang/Ceperley-Alder, *Phys. Rev. B* **45**, 13244 (1992)
4 LDA, X-alpha approximation, J. C. Slater, *Phys. Rev.* **81**, 385 (1951)
5 LSDA, von Barth-Hedin, *J. Phys. C* **5**, 1629 (1972)
20 GGA, Perdew-Burke-Ernzerhof, *Phys. Rev. Lett.* **77**, 3865 (1996)
21 GGA, Revised PBE, Zhang-Yang, *Phys. Rev. Lett.* **80**, 890 (1998)
22 GGA, PBEsol, arXiv:0707.2088v1 (2007)
26 GGA, Wu-Cohen exchange (WC06) with PBE correlation, *Phys. Rev. B* **73**, 235116 (2006)
30 GGA, Armiento-Mattsson (AM05) spin-unpolarised functional, *Phys. Rev. B* **72**, 085108 (2005)

## 6 Contributing to Elk

Please bear in mind when writing code for the Elk project that it should be an exercise in physics and not software engineering. All code should therefore be kept as simple and concise as possible, and above all it should be easy for anyone to locate and follow the Fortran representation of the original mathematics. We would also appreciate the following conventions being adhered to:

- Strict Fortran 90/95 should be used. Features which are marked as obsolescent in F90/95 should be avoided. These include assigned format specifiers, labeled do-loops, computed goto statements and statement functions.
- Modules should be used in place of common blocks for declaring global variables. Use the existing modules to declare new global variables.
- Any code should be written in lower-case free form style, starting from column one. Try and keep the length of each line to fewer than 80 characters using the & character for line continuation.
- Every function or subroutine, no matter how small, should be in its own file named `routine.f90`, where `routine` is the function or subroutine name. It is recommended that the routines are named so as to make their purpose apparent from the name alone.
- Use of `implicit none` is mandatory. Remember also to define the `intent` of any passed arguments.

- Local allocatable arrays must be deallocated on exit of the routine to prevent memory leakage. Use of automatic arrays should be limited to arrays of small size.
- Every function or subroutine must be documented with the Protex source code documentation system. This should include a short LaTeX description of the algorithms and methods involved. Equations which need to be referenced should be labeled with `routine_1`, `routine_2` etc. The authorship of each new piece of code or modification should be indicated in the `REVISION HISTORY` part of the header. See the Protex documentation for details.
- Ensure as much as possible that a routine will terminate the program when given improper input instead of continuing with erroneous results. Specifically, functions should have a well-defined domain for which they return accurate results. Input outside that domain should result in an error message and termination.
- Report errors prior to termination with a short description, for example:

```
write(*,*)
write(*,'("Error(readinput): natoms <= 0 : ",I8)') natoms(is)
write(*,'(" for species ",I4)') is
write(*,*)
```

- Wherever possible, real numbers outputted as ASCII data should be formatted with the `G18.10` specifier.
- Avoid redundant or repeated code: check to see if the routine you need already exists, before writing a new one.
- All reading in of ASCII data should be done in the subroutine `readinput`. For binary data, separate routines for reading and writing should be used (for example `writestate` and `readstate`).
- Input filenames should be in lowercase and have the extension `.in` . All output filenames should be in uppercase with the extension `.OUT` .
- All internal units should be atomic. Input and output units should be atomic by default and clearly stated otherwise. Rydbergs should not be used under any circumstances.

## 6.1 Licensing

Routines which constitute the main part of the code are released under the GNU General Public License (GPL). Library routines are released under the less restrictive GNU Lesser General Public License (LGPL). Both licenses are contained in the file `COPYING`. Any contribution to the code must be licensed at the authors' discretion under either the GPL or LGPL. Author(s) of the code retain the copyrights. Copyright and (L)GPL information must be included at the beginning of every file, and no code will be accepted without this.

## References

[1] EXCITING code developed under the Research and Training Network EXCITING funded by the EU, contract No. HPRN-CT-2002-00317, 2004.

[2] D. J. Singh. *Planewaves, Pseudopotentials and the LAPW Method*. Kluwer Academic Publishers, Boston, 1994.

# 7 Routine/Function Prologues

## 7.1 xcifc (Source File: modxcifc.f90)

INTERFACE:

```
subroutine xcifc(xctype,n,rho,rhoup,rhodn,grho,gup,gdn,g2rho,g2up,g2dn,g3rho, &
 g3up,g3dn,ex,ec,vx,vc,vxup,vxdn,vcup,vcdn)
```

*INPUT/OUTPUT PARAMETERS:*

```
xctype : type of exchange-correlation functional (in,integer)
n      : number of density points (in,integer,optional)
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)
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)
```

DESCRIPTION:

Interface to the exchange-correlation routines. This makes it relatively simple to add new functionals which do not necessarily depend only on $\rho$.

REVISION HISTORY:

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

---

## 7.2 getxcdata (Source File: modxcifc.f90)

INTERFACE:

```
subroutine getxcdata(xctype,xcdescr,xcspin,xcgrad)
```

*INPUT/OUTPUT PARAMETERS:*

```
xctype  : type of exchange-correlation functional (in,integer)
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 `xctype` 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)
```

---

## 7.3 autoradmt (Source File: autoradmt.f90)

INTERFACE:

```
subroutine autoradmt
```

*USES:*

```
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 $i$th species, $\zeta$ is a user-supplied constant ($\sim 0.25$). The parameter $\zeta$ is stored in `rmtapm(1)` and the value which governs the minimum 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)
```

---

## 7.4 findprim (Source File: findprim.f90)

INTERFACE:

```
subroutine findprim
```

*USES:*

```
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)
```

---

## 7.5 findsym (Source File: findsym.f90)

INTERFACE:

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

*USES:*

```
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)
```

---

## 7.6 findsymcrys (Source File: findsymcrys.f90)

INTERFACE:

```
subroutine findsymcrys
```

*USES:*

```
use modmain
use modtest
```

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, $\alpha_R$ is a spatial rotation operation and $\alpha_S$ 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)
```

---

## 7.7 genppts (Source File: genppts.f90)

INTERFACE:

```
subroutine genppts(reducep,tfbz,ngridp,boxl,nppt,ipmap,ivp,vpl,vpc,wppt)
```

*USES:*

```
use modmain
```

*INPUT/OUTPUT PARAMETERS:*

```
reducep : .true. if p-point set is to be reduced (in,logical)
tfbz    : .true. if vpl and vpc should be mapped to the first Brillouin
          zone (in,logical)
ngridp  : p-point grid size (in,integer(3))
boxl    : corners of box containing p-points in lattice coordinates, the
          first vector is the origin (in,real(3,4))
nppt    : total number of p-points (out,integer)
ipmap   : map from integer grid to p-point index
          (out,integer(0:ngridp(1)-1,0:ngridp(2)-1,0:ngridp(3)-1))
ivp     : integer coordinates of the p-points
          (out,integer(3,ngridp(1)*ngridp(2)*ngridp(3)))
vpl     : lattice coordinates of each p-point
          (out,real(3,ngridp(1)*ngridp(2)*ngridp(3)))
vpc     : Cartesian coordinates of each p-point
          (out,real(3,ngridp(1)*ngridp(2)*ngridp(3)))
wppt    : weights of each p-point (out,real(ngridp(1)*ngridp(2)*ngridp(3)))
```

DESCRIPTION:

This routine is used for generating $k$-point or $q$-point sets. Since these are stored in global arrays, the points passed to this and other routines are referred to as $p$-points. If `reducep` is `.true.` the set is reduced with the spatial part of the crystal symmetries. In lattice coordinates, the $\mathbf{p}$ vectors are given by

$$\mathbf{p} = \begin{pmatrix} \mathbf{B}_2 - \mathbf{B}_1 & \mathbf{B}_3 - \mathbf{B}_1 & \mathbf{B}_4 - \mathbf{B}_1 \end{pmatrix} \begin{pmatrix} i_1/n_1 \\ i_2/n_2 \\ i_3/n_3 \end{pmatrix} + \mathbf{B}_1$$

where $i_j$ runs from 0 to $n_j - 1$, and the $\mathbf{B}$ vectors define the corners of a box with $\mathbf{B}_1$ as the origin. If `tfbz` is `.true.` then the vectors `vpl` (and `vpc`) are mapped to the first Brillouin zone. If `tfbz` is `.false.` and `reducep` is `.true.` then the coordinates of `vpl` are mapped to the $[0, 1)$ interval. The $p$-point weights are stored in `wppt` and the array `ipmap` contains the map from the integer coordinates to the reduced index.

REVISION HISTORY:

```
Created August 2002 (JKD)
Updated April 2007 (JKD)
Added mapping to the first Brillouin zone, September 2008 (JKD)
```

---

## 7.8 rfinp (Source File: rfinp.f90)

INTERFACE:

```
real(8) function rfinp(lrstp,rfmt1,rfmt2,rfir1,rfir2)
```

*USES:*

```
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)
```

---

## 7.9 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)
```

---

### 7.10 seceqn (Source File: seceqn.f90)

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

```
use modmain
```

*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)
```

---

### 7.11 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)
```

---

### 7.12 symrfir (Source File: symrfir.f90)

INTERFACE:

```
subroutine symrfir(ngv,rfir)
```

*USES:*

```
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)
```

---

### 7.13 wavefmt (Source File: wavefmt.f90)

INTERFACE:

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

*USES:*

```
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:

Calculates the first-variational wavefunction in the muffin-tin in terms of a spherical harmonic expansion. For atom $\alpha$ and a particular $k$-point $\mathbf{p}$, the $r$-dependent $(l,m)$-coefficients of the wavefunction for the $i$th 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 $i$th eigenvector returned from routine `seceqn`; $A^{\alpha}_{jlm}(\mathbf{G}+\mathbf{p})$ is the matching coefficient; $M^{\alpha}_{l}$ is the order of the APW; $u^{\alpha}_{jl}$ is the APW radial function; $N^{\alpha}$ is the number of local-orbitals; $v^{\alpha}_{j}$ is the $j$th local-orbital radial function; and $(\alpha,j,m)$ 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)
```

---

## 7.14 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)
```

---

## 7.15 force (Source File: force.f90)

INTERFACE:

```
subroutine force
```

*USES:*

```
use modmain
use modtest
```

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 $\alpha$. 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 $\alpha$; and $H^{\alpha}$ and $O^{\alpha}$ 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)
```

---

## 7.16 elfplot (Source File: elfplot.f90)

INTERFACE:

```
subroutine elfplot
```

*USES:*

```
use modmain
```

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)
```

---

## 7.17 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 $(\theta, \phi)$ 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)
```

---

## 7.18 energy (Source File: energy.f90)

INTERFACE:

```
subroutine energy
```

*USES:*

```
use modmain
use modldapu
use modtest
```

DESCRIPTION:

Computes the total energy and its individual contributions. The kinetic energy is given by

$$T_{\rm s} = \sum_i n_i \epsilon_i - \int \rho(\mathbf{r})[v_{\rm C}(\mathbf{r}) + v_{\rm xc}(\mathbf{r})]d\mathbf{r} - \int \mathbf{m}(\mathbf{r}) \cdot (\mathbf{B}_{\rm xc}(\mathbf{r}) + \mathbf{B}_{\rm ext}(\mathbf{r}))d\mathbf{r},$$

where $n_i$ are the occupancies and $\epsilon_i$ are the eigenvalues of both the core and valence states; $\rho$ is the density; $\mathbf{m}$ is the magnetisation density; $v_{\rm C}$ is the Coulomb potential; $v_{\rm xc}$ and $\mathbf{B}_{\rm xc}$ are the exchange-correlation potential and effective magnetic field, respectively; and $\mathbf{B}_{\rm ext}$ is the external magnetic field. The Hartree, electron-nuclear and nuclear-nuclear electrostatic energies are combined into the Coulomb energy:

$$\begin{aligned} E_{\rm C} &= E_{\rm H} + E_{\rm en} + E_{\rm nn} \\ &= \frac{1}{2} V_{\rm C} + E_{\rm Mad}, \end{aligned}$$

where

$$V_{\rm C} = \int \rho(\mathbf{r}) v_{\rm C}(\mathbf{r}) d\mathbf{r}$$

is the Coulomb potential energy. The Madelung energy is given by

$$E_{\rm Mad} = \frac{1}{2} \sum_{\alpha} z_{\alpha} R_{\alpha},$$

where

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

for atom $\alpha$, with $v^{\rm C}_{\alpha;00}$ being the $l = 0$ component of the spherical harmonic expansion of $v_{\rm C}$ in the muffin-tin, and $z_{\alpha}$ is the nuclear charge. Using the nuclear-nuclear energy determined at the start of the calculation, the electron-nuclear and Hartree energies can be isolated with

$$E_{\rm en} = 2 \left( E_{\rm Mad} - E_{\rm nn} \right)$$

and

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

Finally, the total energy is

$$E = T_{\rm s} + E_{\rm C} + E_{\rm xc},$$

where $E_{\rm xc}$ is obtained either by integrating the exchange-correlation energy density, 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 infintesimal. 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 May 2003 (JKD)`

---

## 7.19 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)
```

---

## 7.20 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, $\rho_{lm}$, 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)
```

---

## 7.21 writegeom (Source File: writegeom.f90)

INTERFACE:

```
subroutine writegeom(topt)
```

*USES:*

```
use modmain
```

*INPUT/OUTPUT PARAMETERS:*

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

DESCRIPTION:

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

REVISION HISTORY:

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

---

## 7.22 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)
```

---

## 7.23 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)
```

---

## 7.24 allatoms (Source File: allatoms.f90)

INTERFACE:

```
subroutine allatoms
```

*USES:*

```
use modmain
```

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)
```

---

## 7.25 gridsize (Source File: gridsize.f90)

INTERFACE:

```
subroutine gridsize
```

*USES:*

```
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 $i$th lattice vector.

REVISION HISTORY:

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

---

## 7.26 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)
```

---

## 7.27 genrmesh (Source File: genrmesh.f90)

INTERFACE:

```
subroutine genrmesh
```

*USES:*

```
use modmain
```

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 `radfinr`.

REVISION HISTORY:

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

---

### 7.28 readfermi (Source File: readfermi.f90)

INTERFACE:

```
subroutine readfermi
```

*USES:*

```
use modmain
```

DESCRIPTION:

Reads the Fermi energy from the file `EFERMI.OUT`.

REVISION HISTORY:

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

---

### 7.29 potcoul (Source File: potcoul.f90)

INTERFACE:

```
subroutine potcoul
```

*USES:*

```
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 coverted to complex representations and passed to the routine `zpotcoul`.

REVISION HISTORY:

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

---

### 7.30 gensfacgp (Source File: gensfacgp.f90)

INTERFACE:

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

*USES:*

```
use modmain
```

*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 $\alpha$.

REVISION HISTORY:

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

---

## 7.31 checkmt (Source File: checkmt.f90)

INTERFACE:

```
subroutine checkmt
```

*USES:*

```
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)
```

---

## 7.32 zfinp (Source File: zfinp.f90)

INTERFACE:

```
complex(8) function zfinp(tsh,zfmt1,zfmt2,zfir1,zfir2)
```

*USES:*

```
use modmain
```

*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)
```

---

### 7.33 match (Source File: match.f90)

INTERFACE:

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

*USES:*

```
use modmain
```

*INPUT/OUTPUT PARAMETERS:*

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

DESCRIPTION:

Computes the $(\mathbf{G}+\mathbf{p})$-dependent matching coefficients for the APW basis functions. Inside muffin-tin $\alpha$, 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, $M^{\alpha}_{l}$ is the order of the APW and $u^{\alpha}_{jl}$ 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 $\Omega$ 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 $R_{\alpha}$ the muffin-tin radius. See routine `wavefmt`.

REVISION HISTORY:

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

---

## 7.34 forcek (Source File: forcek.f90)

INTERFACE:

```
subroutine forcek(ik,ffacg)
```

*USES:*

```
use modmain
```

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)
```

---

### 7.35 writeefg (Source File: writeefg.f90)

INTERFACE:

```
subroutine writeefg
```

*USES:*

```
use modmain
```

DESCRIPTION:

Computes the electric field gradient (EFG) tensor for each atom, $\alpha$, 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)
```

---

### 7.36 packeff (Source File: packeff.f90)

INTERFACE:

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

*USES:*

```
use modmain
use modldapu
```

*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 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)
```

---

## 7.37 findsymlat (Source File: findsymlat.f90)

INTERFACE:

```
subroutine findsymlat
```

*USES:*

```
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 $3 \times 3$ 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)
```

---

## 7.38 genlofr (Source File: genlofr.f90)

INTERFACE:

```
subroutine genlofr
```

*USES:*

```
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)
```

---

### 7.39 atom (Source File: atom.f90)

INTERFACE:

```
subroutine atom(sol,ptnucl,zn,nst,n,l,k,occ,xctype,xcgrad,np,nr,r,eval,rho,vr, &
 rwf)
```

*USES:*

```
use modxcifc
```

*INPUT/OUTPUT PARAMETERS:*

```
sol    : speed of light in atomic units (in,real)
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)
np     : order of predictor-corrector polynomial (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)
```

---

### 7.40 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)
```

---

## 7.41 writekpts (Source File: writekpts.f90)

INTERFACE:

```
subroutine writekpts
```

*USES:*

```
use modmain
```

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)
```

---

## 7.42 fsmfield (Source File: fsmfield.f90)

INTERFACE:

```
subroutine fsmfield
```

*USES:*

```
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 $i$th 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 $\lambda$ is a scaling factor.

REVISION HISTORY:

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

---

## 7.43 mossbauer (Source File: mossbauer.f90)

INTERFACE:

```
subroutine mossbauer
```

*USES:*

```
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)
```

---

## 7.44 occupy (Source File: occupy.f90)

INTERFACE:

```
subroutine occupy
```

*USES:*

```
use modmain
use modtest
```

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)
```

---

### 7.45 writelinen (Source File: writelinen.f90)

INTERFACE:

```
subroutine writelinen
```

*USES:*

```
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)
```

---

### 7.46 writeinfo (Source File: writeinfo.f90)

INTERFACE:

```
subroutine writeinfo(fnum)
```

*USES:*

```
use modmain
use modldapu
```

*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)
Updated with LDA+U quantities July 2009 (FC)
```

---

## 7.47 readinput (Source File: readinput.f90)

INTERFACE:

```
subroutine readinput
```

*USES:*

```
use modmain
use modldapu
use modtest
```

DESCRIPTION:

Reads in the input parameters from the file `elk.in`. Also sets default values for the input parameters.

REVISION HISTORY:

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

---

## 7.48 charge (Source File: charge.f90)

INTERFACE:

```
subroutine charge
```

*USES:*

```
use modmain
use modtest
```

DESCRIPTION:

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

REVISION HISTORY:

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

---

## 7.49 moment (Source File: moment.f90)

INTERFACE:

```
subroutine moment
```

*USES:*

```
use modmain
use modtest
```

DESCRIPTION:

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

REVISION HISTORY:

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

---

## 7.50 writesym (Source File: writesym.f90)

INTERFACE:

```
subroutine writesym
```

*USES:*

```
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)
```

---

## 7.51 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)
```

---

## 7.52 gencore (Source File: gencore.f90)

INTERFACE:

```
subroutine gencore
```

*USES:*

```
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)
```

---

## 7.53 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)
```

---

## 7.54 gengvec (Source File: gengvec.f90)

INTERFACE:

```
subroutine gengvec
```

*USES:*

```
use modmain
use modtest
```

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)
```

---

## 7.55 genshtmat (Source File: genshtmat.f90)

INTERFACE:

```
subroutine genshtmat
```

*USES:*

```
use modmain
```

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 $(\theta,\phi)$ points on the sphere.

REVISION HISTORY:

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

---

## 7.56 plot1d (Source File: plot1d.f90)

INTERFACE:

```
subroutine plot1d(fnum1,fnum2,nf,lmax,ld,rfmt,rfir)
```

*USES:*

```
use modmain
```

*INPUT/OUTPUT PARAMETERS:*

```
fnum1 : plot file number (in,integer)
fnum2 : vertex location 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 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)
```

---

## 7.57 plot2d (Source File: plot2d.f90)

INTERFACE:

```
subroutine plot2d(fnum,nf,lmax,ld,rfmt,rfir)
```

*USES:*

```
use modmain
```

*INPUT/OUTPUT PARAMETERS:*

```
fnum : 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 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)
```

---

## 7.58 plot3d (Source File: plot3d.f90)

INTERFACE:

```
subroutine plot3d(fnum,nf,lmax,ld,rfmt,rfir)
```

*USES:*

```
use modmain
```

*INPUT/OUTPUT PARAMETERS:*

```
fnum : 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)
```

---

## 7.59 updatpos (Source File: updatpos.f90)

INTERFACE:

```
subroutine updatpos
```

*USES:*

```
use modmain
```

DESCRIPTION:

Updates the current atomic positions according to the force on each atom. If $\mathbf{r}_{ij}^{m}$ is the position and $\mathbf{F}_{ij}^{m}$ is the force acting on it for atom $j$ of species $i$ and after time step $m$, then the new position is calculated by

$$\mathbf{r}_{ij}^{m+1}=\mathbf{r}_{ij}^{m}+\tau_{ij}^{m}\left(\mathbf{F}_{ij}^{m}+\mathbf{F}_{ij}^{m-1}\right),$$

where $\tau_{ij}^{m}$ is a parameter governing the size of the displacement. If $\mathbf{F}_{ij}^{m}\cdot\mathbf{F}_{ij}^{m-1}>0$ then $\tau_{ij}^{m}$ is increased, otherwise it is decreased.

REVISION HISTORY:

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

---

### 7.60 writeiad (Source File: writeiad.f90)

INTERFACE:

```
subroutine writeiad(topt)
```

*USES:*

```
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)
```

---

### 7.61 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)
```

---

## 7.62 vecplot (Source File: vecplot.f90)

INTERFACE:

```
subroutine vecplot
```

DESCRIPTION:

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)
```

---

## 7.63 genylmg (Source File: genylmg.f90)

INTERFACE:

```
subroutine genylmg
```

*USES:*

```
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)
```

---

## 7.64 linengy (Source File: linengy.f90)

INTERFACE:

```
subroutine linengy
```

*USES:*

```
use modmain
```

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)
```

---

## 7.65 init0 (Source File: init0.f90)

INTERFACE:

```
subroutine init0
```

*USES:*

```
use modmain
use modxcifc
use modldapu
```

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)
```

---

## 7.66 init1 (Source File: init1.f90)

INTERFACE:

```
subroutine init1
```

*USES:*

```
use modmain
use modldapu
use modtest
```

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)
```

---

### 7.67 gengpvec (Source File: gengpvec.f90)

INTERFACE:

```
subroutine gengpvec(vpl,vpc,ngp,igpig,vgpl,vgpc,gpc,tpgpc)
```

*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)
```

---

### 7.68 genpmat (Source File: genpmat.f90)

INTERFACE:

```
subroutine genpmat(ngp,igpig,vgpc,apwalm,evecfv,evecsv,pmat)
```

*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))
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)
```

---

## 7.69 ggamt (Source File: ggamt.f90)

INTERFACE:

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

*USES:*

```
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)
```

---

## 7.70 ggair (Source File: ggair.f90)

INTERFACE:

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

*INPUT/OUTPUT PARAMETERS:*

```
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)
```

---

## 7.71 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)
```

---

### 7.72 gencfun (Source File: gencfun.f90)

INTERFACE:

```
subroutine gencfun
```

*USES:*

```
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 $i$th species and $\Omega$ 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 $j$th atom of the $i$th species.

REVISION HISTORY:

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

---

### 7.73 genapwfr (Source File: genapwfr.f90)

INTERFACE:

```
subroutine genapwfr
```

*USES:*

```
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)
```

---

## 7.74 seceqnfv (Source File: seceqnfv.f90)

INTERFACE:

```
subroutine seceqnfv(nmatp,ngp,igpig,vgpc,apwalm,evalfv,evecfv)
```

*USES:*

```
use modmain
```

*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)
```

---

## 7.75 getngkmax (Source File: getngkmax.f90)

INTERFACE:

```
subroutine getngkmax
```

*USES:*

```
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)
```

---

## 7.76 dos (Source File: dos.f90)

INTERFACE:

```
subroutine dos
```

*USES:*

```
use modmain
use modtest
```

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)
```

---

## 7.77 rhoinit (Source File: rhoinit.f90)

INTERFACE:

```
subroutine rhoinit
```

*USES:*

```
use modmain
```

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)
```

---

### 7.78 potplot (Source File: potplot.f90)

INTERFACE:

```
subroutine potplot
```

*USES:*

```
use modmain
```

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)
```

---

### 7.79 writestate (Source File: writestate.f90)

INTERFACE:

```
subroutine writestate
```

*USES:*

```
use modmain
use modldapu
```

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)
```

---

### 7.80 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 $\rho_{lm}$ to spherical coordinates $(\theta, \phi)$ 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)
```

---

## 7.81 zpotcoul (Source File: zpotcoul.f90)

INTERFACE:

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

*USES:*

```
use modmain
```

*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 $j$th atom of the $i$th species by

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

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, $\rho^{\mathrm{I}}$, into the muffin-tin are found with

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

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

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

where

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

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

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

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

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

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,

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

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

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

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)
```

---

## 7.82 gndstate (Source File: gndstate.f90)

INTERFACE:

```
subroutine gndstate
```

*USES:*

```
use modmain
use modldapu
```

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)
```

---

## 7.83 rhomagk (Source File: rhomagk.f90)

INTERFACE:

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

*USES:*

```
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 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)
Removed conversion to spherical harmonics, January 2009 (JKD)
Partially de-phased the muffin-tin magnetisation for spin-spirals,
 February 2009 (FC, FB & LN)
```

---

## 7.84 readstate (Source File: readstate.f90)

INTERFACE:

```
subroutine readstate
```

*USES:*

```
use modmain
use modldapu
```

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)
```

---

## 7.85 bandstr (Source File: bandstr.f90)

INTERFACE:

```
subroutine bandstr
```

*USES:*

```
use modmain
```

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)
```

---

## 7.86 writeeval (Source File: writeeval.f90)

INTERFACE:

```
subroutine writeeval
```

*USES:*

```
use modmain
```

DESCRIPTION:

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

REVISION HISTORY:

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

---

## 7.87 rhoplot (Source File: rhoplot.f90)

INTERFACE:

```
subroutine rhoplot
```

*USES:*

```
use modmain
```

DESCRIPTION:

Outputs the charge density, read in from `STATE.OUT`, for 1D, 2D or 3D plotting.

REVISION HISTORY:

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

---

### 7.88 symrvfir (Source File: symrvfir.f90)

subroutine symrvfir(ngv,rvfir) *USES:*

```
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)
```

---

### 7.89 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)
```

---

## 7.90 symrfmt (Source File: symrfmt.f90)

INTERFACE:

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

*USES:*

```
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. See the routine `rotrflm`.

REVISION HISTORY:

```
Created May 2003 (JKD)
Changed from rotzflm to rotrflm, December 2008 (JKD)
```

---

## 7.91 hmlrad (Source File: hmlrad.f90)

INTERFACE:

```
subroutine hmlrad
```

*USES:*

```
use modmain
```

DESCRIPTION:

Calculates the radial Hamiltonian integrals of the APW and local-orbital basis functions. In other words, for spin $\sigma$ 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 $q$th 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)
```

---

## 7.92 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 $\sigma$ 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 $q$th APW radial function for angular momentum $l$; and $v_p^{\sigma;ij}$ is the $p$th local-orbital radial function and has angular momentum $l_p$.

REVISION HISTORY:

```
Created November 2003 (JKD)
```

---

## 7.93 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)
```

---

## 7.94 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

$$H^{\mathrm{I}}(\mathbf{G}+\mathbf{k},\mathbf{G}'+\mathbf{k}) = \frac{1}{2}(\mathbf{G}+\mathbf{k})\cdot(\mathbf{G}'+\mathbf{k})\tilde{\Theta}(\mathbf{G}-\mathbf{G}') + V^{\sigma}(\mathbf{G}-\mathbf{G}'),$$

where $V^{\sigma}$ is the effective interstitial potential for spin $\sigma$ and $\tilde{\Theta}$ is the characteristic function. See routine `gencfun`.

REVISION HISTORY:

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

---

### 7.95 hmlaa (Source File: hmlaa.f90)

INTERFACE:

```
subroutine hmlaa(tapp,is,ia,ngp,apwalm,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)
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)
```

---

### 7.96 getevecfv (Source File: getevecfv.f90)

INTERFACE:

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

*USES:*

```
use modmain
```

*INPUT/OUTPUT PARAMETERS:*

```
vpl    : p-point vector in lattice coordinates (in,real(3))
vgpl   : G+p-vectors in lattice coordinates (out,real(3,ngkmax))
evecfv : first-variational eigenvectors (out,complex(nmatmax,nstfv,nspnfv))
```

DESCRIPTION:

Reads in a first-variational eigenvector from file. If the input $k$-point, $\mathbf{p}$, is not in the reduced set, then the eigenvector of the equivalent point is read in and the required rotation/translation operations applied.

REVISION HISTORY:

```
Created Feburary 2007 (JKD)
Fixed transformation error, October 2007 (JKD, Anton Kozhevnikov)
```

---

## 7.97 genwfsv (Source File: genwfsv.f90)

INTERFACE:

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

*USES:*

```
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)
```

---

## 7.98 rhomagsh (Source File: rhomagsh.f90)

INTERFACE:

```
subroutine rhomagsh
```

*USES:*

```
use modmain
```

DESCRIPTION:

Converts the muffin-tin density and magnetisation from spherical coordinates to a spherical harmonic expansion. See `rhomagk`.

REVISION HISTORY:

```
Created January 2009 (JKD)
```

---

## 7.99 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, $(\alpha,\beta,\gamma)$. 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 $\alpha$ about the $x_3$ axis
2. The $x_1''$-, $x_2''$-, $x_3''$-axes are rotated anticlockwise through an angle $\beta$ about the $x_2'$ axis
3. The $x_1'''$-, $x_2'''$-, $x_3'''$-axes are rotated anticlockwise through an angle $\gamma$ 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)
```

---

### 7.100 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 formulae for the $3j$-symbols, the following provides high accuracy for $j \leq 50$

$$\begin{pmatrix} j_1 & j_2 & j_3 \\ m_1 & m_2 & m_3 \end{pmatrix} =$$

$$(-1)^{j1+j2+m3}\sqrt{\frac{(j_1+m_1)!\,(j_2+m_2)!\,(j_3+m_3)!\,(j_3-m_3)!\,(j_1-m_1)!\,(j_2-m_2)!}{(j_2-j_1+j_3)!\,(j_1-j_2+j_3)!\,(j_1+j_2-j_3)!\,(1+j_1+j_2+j_3)!}}\sum_k(-1)^k$$

$$\frac{(j_2-j_1+j_3)!\,(j_1-j_2+j_3)!\,(j_1+j_2-j_3)!}{(j_3-j_1-m_2+k)!\,(j_3-j_2+m_1+k)!\,(j_1+j_2-j_3-k)!\,k!\,(j_1-m_1-k)!\,(j_2+m_2-k)!},$$

where the sum is over all integers $k$ for which the factorials in the summand are non-negative.

REVISION HISTORY:

```
Created November 2002 (JKD)
```

---

### 7.101 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)
```

---

### 7.102 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)
```

---

### 7.103 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 $3 \times 3$ matrices.

REVISION HISTORY:

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

---

### 7.104 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 $3 \times 3$ matrix with another.

REVISION HISTORY:

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

---

### 7.105 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)
```

---

### 7.106 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 $3 \times 3$ matrix with a vector.

REVISION HISTORY:

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

---

### 7.107 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 $3 \times 3$ matrix with a vector.

REVISION HISTORY:

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

---

### 7.108 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)
```

---

### 7.109 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)
```

---

## 7.110 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)
```

---

## 7.111 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)
```

---

### 7.112 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 $3 \times 3$ matrix.

REVISION HISTORY:

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

---

### 7.113 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 $3 \times 3$ matrix $A$.

REVISION HISTORY:

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

---

### 7.114 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 $\mathrm{frac}\,(x) = x - \lfloor x \rfloor$. 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)
```

---

### 7.115 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 $3 \times 3$ matrix $A$.

REVISION HISTORY:

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

---

### 7.116 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 $3 \times 3$ matrix with a vector.

REVISION HISTORY:

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

---

### 7.117 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

$$n \underbrace{!!\ldots!}_{m\,\text{times}} = \prod_{i\geq 0,\, n-im>0} n - im$$

for $n$, $m \geq 0$. $n$ should be less than 150.

REVISION HISTORY:

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

---

### 7.118 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)
```

---

### 7.119 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 $n$th 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)
```

---

## 7.120 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 $\omega_i$, this routine computes the integrals

$$g(\omega_i) = \frac{\Omega}{(2\pi)^3} \int_{\text{BZ}} f(\mathbf{k})\delta(\omega_i - e(\mathbf{k}))d\mathbf{k},$$

where $\Omega$ 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 $\omega_i$ 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)
```

---

### 7.121 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 $(r, \theta, \phi)$ of a vector

$$\mathbf{v} = (r\sin(\theta)\cos(\phi), r\sin(\theta)\sin(\phi), r\cos(\theta)).$$

REVISION HISTORY:

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

---

### 7.122 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 $(\theta, \phi)$ 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)
```

---

### 7.123 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)
```

---

### 7.124 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-Gordon 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)
```

---

### 7.125 wigner6j (Source File: wigner6j.f90)

INTERFACE:

```
real(8) function wigner6j(j1,j2,j3,k1,k2,k3)
```

*INPUT/OUTPUT PARAMETERS:*

```
j1, j2, j3 : angular momentum quantum numbers (in,integer)
k1, k2, k3 : angular momentum quantum numbers (in,integer)
```

DESCRIPTION:

Returns the Wigner $6j$-symbol for integer arguments. This is computed using the Racah formula:

$$\begin{Bmatrix} j_1 & j_2 & j_3 \\ k_1 & k_2 & k_3 \end{Bmatrix} = \sqrt{\Delta(j_1\, j_2\, j_3)\Delta(j_1\, k_2\, k_3)\Delta(k_1\, j_2\, k_3)\Delta(k_1\, k_2\, j_3)} \sum_n \frac{(-1)^n (n+1)!}{f(n)},$$

where

$$\begin{aligned} f(n) =& (n - j_1 - j_2 - j_3)!\,(n - j_1 - k_2 - k_3)!\,(n - k_1 - j_2 - k_3)!\,(n - k_1 - k_2 - j_3)! \\ & \times (j_1 + j_2 + k_1 + k_2 - n)!\,(j_2 + j_3 + k_2 + k_3 - n)!\,(j_1 + j_3 + k_1 + k_3 - n)! \end{aligned}$$

and

$$\Delta(a, b, c) = \frac{(a + b - c)!\,(a - b + c)!\,(-a + b + c)!}{(a + b + c + 1)!}$$

is the triangle coefficient, and where the sum is over all integers $n$ for which the factorials in $f(n)$ have non-negative arguments. The Wigner-$6j$ function is zero unless each triad, $(j_1\, j_2\, j_3)$, $(j_1\, k_2\, k_3)$, $(k_1\, j_2\, k_3)$ and $(k_1\, k_2\, j_3)$, satifies the triangle inequalities:

$$|x - y| \leq z \leq x + y,$$

for triad $(x, y, z)$. See, for example, A. Messiah, *Quantum Mechanics*, Vol. 2., 1061-1066 (1962).

REVISION HISTORY:

```
Created August 2009 (JKD)
```

---

### 7.126 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)
```

---

## 7.127 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 $m$th 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)
```

---

### 7.128 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 $(\theta, \phi)$ 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)
```

---

### 7.129 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 $(\theta, \phi)$ 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)
```

---

## 7.130 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)
```

---

## 7.131 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

$$(L_x + iL_y)Y_{lm}(\theta,\phi) = \sqrt{(l-m)(l+m+1)}Y_{lm+1}(\theta,\phi)$$
$$(L_x - iL_y)Y_{lm}(\theta,\phi) = \sqrt{(l+m)(l-m+1)}Y_{lm-1}(\theta,\phi)$$
$$L_zY_{lm}(\theta,\phi) = mY_{lm}(\theta,\phi).$$

REVISION HISTORY:

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

---

## 7.132 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)
```

---

### 7.133 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)
```

---

### 7.134 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)
```

---

## 7.135 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)
```

---

## 7.136 findband (Source File: findband.f90)

INTERFACE:

```
subroutine findband(sol,l,k,np,nr,r,vr,de0,eps,e,fnd)
```

*INPUT/OUTPUT PARAMETERS:*

```
sol : speed of light in atomic units (in,real)
l   : angular momentum quantum number (in,integer)
k   : quantum number k, zero if Dirac eqn. is not to be used (in,integer)
np  : order of predictor-corrector polynomial (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)
eps : energy search tolerance (in,real)
e   : input energy and returned band energy (inout,real)
fnd : set to .true. if the band energy is found (out,logical)
```

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)
```

---

## 7.137 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 formula (see, for example, V. Devanathan, *Angular Momentum Techniques In Quantum Mechanics*)

$$\nabla^s_\mu f_{lm}(r)Y_{lm}(\hat{\mathbf{r}}) = \sqrt{\frac{l+1}{2l+3}}C(l,1,l+1|m,\mu,m+\mu)Y_{l+1m+\mu}(\hat{\mathbf{r}})\left(\frac{d}{dr}-\frac{l}{r}\right)f_{lm}(r)$$
$$-\sqrt{\frac{l}{2l-1}}C(l,1,l-1|m,\mu,m+\mu)Y_{l-1,m+\mu}(\hat{\mathbf{r}})\left(\frac{d}{dr}+\frac{l+1}{r}\right)f_{lm}(r),$$

where $C$ are Clebsch-Gordan coefficients and the gradient $\nabla^s_\mu$ is in terms of the spherical unit vectors $\hat{\mathbf{e}}_\mu$:

$$\hat{\mathbf{e}}_{+1} = -\frac{\hat{\mathbf{x}}+i\hat{\mathbf{y}}}{\sqrt{2}}, \qquad \hat{\mathbf{e}}_0 = \hat{\mathbf{z}}, \qquad \hat{\mathbf{e}}_{-1} = \frac{\hat{\mathbf{x}}-i\hat{\mathbf{y}}}{\sqrt{2}}.$$

Note that the gradient returned is in terms of the global $(\hat{\mathbf{x}},\hat{\mathbf{y}},\hat{\mathbf{z}})$ coordinate system.

REVISION HISTORY:

Rewritten May 2009 (JKD)

---

## 7.138 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)

---

### 7.139 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)
```

---

### 7.140 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)
```

---

### 7.141 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 $(\theta,\phi)$. Note that `zflm1` and `zflm2` can refer to the same array.

REVISION HISTORY:

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

---

### 7.142 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 the 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 complex 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 $(\alpha, \beta, \gamma)$ of $R^{-1}$ (see routine `euler`) and then applying the spherical harmonic rotation matrix generated by the routine `ylmrot`.

REVISION HISTORY:

```
Created April 2003 (JKD)
Modified, December 2008 (JKD)
```

---

## 7.143 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 $m$th 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)
```

---

## 7.144 sdelta (Source File: sdelta.f90)

INTERFACE:

```
real(8) 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
5. Lorentzian

See routines `stheta`, `sdelta_mp`, `sdelta_fd` and `sdelta_sq`.

REVISION HISTORY:

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

---

## 7.145 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)
```

---

### 7.146 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)
```

---

### 7.147 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 $j$th-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)
```

---

### 7.148 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 $j$th-order Hermite polynomial. This procedure is numerically stable and accurate to near machine precision for $N \leq 10$.

REVISION HISTORY:

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

---

### 7.149 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)
```

---

### 7.150 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)
```

---

### 7.151 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

$$\tilde{\delta}(x) = \begin{cases} 1 & |x| \leq 1/2 \\ 0 & |x| > 1/2 \end{cases}$$

REVISION HISTORY:

```
Created July 2008 (JKD)
```

---

### 7.152 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)
```

---

### 7.153 rdiracint (Source File: rdiracint.f90)

INTERFACE:

```
subroutine rdiracint(sol,m,kpa,e,np,nr,r,vr,nn,g0p,f0p,g0,g1,f0,f1)
```

*INPUT/OUTPUT PARAMETERS:*

```
sol : speed of light in atomic units (in,real)
m   : order of energy derivative (in,integer)
kpa : quantum number kappa (in,integer)
e   : energy (in,real)
np  : order of predictor-corrector polynomial (in,integer)
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))
```

DESCRIPTION:

Integrates the $m$th energy derivative of the radial Dirac equation from $r = 0$ outwards. This involves using the predictor-corrector method to solve the coupled first-order equations (in atomic units)

$$\left(\frac{d}{dr} + \frac{\kappa}{r}\right) G_\kappa^{(m)} = \frac{1}{c}\{2E_0 + E - V\}F_\kappa^{(m)} + \frac{m}{c}F_\kappa^{(m-1)}$$
$$\left(\frac{d}{dr} - \frac{\kappa}{r}\right) F_\kappa^{(m)} = -\frac{1}{c}\{E - V\}G_\kappa^{(m)} - \frac{m}{c}G_\kappa^{(m-1)},$$

where $G_\kappa^{(m)} = rg_\kappa^{(m)}$ and $F_\kappa^{(m)} = rf_\kappa^{(m)}$ are the $m$th energy derivatives of the major and minor components multiplied by $r$, respectively; $V$ is the external potential; $E_0$ is the electron rest energy; $E$ is the eigen energy (excluding $E_0$); and $\kappa = l$ for $j = l - \frac{1}{2}$ or $\kappa = -(l + 1)$ for $j = l + \frac{1}{2}$. If $m = 0$ then the arrays `g0p` and `f0p` are not referenced.

REVISION HISTORY:

Created September 2002 (JKD)

---

## 7.154 rdiracdme (Source File: rdiracdme.f90)

INTERFACE:

```
subroutine rdiracdme(sol,m,kpa,e,np,nr,r,vr,nn,g0,g1,f0,f1)
```

*INPUT/OUTPUT PARAMETERS:*

```
sol : speed of light in atomic units (in,real)
m   : order of energy derivative (in,integer)
kpa : quantum number kappa (in,integer)
e   : energy (in,real)
np  : order of predictor-corrector polynomial (in,integer)
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 $m$th energy derivative of the radial Dirac equation using the routine `rdiracint`.

REVISION HISTORY:

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

---

### 7.155 rdirac (Source File: rdirac.f90)

INTERFACE:

```
subroutine rdirac(sol,n,l,k,np,nr,r,vr,eval,g0,f0)
```

*INPUT/OUTPUT PARAMETERS:*

```
sol  : speed of light in atomic units (in,real)
n    : principal quantum number (in,integer)
l    : quantum number l (in,integer)
k    : quantum number k (l or l+1) (in,integer)
np   : order of predictor-corrector polynomial (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))
```

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)
```

---

### 7.156 rschrodint (Source File: rschrodint.f90)

INTERFACE:

```
subroutine rschrodint(sol,m,l,e,np,nr,r,vr,nn,p0p,p0,p1,q0,q1)
```

*INPUT/OUTPUT PARAMETERS:*

```
sol : speed of light in atomic units (in,real)
m   : order of energy derivative (in,integer)
l   : angular momentum quantum number (in,integer)
e   : energy (in,real)
np  : order of predictor-corrector polynomial (in,integer)
```

```
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 $m$th energy derivative of the scalar relativistic radial Schrödinger equation from $r = 0$ outwards. This involves using the predictor-corrector method to solve the coupled first-order equations (in atomic units)

$$\frac{d}{dr}P_l^{(m)} = 2MQ_l^{(m)} + \frac{1}{r}P_l^{(m)}$$
$$\frac{d}{dr}Q_l^{(m)} = -\frac{1}{r}Q_l^{(m)} + \left[\frac{l(l+1)}{2Mr^2} + (V-E)\right]P_l^{(m)} - mP_l^{(m-1)},$$

where $V$ is the external potential, $E$ is the eigenenergy and $M = 1 - V/2c^2$. Following the convention of Koelling and Harmon, *J. Phys. C: Solid State Phys.* **10**, 3107 (1977), the functions $P_l$ and $Q_l$ are defined by

$$P_l = rg_l$$
$$Q_l = \frac{r}{2M}\frac{dg_l}{dr},$$

where $g_l$ is the major component of the Dirac equation (see the routine `rdiracint`). Note that in order to make the equations linear in energy, the full definition $M = 1+(E-V)/2c^2$ is not used. If $m = 0$ then the array `p0p` is not referenced.

REVISION HISTORY:

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

---

## 7.157 rschroddme (Source File: rschroddme.f90)

INTERFACE:

```
subroutine rschroddme(sol,m,l,k,e,np,nr,r,vr,nn,p0,p1,q0,q1)
```

*INPUT/OUTPUT PARAMETERS:*

```
sol : speed of light in atomic units (in,real)
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)
np  : order of predictor-corrector polynomial (in,integer)
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 $m$th energy derivative of the scalar relativistic radial Schrödinger equation using the routine `rschrodint`.

REVISION HISTORY:

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

---

### 7.158 rschrodapp (Source File: rschrodapp.f90)

INTERFACE:

```
subroutine rschrodapp(sol,l,nr,r,vr,p0,q0,q1,hp0)
```

*INPUT/OUTPUT PARAMETERS:*

```
sol : speed of light in atomic units (in,real)
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)
```

---

### 7.159 reciplat (Source File: reciplat.f90)

INTERFACE:

```
subroutine reciplat
```

*USES:*

```
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)
```

---

### 7.160 connect (Source File: connect.f90)

INTERFACE:

```
subroutine connect(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.

REVISION HISTORY:

```
Created June 2003 (JKD)
Improved September 2007 (JKD)
```

---

## 7.161 flushifc (Source File: flushifc.f90)

INTERFACE:

```
subroutine flushifc(fnum)
```

*INPUT/OUTPUT PARAMETERS:*

```
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)
```

---

## 7.162 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 (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)
```

---

### 7.163 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)
```

---

### 7.164 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)
```

---

### 7.165 rfmtctof (Source File: rfmtctof.f90)

INTERFACE:

```
subroutine rfmtctof(rfmt)
```

*INPUT/OUTPUT PARAMETERS:*

```
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)
```

---

### 7.166 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, not referenced if m=-2 or m=-3 (out,real(4,n))
```

DESCRIPTION:

Given function $f$ defined on a set of points $x_i$ then if $m \geq 0$ this routine computes the $m$th 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. If $m = -1$ then an accurate integral is computed by fitting the function to a clamped cubic spline. When $m = -2$ the fast but low accuracy trapezoidal integration method is used. Simpson's integration, which is slower but more accurate than the trapezoidal method, is used if $m = -3$.

REVISION HISTORY:

```
Created May 2002 (JKD)
```

---

### 7.167 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)
```

---

### 7.168 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 $\theta$. 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)
```

---

## 7.169 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 $3\times 3$ matrix: $B=A^{-1}$.

REVISION HISTORY:

```
Created November 2003 (JKD)
```

---

## 7.170 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 $\theta$. 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)
```

---

## 7.171 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 $2 \times 2$ matrices. Note that the output matrix cannot be one of the input matrices.

REVISION HISTORY:

```
Created October 2007 (JKD)
```

---

## 7.172 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 $2 \times 2$ matrix with another. Note that the output matrix cannot be one of the input matrices.

REVISION HISTORY:

```
Created October 2007 (JKD)
```

---

### 7.173 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 $2 \times 2$ 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)
```

---

### 7.174 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)
```

---

### 7.175 mixadapt (Source File: mixadapt.f90)

INTERFACE:

```
subroutine mixadapt(iscl,beta0,betamax,n,nu,mu,beta,f,d)
```

*INPUT/OUTPUT PARAMETERS:*

```
iscl    : self-consistent loop number (in,integer)
beta0   : mixing parameter (in,real)
betamax : maximum mixing parameter (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 $\mu^i$ and output vector $\nu^i$ of the $i$th self-consistent loop, this routine generates the next input vector to the loop using an adaptive mixing scheme. The $j$th 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 $\beta_j^{i+1} = (1 + \beta_0)\beta_j^i + \beta_0$ 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 $\beta_j^{i+1} = \beta_0(1 + \beta_j^i)$. $\beta_j^i$ is not allowed to exceed $\beta_{\max}$ and is also initialised to this value when $i \leq 1$. 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 thread-safe so long as each thread has its own independent work arrays. Complex arrays may be passed as real arrays with $n$ doubled.

REVISION HISTORY:

```
Created March 2003 (JKD)
Modified, September 2008 (JKD)
```

---

### 7.176 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)
```

---

### 7.177 ylmrot (Source File: ylmrot.f90)

INTERFACE:

```
subroutine ylmrot(p,alpha,beta,gamma,lmax,ld,d)
```

*INPUT/OUTPUT PARAMETERS:*

```
p     : if p=-1 then the rotation matrix is improper (in,integer)
alpha : first Euler angle (in,real)
beta  : second Euler angle (in,real)
gamma : third Euler angle (in,real)
lmax  : maximum angular momentum (in,integer)
ld    : leading dimension (in,integer)
d     : complex spherical harmonic rotation matrix (out,complex(ld,*))
```

DESCRIPTION:

Returns the rotation matrix in the basis of complex spherical harmonics given the three Euler angles, $(\alpha, \beta, \gamma)$, and the parity, $p$, of the rotation. The matrix is given by the formula

$$D^l_{m_1 m_2}(\alpha, \beta, \gamma) = d^l_{m_1 m_2}(\beta) e^{-i(m_1 \alpha + m_2 \gamma)},$$

where $d$ is the rotation matrix about the $y$-axis. For improper rotations, i.e. those which are a combination of a rotation and inversion, $D$ is modified with $D^l_{m_1 m_2} \rightarrow (-1)^l D^l_{m_1 m_2}$. See the routines `euler` and `ylmroty`.

REVISION HISTORY:

```
Created December 2008 (JKD)
```

---

### 7.178 ylmroty (Source File: ylmroty.f90)

INTERFACE:

```
subroutine ylmroty(beta,lmax,ld,dy)
```

*INPUT/OUTPUT PARAMETERS:*

```
beta : rotation angle about y-axis (in,real)
lmax : maximum angular momentum (in,integer)
ld   : leading dimension (in,integer)
dy   : rotation matrix for complex spherical harmonics (out,real(ld,*))
```

DESCRIPTION:

Returns the rotation matrix in the basis of complex spherical harmonics for a rotation of angle $\beta$ about the $y$-axis. This matrix is real and is given by the formula

$$d^l_{m_1m_2}(\beta) = [(l+m_1)!(l-m_1)!(l+m_2)!(l-m_2)!]^{1/2} \times \sum_k (-1)^k \frac{\left(\cos\frac{\beta}{2}\right)^{2(l-k)-m_2+m_1} \left(\sin\frac{\beta}{2}\right)^{2k+m_2-m_1}}{k!(l+m_1-k)!(l-m_2-k)!(m_2-m_1+k)!},$$

where $k$ runs through all integer values for which the factorials exist.

REVISION HISTORY:

```
Created December 2008 (JKD)
```

---

### 7.179 rlmrot (Source File: rlmrot.f90)

INTERFACE:

```
subroutine rlmrot(p,alpha,beta,gamma,lmax,ld,d)
```

*INPUT/OUTPUT PARAMETERS:*

```
p     : if p=-1 then the rotation matrix is improper (in,integer)
alpha : first Euler angle (in,real)
beta  : second Euler angle (in,real)
gamma : third Euler angle (in,real)
lmax  : maximum angular momentum (in,integer)
ld    : leading dimension (in,integer)
d     : real spherical harmonic rotation matrix (out,real(ld,*))
```

DESCRIPTION:

Returns the rotation matrix in the basis of real spherical harmonics given the three Euler angles, $(\alpha, \beta, \gamma)$, and the parity, $p$, of the rotation. The matrix is determined using the

formula of V. V. Nechaev, [*J. Struct. Chem.* **35**, 115 (1994)], suitably modified for our definition of the real spherical harmonics ($m_1 > 0$, $m_2 > 0$):

$\Delta^l_{00} = d^l_{00}$,
$\Delta^l_{m_10} = \sqrt{2}\,(-1)^{m_1} d^l_{0m_1} \cos(m_1\alpha)$,
$\Delta^l_{0m_2} = \sqrt{2}\,(-1)^{m_2} d^l_{m_20} \cos(m_2\gamma)$,
$\Delta^l_{-m_10} = -\sqrt{2}\, d^l_{0m_1} \sin(m_1\alpha)$,
$\Delta^l_{0-m_2} = \sqrt{2}\, d^l_{m_20} \sin(m_2\gamma)$,
$\Delta^l_{m_1m_2} = (-1)^{m_1}(-1)^{m_2}\{\cos(m_1\alpha)\cos(m_2\gamma)[d_A + d_B] - \sin(m_1\alpha)\sin(m_2\gamma)[d_A - d_B]\}$,
$\Delta^l_{m_1-m_2} = (-1)^{m_1}\{\sin(m_1\alpha)\cos(m_2\gamma)[d_A - d_B] + \cos(m_1\alpha)\sin(m_2\gamma)[d_A + d_B]\}$,
$\Delta^l_{-m_1m_2} = -(-1)^{m_2}\{\sin(m_1\alpha)\cos(m_2\gamma)[d_A + d_B] + \cos(m_1\alpha)\sin(m_2\gamma)[d_A - d_B]\}$,
$\Delta^l_{-m_1-m_2} = \cos(m_1\alpha)\cos(m_2\gamma)[d_A - d_B] - \sin(m_1\alpha)\sin(m_2\gamma)[d_A + d_B]$,

where $d_A \equiv d^l_{-m_1-m_2}$, $d_B \equiv (-1)^{m_1} d^l_{m_1-m_2}$ and $d$ is the rotation matrix about the $y$-axis for complex spherical harmonics. See the routines `genrlm`, `euler` and `ylmroty`.

REVISION HISTORY:

`Created December 2008 (JKD)`

---

## 7.180 rotrflm (Source File: rotrflm.f90)

INTERFACE:

```
subroutine rotrflm(rot,lmax,n,ld,rflm1,rflm2)
```

*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)
rflm1 : coefficients of the real spherical harmonic expansion for each
        function (in,real(ld,n))
rflm2 : coefficients of rotated functions (out,complex(ld,n))
```

DESCRIPTION:

Rotates a set of real functions

$$f_i(\mathbf{r}) = \sum_{lm} f^i_{lm} R_{lm}(\hat{\mathbf{r}})$$

for all $i$, given the coefficients $f^i_{lm}$ and a rotation matrix $R$. This is done by first the computing the Euler angles $(\alpha, \beta, \gamma)$ of $R^{-1}$ (see routine `euler`) and then applying the spherical harmonic rotation matrix generated by the routine `rlmrot`.

REVISION HISTORY:

`Created December 2008 (JKD)`

---

### 7.181 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)
```

---

### 7.182 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)
```

---

### 7.183 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)
```

*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 $\kappa$, 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)
```

---

### 7.184 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
```

---

### 7.185 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)
```

---

### 7.186 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)
```

---

### 7.187 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)
```

---

### 7.188 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)
```

---

### 7.189 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:

$X_\alpha$ 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)
```

---

### 7.190 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)
```

---

### 7.191 vnlrho (Source File: vnlrho.f90)

INTERFACE:

```
subroutine vnlrho(tsh,wfmt1,wfmt2,wfir1,wfir2,zrhomt,zrhoir)
```

*USES:*

```
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)
```

---

## 7.192 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)
```

---

## 7.193 genwiq2 (Source File: genwiq2.f90)

INTERFACE:

```
subroutine genwiq2
```

*USES:*

```
use modmain
use modtest
```

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

$$w_{\mathbf{q}_i} \equiv \int_{V_i} \frac{1}{q^2} \, d\mathbf{q} \, , \tag{1}$$

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 (1) 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)
```

---

## 7.194 sumrule (Source File: sumrule.f90)

INTERFACE:

```
subroutine sumrule(dynq)
```

*INPUT/OUTPUT PARAMETERS:*

```
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 $\omega_k^0$ is the $k$th eigenvalue of the $\mathbf{q} = 0$ dynamical matrix and $v_{k;i}^0$ the $i$th 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)
```