The EXCITING Code Developers' Guide
Version 0.9.74

J. K. Dewhurst, S. Sharma and C. Ambrosch-Draxl

Contents

1 Introduction

Welcome to the EXCITING code developers' manual! If you're reading this then you would presumably like to understand, change or contribute to the code. This automatically generated manual is intended to make that task as easy as possible.

2 Contributing to EXCITING

Please bear in mind when writing code for the EXCITING 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 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(readparam): invalid spnst : ",I8)') spnst(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 file names should be in lowercase and have the extension `.in` . All output file names 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.

3 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.

J. Kay Dewhurst
Edinburgh, 2006

4 Routine/Function Prologues

4.1 Fortran: Module Interface modmain (Source File: modmain.f90)

Contains all the global variables required by the EXCITING code.

REVISION HISTORY:

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

4.1.1 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 ρ.

REVISION HISTORY:

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

4.1.2 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 labelled 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)
```

4.1.3 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 α and a particular p-point, the r-dependent (l,m)-coefficients of the wavefunction for the ith state are given by

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

where $\Phi^{i\mathbf{p}}$ is the ith eigenvector returned from routine `seceqn`; $A^\alpha_{jlm}(\mathbf{G}+\mathbf{p})$ is the matching coefficient; M^α_l is the order of the APW; u^α_{jl} is the APW radial function; N^α is the number of local-orbitals; v^α_k is the kth local-orbital radial function; and (α,k,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)
```

4.1.4 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)
```

4.1.5 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

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

where

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

and

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

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 $n({\bf r})$:

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

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)
```

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

REVISION HISTORY:

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

4.1.7 rhonorm (Source File: rhonorm.f90)

INTERFACE:

```
subroutine rhonorm
```

USES:

```
use modmain
```

DESCRIPTION:

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

REVISION HISTORY:

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

4.1.8 energy (Source File: energy.f90)

INTERFACE:

```
subroutine energy
```

USES:

```
use modmain
```

DESCRIPTION:

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

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

where n_i are the occupancies and ϵ_i are the eigenvalues of both the core and valence states; ρ 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

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

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

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

where

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

for atom α, 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_α 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. If the global variable `bfinite` is `.false.` then the external magnetic fields are taken to be infinitesimal (i.e. their only purpose is to break the spin symmetry). In this case the external field contribution to the total energy is set to zero. When `bfinite` is `.true.` the integral over the unit cell

$$E_{\bf B{\rm ext}}=\int{\bf m}({\bf r})\cdot{\bf B}_{\rm ext}({\bf r})d{\bf r}$$

is added to the total energy. See `potxc`, `exxengy` and related subroutines.

REVISION HISTORY:

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

4.1.9 spinchar (Source File: spinchar.f90)

INTERFACE:

```
subroutine spinchar(ik,evecsv)
```

USES:

```
use modmain
```

INPUT/OUTPUT PARAMETERS:

```
    ik     : k-point number (in,integer)
    evecsv : second-variational eigenvectors (in,complex(nstsv,nstsv))
```

DESCRIPTION:

Computes the spin character of a second-variational eigenstate. These are given by

$$\xi^q(\sigma) = \sum_p \left|C^q_{\sigma;p}\right|^2,$$

where $C^q_{\sigma;p}$ are the coefficients of the qth second-variational state in the basis of first-variational states labelled with p. The results are stored in the global array `spnchr`.

REVISION HISTORY:

```
    Created December 2005 (JKD)
```

4.1.10 zpotclmt (Source File: zpotclmt.f90)

INTERFACE:

```
subroutine zpotclmt(lmax,nr,r,zpchg,ld,zrhomt,zvclmt)
```

INPUT/OUTPUT PARAMETERS:

```
    lmax   : maximum angular momentum (in,integer)
    nr     : number of radial mesh points (in,integer)
    r      : radial mesh (in,real(nr))
    zpchg  : point charge  at the atomic center (in,complex(natmtot))
    ld     : leading dimension (in,integer)
    zrhomt : muffin-tin charge density (in,complex(ld,nr))
    zvclmt : muffin-tin Coulomb potential (out,complex(ld,nr))
```

DESCRIPTION:

Solves the Poisson equation for the charge density contained in an isolated muffin-tin using the Green's function approach. In other words, the spherical harmonic expansion of the Coulomb potential, V_{lm}, is obtained from the density expansion, ρ_{lm}, by

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

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

REVISION HISTORY:

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

4.1.11 writegeom (Source File: writegeom.f90)

INTERFACE:

```
subroutine writegeom(topt)
```

USES:

```
use modmain
```

INPUT/OUTPUT PARAMETERS:

```
topt : .true. if GEOMETRY-OPT.OUT is to be written
```

DESCRIPTION:

Outputs the lattice vectors and atomic positions to file, in a format which may be then used directly in `exciting.in`. If `topt` is `.false.` then the file name is `GEOMETRY.OUT`, otherwise it is `GEOMETRY_OPT.OUT`.

REVISION HISTORY:

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

4.1.12 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)
```

4.1.13 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)
```

4.1.14 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. Note that the exchange-correlation functional type 2 is used, irrespective of the value of `xctype`.

REVISION HISTORY:

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

4.1.15 gridsize (Source File: gridsize.f90)

INTERFACE:

```
subroutine gridsize
```

USES:

```
use modmain
```

DESCRIPTION:

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

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

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

REVISION HISTORY:

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

4.1.16 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)
```

4.1.17 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`. See routine `radmesh`.

REVISION HISTORY:

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

4.1.18 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)
```

4.1.19 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)
```

4.1.20 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 α.

REVISION HISTORY:

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

4.1.21 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)
```

4.1.22 zfinp (Source File: zfinp.f90)

INTERFACE:

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

USES:

```
 use modmain
```

INPUT/OUTPUT PARAMETERS:

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

DESCRIPTION:

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

REVISION HISTORY:

```
    Created July 2004 (Sharma)
```

4.1.23 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 α, the APW functions are given by

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

where $A^{\alpha}_{jlm}(\mathbf{G}+\mathbf{p})$ is the matching coefficient, M^{α}_l is the order of the APW and u^{α}_{jl} is the radial function. In the interstitial region, an APW function is a plane wave, $\exp(i(\mathbf{G}+\mathbf{p})\cdot\mathbf{r})/\sqrt{\Omega}$, where Ω is the unit cell volume. Ensuring continuity up to the (M^{α}_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

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

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

REVISION HISTORY:

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

4.1.24 force (Source File: force.f90)

INTERFACE:

```
subroutine force
```

USES:

```
use modmain
```

DESCRIPTION:

Computes the various contributions to the atomic forces. In principle, the force acting on a nucleus is simply the gradient at that site of the classical electrostatic potential from the other nuclei and the electronic density. This is a result of the Hellmann-Feynman theorem. However because the basis set is dependent on the nuclear coordinates and is not complete, the Hellman-Feynman force is inacurate and corrections to it are required. The first is the core correction which arises because the core wavefunctions were determined by neglecting the non-spherical parts of the effective potential v_s. Explicitly this is given by

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

for atom α. The second 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 so-called incomplete basis set (IBS) correction to the ith first-variational eigenvalue, ϵ^i, is given by

$$\begin{aligned}\mathbf{F}^{i\alpha}_{\rm FV}=&-\sum_{\mathbf{G},\mathbf{G}'}\left\{\frac{1}{2}(\mathbf{G}+\mathbf{k})\cdot(\mathbf{G}'+\mathbf{k})-\epsilon_i\right\}\Phi^{i*}_{\mathbf{G}+\mathbf{k}}\Phi^{i}_{\mathbf{G}'+\mathbf{k}}i(\mathbf{G}-\mathbf{G}')\tilde{\Theta}_\alpha(\mathbf{G}-\mathbf{G}')e^{-i(\mathbf{G}-\mathbf{G}')\cdot\mathbf{r}_\alpha}\\&-\sum_{\mathbf{G}}\sum_{\mathbf{G}'}{}'\left\{\left(H^{\rm MT_\alpha}_{\mathbf{G}+\mathbf{k},\mathbf{G}'+\mathbf{k}}-\epsilon_i O^{\rm MT_\alpha}_{\mathbf{G}+\mathbf{k},\mathbf{G}'+\mathbf{k}}\right)\Phi^{i*}_{\mathbf{G}+\mathbf{k}}\Phi^{i}_{\mathbf{G}'+\mathbf{k}}i(\mathbf{G}+\mathbf{k})+{\rm c.c.}\right\},\end{aligned}$$

where Φ^i is the first-variational eigenvector; $\tilde{\Theta}_\alpha$ is the form factor of the smooth step function for muffin-tin α; $H^{\rm MT_\alpha}$ and $O^{\rm MT_\alpha}$ are the muffin-tin Hamiltonian and overlap matrices, respectively; and the primed sum is over both the **G**-vectors and local-orbital indices. Having obtained the derivative of the first-variational eigenvalues with respect to the basis set, the derivatives of the second-variational eigenvalues can be obtained with

$$\mathbf{F}^{i\alpha}_{\rm SV}=\sum_j\left|\psi^i_j\right|^2\mathbf{F}^{j\alpha}_{\rm FV},$$

where ψ^i is the second-variational eigenvector. Finally the IBS correction is

$$\mathbf{F}^{\alpha}_{\rm IBS}=\sum_i n_i\mathbf{F}^{i\alpha}_{\rm SV}+\int_{\rm MT_\alpha}v_{\rm s}(\mathbf{r})\nabla\left[\rho(\mathbf{r})-\rho^{\alpha}_{\rm core}(\mathbf{r})\right]\,d\mathbf{r},$$

where n_i are the state occupancies. See routines `hmlaa`, `olpaa`, `hmlalo`, `olpalo`, `energy`, `seceqn` and `gencfun`.

REVISION HISTORY:

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

4.1.25 forcek (Source File: forcek.f90)

INTERFACE:

```
subroutine forcek(ik,ff)
```

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)
```

4.1.26 writeefg (Source File: writeefg.f90)

INTERFACE:

```
subroutine writeefg
```

USES:

```
use modmain
```

DESCRIPTION:

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

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

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

REVISION HISTORY:

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

4.1.27 packeff (Source File: packeff.f90)

INTERFACE:

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

USES:

```
use modmain
```

INPUT/OUTPUT PARAMETERS:

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

DESCRIPTION:

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

REVISION HISTORY:

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

4.1.28 bandchar (Source File: bandchar.f90)

INTERFACE:

```
subroutine bandchar(lmax,ik,evecfv,evecsv,ld,bndchr,elmsym)
```

USES:

```
use modmain
```

INPUT/OUTPUT PARAMETERS:

```
lmax   : maximum angular momentum (in,integer)
ik     : k-point number (in,integer)
evecfv : first-variational eigenvectors (in,complex(nmatmax,nstfv))
evecsv : second-variational eigenvectors (in,complex(nstsv,nstsv))
ld     : leading dimension (in,integer)
bndchr : band character (out,real(ld,natmtot,nspinor,nstsv))
elmsym : eigenvalues of a matrix in the Y_lm basis which has been
         symmetrised with the site symmetries (out,real(ld,natmtot))
```

DESCRIPTION:

Returns the so-called "band characters" of the second-variational states. These are given by

$$\xi^{i\mathbf{p}\alpha}_{lm\sigma} = \int_0^{R_\alpha} \left| \sum_j C^i_{j\sigma} \sum_{m'} v^{\alpha *}_{lmm'} \Psi^{j\mathbf{p}\alpha}_{lm'}(r) \right|^2 r^2 dr$$

where $\Psi^{j\mathbf{p}\alpha}_{lm'}$ are the r-dependent (l,m)-components of the first-variational muffin-tin wave-function for state j, k-point $\mathbf{p}$ and atom α; and $C^i_{j\sigma}$ are the coefficients of spinor component σ of the ith second-variational state. In order to obtain a physically relevant m projection, the vector $v^\alpha_{lmm'}$ is taken to be the mth eigenvector of a random Hermitian matrix of dimension $2l+1$ which has been symmetrised with site symmetries S^α in the spherical harmonic basis:

$$h = \sum_i S^\alpha_i h_0 (S^\alpha_i)^{-1}.$$

Thus the degeneracy of the eigenvalues of h will determine the irreducible representations of the site symmetry group. These eigenvalues are returned in the array `elmsym`. If the global variable `bcsym` is `.false.` then the band characters refer to non-symmetrised m-projections, i.e. $v^\alpha_{lmm'} = \delta_{mm'}$. Band characters give an indication of the spin, atomistic and (l,m)-strengths of each state. See routines `seceqnsv` and `wavefmt`.

REVISION HISTORY:

```
Created December 2003 (JKD)
Fixed problem with second-variational states, November 2006 (JKD)
```

4.1.29 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)
```

4.1.30 atom (Source File: atom.f90)

INTERFACE:

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

USES:

```
use modxcifc
```

INPUT/OUTPUT PARAMETERS:

```
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)
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`. Note that only local spin-unpolarised functionals may be used.

REVISION HISTORY:

```
Created September 2002 (JKD)
Fixed s.c. convergence problem, October 2003 (JKD)
```

4.1.31 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)
```

4.1.32 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)
```

4.1.33 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 ith self-consistent loop and the required moment:

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

where λ is a scaling factor.

REVISION HISTORY:

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

4.1.34 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 $R_{\rm N} = 1.2Z^{1/3}$ fm, where Z is the atomic number.

REVISION HISTORY:

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

4.1.35 occupy (Source File: occupy.f90)

INTERFACE:

```
subroutine occupy
```

USES:

```
use modmain
```

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)
```

4.1.36 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)
```

4.1.37 writeinfo (Source File: writeinfo.f90)

INTERFACE:

```
subroutine writeinfo(fnum)
```

USES:

```
use modmain
```

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)
```

4.1.38 readinput (Source File: readinput.f90)

INTERFACE:

```
subroutine readinput
```

USES:

```
use modmain
```

DESCRIPTION:

Reads in the input parameters from the file `exciting.in` as well as from the species files. Also sets default values for the input parameters.

REVISION HISTORY:

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

4.1.39 charge (Source File: charge.f90)

INTERFACE:

```
subroutine charge
```

USES:

```
use modmain
```

DESCRIPTION:

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

REVISION HISTORY:

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

4.1.40 moment (Source File: moment.f90)

INTERFACE:

```
subroutine moment
```

USES:

```
use modmain
```

DESCRIPTION:

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

REVISION HISTORY:

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

4.1.41 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)
```

4.1.42 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)
```

4.1.43 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 $r > R^{\rm MT}$. In the case of spin-polarised calculations, the effective magnetic field is ignored.

REVISION HISTORY:

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

4.1.44 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)
```

4.1.45 gengvec (Source File: gengvec.f90)

INTERFACE:

```
subroutine gengvec
```

USES:

```
use modmain
```

DESCRIPTION:

Generates a set of **G**-vectors for the Fourier transform of the charge density and potential, subject to the condition $|\mathbf{G}| \leq G_{\rm max}$, 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.

REVISION HISTORY:

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

4.1.46 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 (θ,ϕ) points on the sphere.

REVISION HISTORY:

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

4.1.47 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)
```

4.1.48 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)
```

4.1.49 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` spanning the number of unit cells in the global array `np3d`. See routine `rfarray`.

REVISION HISTORY:

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

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

REVISION HISTORY:

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

4.1.51 writeiad (Source File: writeiad.f90)

INTERFACE:

```
subroutine writeiad
```

USES:

```
use modmain
```

DESCRIPTION:

Outputs the interatomic distances to the file `IAD.OUT`.

REVISION HISTORY:

```
    Created May 2005 (JKD)
```

4.1.52 symvect (Source File: symvect.f90)

INTERFACE:

```
subroutine symvect(vc)
```

INPUT/OUTPUT PARAMETERS:

```
    vc : vectors in Cartesian coordinates for all atoms (in,real(3,natmtot))
```

DESCRIPTION:

Symmetrises a set of input vectors by rotating and averaging over equivalent atoms. The vectors could be atomic forces for example.

REVISION HISTORY:

```
    Created June 2004 (JKD)
```

4.1.53 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)
```

4.1.54 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)
```

4.1.55 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)
```

4.1.56 init0 (Source File: init0.f90)

INTERFACE:

```
subroutine init0
```

USES:

```
use modmain
use modxcifc
```

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)
```

4.1.57 init1 (Source File: init1.f90)

INTERFACE:

```
subroutine init1
```

USES:

```
use modmain
```

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)
```

4.1.58 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)
```

4.1.59 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)
```

4.1.60 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`. Note that `g3rhomt`, `g3upmt` and `g3dnmt` are approximated by $|\nabla\rho|\nabla^2\rho$ etc., because of numerical instability in computing the exact expression.

REVISION HISTORY:

```
Created April 2004 (JKD)
Approximated {\tt g3rhomt} etc., July 2006 (JKD)
```

4.1.61 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)
```

4.1.62 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)
```

4.1.63 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 $G < G_{\rm max}$. The form factors are given by

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

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

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

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

REVISION HISTORY:

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

4.1.64 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)
```

4.1.65 seceqnfv (Source File: seceqnfv.f90)

INTERFACE:

```
 subroutine seceqnfv(ik,ispn,apwalm,evalfv,evecfv)
```

USES:

```
 use modmain
```

INPUT/OUTPUT PARAMETERS:

```
   ik     : k-point number (in,integer)
   ispn   : first-variational spin index (in,integer)
   apwalm : APW matching coefficients
            (in,complex(ngkmax,apwordmax,lmmaxapw,natmtot))
   evalfv : calculated eigenvalues for current k-point for each state
            (out,real(nstfv))
   evecfv : calculated eigenvectors (out,complex(nmatmax,nstfv))
```

DESCRIPTION:

Solves the secular equation,
$$(H-\epsilon O)\Phi=0,$$
for the first-variational states.

REVISION HISTORY:

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

4.1.66 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)
```

4.1.67 dos (Source File: dos.f90)

INTERFACE:

```
subroutine dos
```

USES:

```
use modmain
```

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. Eigenvalues of a matrix in the Y_{lm} basis which has been symmetrised with the site symmetries are written to `ELMSYM.OUT`. This allows for identification of the irreducible representations of the site symmetries, for example e_g or t_{2g}. Spin-up is made positive and spin-down negative for the partial DOS in both the collinear and non-collinear cases and for the total DOS in the collinear case. See the routines `bandchar` and `brzint`.

REVISION HISTORY:

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

4.1.68 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)
```

4.1.69 symrfir (Source File: symrfir.f90)

INTERFACE:

```
subroutine symrfir(sym,rfir,srfir)
```

USES:

```
use modmain
```

INPUT/OUTPUT PARAMETERS:

```
sym   : symmetry (in,integer(3,3))
rfir  : input interstitial function (in,real(ngrid(1)*ngrid(2)*ngrid(3)))
srfir : output interstitial function (out,real(ngrid(1)*ngrid(2)*ngrid(3)))
```

DESCRIPTION:

Applies a symmetry to a real intersitial function, f, defined on an integer grid. In other words, for each integer vector $\mathbf{v}$ the routine returns

$$Sf(\mathbf{v}) \equiv f(S^{-1}\mathbf{v}),$$

where S is a 3×3 symmetry matrix. Note that the arrays `rfir` and `srfir` represent the functions f and Sf respectively, but stored in the usual way with indices begining at 1.

REVISION HISTORY:

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

4.1.70 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 smooth 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)
```

4.1.71 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)
```

4.1.72 writestate (Source File: writestate.f90)

INTERFACE:

```
subroutine writestate
```

USES:

```
use modmain
```

DESCRIPTION:

Writes the charge density, potentials and other relevant variables to the file `STATE.OUT`.

REVISION HISTORY:

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

4.1.73 potxc (Source File: potxc.f90)

INTERFACE:

```
subroutine potxc
```

USES:

```
use modmain
use modxcifc
```

DESCRIPTION:

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

REVISION HISTORY:

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

4.1.74 zpotcoul (Source File: zpotcoul.f90)

INTERFACE:

```
subroutine zpotcoul(nr,nrmax,ld,r,igp0,gpc,jlgpr,ylmgp,sfacgp,zpchg,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))
zpchg  : point charges at the atomic centers (in,complex(natmtot))
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=0 term of the pseudocharge density (out,complex)
```

DESCRIPTION:

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

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

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

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

remembering that

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

should be used for the case ${\bf G}=0$. 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 **G**-vectors. In each muffin-tin the pseudocharge has the form

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

where

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

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

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

which may be used for solving Poisson's equation directly

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

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

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

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,

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

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

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

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

REVISION HISTORY:

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

4.1.75 gndstate (Source File: gndstate.f90)

INTERFACE:

```
 subroutine gndstate
```

USES:

```
 use modmain
```

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)
```

4.1.76 rhovalk (Source File: rhovalk.f90)

INTERFACE:

```
 subroutine rhovalk(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 transformed with a forward SHT and accumulated in the global variable `rhomt`. A similar proccess is used for the intersitial density in which the wavefunction in real-space is obtained from a Fourier transform of the sum of APW functions. The interstitial density is added to the global array `rhoir`. See routines `wavefmt`, `genshtmat` and `seceqn`.

REVISION HISTORY:

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

4.1.77 readstate (Source File: readstate.f90)

INTERFACE:

```
 subroutine readstate
```

USES:

```
 use modmain
```

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)
```

4.1.78 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`. See routine `bandchar`.

REVISION HISTORY:

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

4.1.79 writeeval (Source File: writeeval.f90)

INTERFACE:

```
subroutine writeeval
```

USES:

```
use modmain
```

DESCRIPTION:

Outputs the second-variational eigenvalues, occupancion numbers and spin characters to the file `EIGVAL.OUT`.

REVISION HISTORY:

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

4.1.80 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)
```

4.1.81 symrfmt (Source File: symrfmt.f90)

INTERFACE:

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

USES:

```
use modmain
```

INPUT/OUTPUT PARAMETERS:

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

DESCRIPTION:

Applies a symmetry to a real muffin-tin function. See the routine `rotzfmt`.

REVISION HISTORY:

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

4.1.82 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 σ and atom j of species i, it computes integrals of the form

$$h^{\sigma;ij}_{qq';ll'l''m''} = \begin{cases} \int_0^{R_i} u^{\sigma;ij}_{q;l}(r) H u^{\sigma;ij}_{q';l'}(r) r^2 dr & l''=0 \\ \int_0^{R_i} u^{\sigma;ij}_{q;l}(r) V^{\sigma;ij}_{l''m''}(r) u^{\sigma;ij}_{q';l'}(r) r^2 dr & l''>0 \end{cases},$$

where $u^{\sigma;ij}_{q;l}$ is the qth APW radial function for angular momentum l; H is the Hamiltonian of the radial Schrödinger equation; and $V^{\sigma;ij}_{l''m''}$ is the effective muffin-tin potential. Similar integrals are calculated for APW-local-orbital and local-orbital-local-orbital contributions.

REVISION HISTORY:

```
Created December 2003 (JKD)
```

4.1.83 olprad (Source File: olprad.f90)

INTERFACE:

```
subroutine olprad
```

USES:

```
use modmain
```

DESCRIPTION:

Calculates the radial overlap integrals of the APW and local-orbital basis functions. In other words, for spin σ and atom j of species i, it computes integrals of the form

$$o_{qp}^{\sigma;ij}=\int_0^{R_i}u_{q;l_p}^{\sigma;ij}(r)v_p^{\sigma;ij}(r)r^2dr$$

and

$$o_{pp'}^{\sigma;ij}=\int_0^{R_i}v_p^{\sigma;ij}(r)v_{p'}^{\sigma;ij}(r)r^2dr,\quad l_p=l_{p'}$$

where $u_{q;l}^{\sigma;ij}$ is the qth APW radial function for angular momentum l; and $v_p^{\sigma;ij}$ is the pth local-orbital radial function and has angular momentum l_p.

REVISION HISTORY:

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

4.1.84 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(npmatmax))
```

DESCRIPTION:

Computes the interstitial contribution to the overlap matrix for the APW basis functions. The overlap is given by

$$O^{\rm I}({\bf G}+{\bf k},{\bf G}'+{\bf k})=\tilde{\Theta}({\bf G}-{\bf G}'),$$

where $\tilde{\Theta}$ is the characteristic function. See routine `gencfun`.

REVISION HISTORY:

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

4.1.85 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(npmatmax))
```

DESCRIPTION:

Computes the interstitial contribution to the Hamiltonian matrix for the APW basis functions. The Hamiltonian is given by

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

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

REVISION HISTORY:

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

4.1.86 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(npmatmax))
```

DESCRIPTION:

Calculates the APW-APW contribution to the Hamiltonian matrix.

REVISION HISTORY:

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

4.1.87 euler (Source File: euler.f90)

INTERFACE:

```
subroutine euler(r,ang)
```

INPUT/OUTPUT PARAMETERS:

```
r   : rotation matrix (in,real(3,3))
ang : euler angles (alpha, beta, gamma) (out,real(3))
```

DESCRIPTION:

Given a rotation matrix

$$R(\alpha,\beta,\gamma)=$$
$$\begin{pmatrix} \cos\gamma\cos\beta\cos\alpha-\sin\gamma\sin\alpha & \cos\gamma\cos\beta\sin\alpha+\sin\gamma\cos\alpha & -\cos\gamma\sin\beta \\ -\sin\gamma\cos\beta\cos\alpha-\cos\gamma\sin\alpha & -\sin\gamma\cos\beta\sin\alpha+\cos\gamma\cos\alpha & \sin\gamma\sin\beta \\ \sin\beta\cos\alpha & \sin\beta\sin\alpha & \cos\beta \end{pmatrix},$$

this routine determines the Euler angles, (α,β,γ). This corresponds to the so-called "y-convention", which involves the following successive rotations of the coordinate system:

1. The x_1'-, x_2'-, x_3'-axes are rotated anticlockwise through an angle α about the x_3 axis
2. The x_1''-, x_2''-, x_3''-axes are rotated anticlockwise through an angle β about the x_2' axis
3. The x_1'''-, x_2'''-, x_3'''-axes are rotated anticlockwise through an angle γ about the x_3'' axis

Note that the Euler angles are not necessarily unique for a given rotation matrix.

REVISION HISTORY:

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

4.1.88 wigner3j (Source File: wigner3j.f90)

INTERFACE:

```
real(8) function wigner3j(j1,j2,j3,m1,m2,m3)
```

INPUT/OUTPUT PARAMETERS:

```
j1, j2, j3 : angular momentum quantum numbers (in,integer)
m1, m2, m3 : magnetic quantum numbers (in,integer)
```

DESCRIPTION:

Returns the Wigner $3j$-symbol. There are many equivalent definitions for the $3j$-symbols, the following provides high accuracy for $j \le 50$

$$\begin{pmatrix} j_1 & j_2 & j_3 \\ m_1 & m_2 & m_3 \end{pmatrix} = (-1)^{j1+j2+m3}$$
$$\times \sqrt{\frac{(j_1+m_1)!(j_2+m_2)!(j_3+m_3)!(j_3-m_3)!(j_1-m_1)!(j_2-m_2)!}{(j_2-j_1+j_3)!(j_1-j_2+j_3)!(j_1+j_2-j_3)!(1+j_1+j_2+j_3)!}} \times \sum_{\max(0,j_2-j_3-m_1,j_1-j_3+m_2)}^{\min(j_1+j_2-j_3,j_1-m_1,j_2+m_2)}$$
$$(-1)^k \frac{(j_2-j_1+j_3)!(j_1-j_2+j_3)!(j_1+j_2-j_3)!}{(j_3-j_1-m_2+k)!(j_3-j_2+m_1+k)!(j_1+j_2-j_3-k)!k!(j_1-m_1-k)!(j_2+m_2-k)}.$$

REVISION HISTORY:

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

4.1.89 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)
```

4.1.90 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)
```

4.1.91 r3mm (Source File: r3mm.f90)

INTERFACE:

```
subroutine r3mm(a,b,c)
```

INPUT/OUTPUT PARAMETERS:

```
    a : input matrix 1 (in,real(3,3))
    b : input matrix 2 (in,real(3,3))
    c : output matrix (out,real(3,3))
```

DESCRIPTION:

Multiplies two real 3×3 matrices. The output matrix can also be an input matrix.

REVISION HISTORY:

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

4.1.92 r3mtm (Source File: r3mtm.f90)

INTERFACE:

```
subroutine r3mtm(a,b,c)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix 1 (in,real(3,3))
b : input matrix 2 (in,real(3,3))
c : output matrix (out,real(3,3))
```

DESCRIPTION:

Multiplies the transpose of one real 3×3 matrix with another. The output matrix can also be an input matrix.

REVISION HISTORY:

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

4.1.93 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. The output matrix can also be an input matrix.

REVISION HISTORY:

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

4.1.94 r3mv (Source File: r3mv.f90)

INTERFACE:

```
subroutine r3mv(a,x,y)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix (in,real(3,3))
x : input vector (in,real(3))
y : output vector (out,real(3))
```

DESCRIPTION:

Multiplies a real 3×3 matrix with a vector. The output vector can also be the input vector.

REVISION HISTORY:

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

4.1.95 r3mtv (Source File: r3mtv.f90)

INTERFACE:

```
subroutine r3mtv(a,x,y)
```

INPUT/OUTPUT PARAMETERS:

```
a : input matrix (in,real(3,3))
x : input vector (in,real(3))
y : output vector (out,real(3))
```

DESCRIPTION:

Multiplies the transpose of a real 3×3 matrix with a vector. The output vector can also be the input vector.

REVISION HISTORY:

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

4.1.96 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. The output vector can also be the input vector.

REVISION HISTORY:

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

4.1.97 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)
```

4.1.98 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)
```

4.1.99 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)
```

4.1.100 r3minv (Source File: r3minv.f90)

INTERFACE:

```
subroutine r3minv(a,b)
```

INPUT/OUTPUT PARAMETERS:

```
    a : input matrix (in,real(3,3))
    b : output matrix (in,real(3,3))
```

DESCRIPTION:

Computes the inverse of a real 3×3 matrix. The output matrix can also be the input matrix.

REVISION HISTORY:

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

4.1.101 r3mdet (Source File: r3mdet.f90)

INTERFACE:

```
real(8) function r3mdet(a)
```

INPUT/OUTPUT PARAMETERS:

```
    a : input matrix (in,real(3,3))
```

DESCRIPTION:

Returns the determinant of a real 3×3 matrix A.

REVISION HISTORY:

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

4.1.102 r3frac (Source File: r3frac.f90)

INTERFACE:

```
 subroutine r3frac(eps,v,id)
```

INPUT/OUTPUT PARAMETERS:

```
     eps : zero component tolerance (in,real)
     v   : input vector (inout,real(3))
     id  : 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 `id`.

REVISION HISTORY:

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

4.1.103 i3mdet (Source File: i3mdet.f90)

INTERFACE:

```
 integer function i3mdet(a)
```

INPUT/OUTPUT PARAMETERS:

```
     a : input matrix (in,integer(3,3))
```

DESCRIPTION:

Returns the determinant of an integer 3×3 matrix A.

REVISION HISTORY:

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

4.1.104 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

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

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

REVISION HISTORY:

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

4.1.105 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)
```

4.1.106 hermite (Source File: hermite.f90)

INTERFACE:

```
real(8) function hermite(n,x)
```

INPUT/OUTPUT PARAMETERS:

```
n : order of Hermite polynomial (in,integer)
x : real argument (in,real)
```

DESCRIPTION:

Returns the nth Hermite polynomial. The recurrence relation

$$H_i(x) = 2xH_{i-1}(x) - 2nH_{i-2}(x),$$

with $H_0 = 1$ and $H_1 = 2x$, is used. This procedure is numerically stable and accurate to near machine precision for $n \leq 20$.

REVISION HISTORY:

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

4.1.107 findprim (Source File: findprim.f90)

INTERFACE:

```
subroutine findprim(eps,avec,nspecies,natoms,ld,atposl,plrvc)
```

INPUT/OUTPUT PARAMETERS:

```
eps      : zero vector tolerance (in,real)
avec     : lattice vectors stored column-wise (inout,real(3,3))
nspecies : number of species (in,integer)
natoms   : number atoms for each species (inout,integer(nspecies))
ld       : leading dimension (in,integer)
atposl   : atomic positions in lattice coordinates
           (inout,real(3,ld,nspecies))
plrvc    : polarisation vector at each atom site in lattice coordinates
           (inout,real(3,ld,nspecies))
```

DESCRIPTION:

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

REVISION HISTORY:

```
Created July 2005 (JKD)
```

4.1.108 brzint (Source File: brzint.f90)

INTERFACE:

```
 subroutine brzint(nsm,ngridk,nsk,ikmap,nw,wint,n,ld,e,f,g)
```

INPUT/OUTPUT PARAMETERS:

```
     nsm    : level of smoothing for output function (in,integer)
     ngridk : k-point grid size (in,integer(3))
     nsk    : k-point subdivision grid size (in,integer(3))
     ikmap  : map from grid to k-point set
              (in,integer(0:ngridk(1)-1,0:ngridk(2)-1,0:ngridk(3)-1))
     nw     : number of energy divisions (in,integer)
     wint   : energy interval (in,real(2))
     n      : number of functions to integrate (in,integer)
     ld     : leading dimension (in,integer)
     e      : array of energies as a function of k-points (in,real(ld,*))
     f      : array of weights as a function of k-points (in,real(ld,*))
     g      : output function (out,real(nw))
```

DESCRIPTION:

Given energy and weight functions, e and f, on the Brillouin zone and a set of equidistant energies ω_i, the routine computes the integrals

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

where Ω is the unit cell volume. This is done by first interpolating e and f on a finer k-point grid using the trilinear method. Then for each $e(\mathbf{k})$ on the finer grid the nearest ω_i is found and $f(\mathbf{k})$ is accumulated in $g(\omega_i)$. If the output function is noisy then either `nsk` should be increased or `nw` decreased. Alternatively, the output function can be artificially smoothed up to a level given by `nsm`. See routine `fsmooth`.

REVISION HISTORY:

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

4.1.109 sphcrd (Source File: sphcrd.f90)

INTERFACE:

```
 subroutine sphcrd(v,r,tp)
```

INPUT/OUTPUT PARAMETERS:

```
     v  : input vector (in,real(3))
     r  : length of v (out,real)
     tp : (theta, phi) coordinates (out,real(2))
```

DESCRIPTION:

Returns the spherical coordinates (r, θ, ϕ) of a vector

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

REVISION HISTORY:

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

4.1.110 sphcover (Source File: sphcover.f90)

INTERFACE:

```
subroutine sphcover(ntp,tp)
```

INPUT/OUTPUT PARAMETERS:

```
ntp : number of required points (in,integer)
tp  : (theta, phi) coordinates (out,real(2,ntp))
```

DESCRIPTION:

Produces a set of (θ, ϕ) points which cover the sphere nearly optimally and have no point group symmetries. The set is generated by successively adding points, each of which maximises the minimum distance between it and the existing points. This ensures that any subset consisting of the first m points from the original set is also optimal.

REVISION HISTORY:

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

4.1.111 erf (Source File: erf.f90)

INTERFACE:

```
real(8) function erf(x)
```

INPUT/OUTPUT PARAMETERS:

```
x : real argument (in,real)
```

DESCRIPTION:

Returns the error function 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)
```

4.1.112 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)
```

4.1.113 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)
```

4.1.114 sbesseldm (Source File: sbesseldm.f90)

INTERFACE:

```
subroutine sbesseldm(m,lmax,x,djl)
```

INPUT/OUTPUT PARAMETERS:

```
m    : order of derivatve (in,integer)
lmax : maximum order of Bessel function (in,integer)
x    : real argument (in,real)
djl  : array of returned values (out,real(0:lmax))
```

DESCRIPTION:

Computes the mth derivative of the spherical Bessel function of the first kind, $j_l(x)$, for argument x and $l = 0, 1, \ldots, l_{\max}$. For $x \geq 1$ this is done by repeatedly using the relations

$$\frac{d}{dx}j_l(x) = \frac{l}{x}j_l(x) - j_{l+1}(x)$$
$$j_{l+1}(x) = \frac{2l+1}{x}j_l(x) - j_{l-1}(x).$$

While for $x < 1$ the series expansion of the Bessel function is used

$$\frac{d^m}{dx^m}j_l(x) = \sum_{i=0}^{\infty} \frac{(2i+l)!}{(-2)^i i!(2i+l-m)!(2i+2l+1)!!} x^{2i+l-m}.$$

This procedure is numerically stable and accurate to near machine precision for $l \leq 30$ and $m \leq 6$.

REVISION HISTORY:

```
Created March 2003 (JKD)
Modified to return an array of values, October 2004 (JKD)
```

4.1.115 genylm (Source File: genylm.f90)

INTERFACE:

```
subroutine genylm(lmax,tp,ylm)
```

INPUT/OUTPUT PARAMETERS:

```
lmax : maximum angular momentum (in,integer)
tp   : (theta, phi) coordinates (in,real(2))
ylm  : array of spherical harmonics (out,complex((lmax+1)**2))
```

DESCRIPTION:

Generates a sequence of spherical harmonics, including the Condon-Shortley phase, evaluated at angles (θ,ϕ) for $0 < l < l_{\rm max}$. 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 $l \le 50$.

REVISION HISTORY:

```
Created March 2004 (JKD)
Improved stability, December 2005 (JKD)
```

4.1.116 genrlm (Source File: genrlm.f90)

INTERFACE:

```
subroutine genrlm(lmax,tp,rlm)
```

INPUT/OUTPUT PARAMETERS:

```
lmax : maximum angular momentum (in,integer)
tp   : (theta, phi) coordinates (in,real(2))
rlm  : array of real spherical harmonics (out,real((lmax+1)**2))
```

DESCRIPTION:

Generates a sequence of real spherical harmonics evaluated at angles (θ,ϕ) for $0 < l < l_{\rm max}$. 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 $l \le 50$. See routine `genylm`.

REVISION HISTORY:

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

4.1.117 radmesh (Source File: radmesh.f90)

INTERFACE:

```
 subroutine radmesh(nr,irf,rf,rmin,r)
```

INPUT/OUTPUT PARAMETERS:

```
     nr   : number of mesh points (in,integer)
     irf  : fixed point number (in,integer)
     rf   : fixed point radius (in,real)
     rmin : minimum radius (in,real)
     r    : radial mesh (out,real(n))
```

DESCRIPTION:

Generates a logarithmic radial mesh containing n_r points. The mesh contains a fixed point $i_{\rm f}$ corresponding to radius $r_{\rm f}$. Mesh radii are given by

$$r(i) = r_{\rm min} \exp\left(\frac{i-1}{i_{\rm f}-1}t\right),$$

where $t = \log(r_{\rm f}/r_{\rm min})$.

REVISION HISTORY:

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

4.1.118 zmatrix (Source File: zmatrix.f90)

INTERFACE:

```
 subroutine zmatrix(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 operation

$$A_{ij} \to \alpha \mathbf{x}_i^* \mathbf{y}_j + \alpha^* \mathbf{y}_i^* \mathbf{x}_j + A_{ij},$$

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)
```

4.1.119 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

$$\begin{aligned}
(L_x + iL_y)Y_{lm}(\theta,\phi) &= \sqrt{(l-m)(l+m+1)}Y_{lm+1}(\theta,\phi) \\
(L_x - iL_y)Y_{lm}(\theta,\phi) &= \sqrt{(l+m)(l-m+1)}Y_{lm-1}(\theta,\phi) \\
L_z Y_{lm}(\theta,\phi) &= mY_{lm}(\theta,\phi).
\end{aligned}$$

REVISION HISTORY:

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

4.1.120 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)
```

4.1.121 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)
```

4.1.122 zfmtinp (Source File: zfmtinp.f90)

INTERFACE:

```
complex(8) function zfmtinp(lmax,nr,r,ld,zfmt1,zfmt2)
```

INPUT/OUTPUT PARAMETERS:

```
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 function inside muffin-tin (in,complex(ld,nr))
zfmt2 : second complex function inside muffin-tin (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

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

The radial integral is performed using low accuracy trapezoidal integration.

REVISION HISTORY:

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

4.1.123 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

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

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

REVISION HISTORY:

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

4.1.124 findband (Source File: findband.f90)

INTERFACE:

```
subroutine findband(l,np,nr,r,vr,de0,e)
```

INPUT/OUTPUT PARAMETERS:

```
l   : angular momentum quantum number (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)
e   : input energy and returned band energy (inout,real)
```

DESCRIPTION:

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

REVISION HISTORY:

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

4.1.125 gradzfmt (Source File: gradzfmt.f90)

INTERFACE:

```
subroutine gradzfmt(lmax,nr,r,ld1,ld2,zfmt,gzfmt)
```

INPUT/OUTPUT PARAMETERS:

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

DESCRIPTION:

Calculates the gradient of a complex muffin-tin function. In other words, given the spherical harmonic expansion coefficients, $f_{lm}(r)$, of a function $f(\mathbf{r})$, the routine returns $\mathbf{F}_{lm}$ where

$$\sum_{lm} \mathbf{F}_{lm}(r) Y_{lm}(\hat{\mathbf{r}}) = \nabla f(\mathbf{r}).$$

This is done using the identity

$$\begin{aligned}\nabla\left[f_{lm}(r)Y_{lm}(\hat{\mathbf{r}})\right] = &-\left[\frac{l+1}{2l+1}\right]^{1/2}\left[\frac{d}{dr}-\frac{l}{r}\right] f_{lm}(r)\mathbf{Y}_{ll+1m}(\hat{\mathbf{r}})\\ &+\left[\frac{l}{2l+1}\right]^{1/2}\left[\frac{d}{dr}+\frac{l+1}{r}\right] f_{lm}(r)\mathbf{Y}_{ll-1m}(\hat{\mathbf{r}}),\end{aligned}$$

where the vector spherical harmonics are given by

$$\mathbf{Y}_{ll'm}(\hat{\mathbf{r}}) = \sum_{m'=-l'}^{l'} \sum_{m''=-1}^{1} C(l'1l|m'm''m)\, Y_{lm}(\hat{\mathbf{r}})\, \hat{\mathbf{e}}_{m''},$$

C is a Clebsch-Gordan coefficient and

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

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

REVISION HISTORY:

```
Created August 2003 (JKD)
```

4.1.126 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)
```

4.1.127 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)
```

4.1.128 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)
```

4.1.129 zflmconj (Source File: zflmconj.f90)

INTERFACE:

```
subroutine zflmconj(lmax,zflm1,zflm2)
```

INPUT/OUTPUT PARAMETERS:

```
lmax  : maximum angular momentum (in,integer)
zflm1 : coefficients of input complex spherical harmonic expansion
        (in,complex((lmax+1)**2)))
zflm2 : coefficients of output complex spherical harmonic expansion
        (out,complex((lmax+1)**2)))
```

DESCRIPTION:

Returns the complex conjugate of a function expanded in spherical harmonics. In other words, given the input function coefficients z_{lm}, the routine returns $z'_{lm} = (-1)^m z^*_{l-m}$ so that

$$\sum_{lm} z'_{lm} Y_{lm}(\theta, \phi) = \left(\sum_{lm} z_{lm} Y_{lm}(\theta, \phi) \right)^*$$

for all (θ, ϕ). Note that `zflm1` and `zflm2` can refer to the same array.

REVISION HISTORY:

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

4.1.130 rotzflm (Source File: rotzflm.f90)

INTERFACE:

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

INPUT/OUTPUT PARAMETERS:

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

DESCRIPTION:

Rotates a set of functions

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

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

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

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

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

where the sum runs over all i which make the factorial arguments non-negative. For improper rotations, i.e. those which are a combination of a rotation and inversion, the rotation is first made proper with $R \to -R$ and D is modified with $D^l_{mm'} \to (-1)^l D^l_{mm'}$. The routine may be used in-place, in other words, `zflm1` and `zflm2` can refer to the same array.

REVISION HISTORY:

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

4.1.131 polynom (Source File: polynom.f90)

INTERFACE:

```
real(8) function polynom(m,np,xa,ya,c,x)
```

INPUT/OUTPUT PARAMETERS:

```
m  : order of derivative (in,integer)
np : number of points to fit (in,integer)
xa : abscissa array (in,real(np))
ya : ordinate array (in,real(np))
c  : work array (out,real(np))
x  : evaluation abscissa (in,real)
```

DESCRIPTION:

Fits a polynomial of order $n_p - 1$ to a set of n_p points. If $m \geq 0$ the function returns the mth derviative of the polynomial at x, while for $m < 0$ the integral of the polynomial from the first point in the array to x is returned.

REVISION HISTORY:

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

4.1.132 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

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

REVISION HISTORY:

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

4.1.133 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)
```

4.1.134 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)
```

4.1.135 sdelta_mp (Source File: sdelta_mp.f90)

INTERFACE:

```
real(8) function sdelta_mp(n,x)
```

INPUT/OUTPUT PARAMETERS:

```
n : order (in,integer)
x : real argument (in,real)
```

DESCRIPTION:

Returns the smooth approximation to the Dirac delta function of order N given by Methfessel and Paxton [Phys. Rev. B 40, 3616 (1989)],

$$\tilde{\delta}(x)=\sum_{i=0}^{N}\frac{(-1)^i}{i!4^n\sqrt{\pi}}H_{2i}(x)e^{-x^2},$$

where H_j is the jth-order Hermite polynomial. This function has the property

$$\int_{-\infty}^{\infty}\tilde{\delta}(x)P(x)=P(0),$$

where $P(x)$ is any polynomial of degree $2N+1$ or less. The case $N=0$ corresponds to Gaussian smearing. This procedure is numerically stable and accurate to near machine precision for $N\leq 10$.

REVISION HISTORY:

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

4.1.136 stheta_mp (Source File: stheta_mp.f90)

INTERFACE:

```
real(8) function stheta_mp(n,x)
```

INPUT/OUTPUT PARAMETERS:

```
n : order (in,integer)
x : real argument (in,real)
```

DESCRIPTION:

Returns the smooth approximation to the Heaviside step function of order N given by Methfessel and Paxton [Phys. Rev. B 40, 3616 (1989)]

$$\tilde{\Theta}(x)=1-S_N(x)$$

where

$$S_N(x) = S_0(x) + \sum_{i=1}^{N} \frac{(-1)^i}{i!4^n\sqrt{\pi}} H_{2i-1}(x)e^{-x^2},$$
$$S_0(x) = \frac{1}{2}(1 - \mathrm{erf}(x))$$

and H_j is the jth-order Hermite polynomial. This procedure is numerically stable and accurate to near machine precision for $N \leq 10$.

REVISION HISTORY:

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

4.1.137 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)
```

4.1.138 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)
```

4.1.139 rdiracint (Source File: rdiracint.f90)

INTERFACE:

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

INPUT/OUTPUT PARAMETERS:

```
     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 mth 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 mth 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)
```

4.1.140 rdiracdme (Source File: rdiracdme.f90)

INTERFACE:

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

INPUT/OUTPUT PARAMETERS:

```
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 mth energy derivative of the radial Dirac equation using the routine `rdiracint`.

REVISION HISTORY:

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

4.1.141 rdirac (Source File: rdirac.f90)

INTERFACE:

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

INPUT/OUTPUT PARAMETERS:

```
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)
```

4.1.142 rschrodint (Source File: rschrodint.f90)

INTERFACE:

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

INPUT/OUTPUT PARAMETERS:

```
m   : order of energy derivative (in,integer)
l   : angular momentum quantum number (in,integer)
e   : energy (in,real)
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 mth 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)

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

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

$$\begin{aligned} P_l &= rg_l \\ Q_l &= \frac{r}{2M}\frac{dg_l}{dr}, \end{aligned}$$

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)
```

4.1.143 rschroddme (Source File: rschroddme.f90)

INTERFACE:

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

INPUT/OUTPUT PARAMETERS:

```
m   : order of energy derivative (in,integer)
l   : angular momentum quantum number (in,integer)
e   : energy (in,real)
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 mth energy derivative of the scalar relativistic radial Schrödinger equation using the routine `rschrodint`.

REVISION HISTORY:

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

4.1.144 rschrodapp (Source File: rschrodapp.f90)

INTERFACE:

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

INPUT/OUTPUT PARAMETERS:

```
     l   : angular momentum quantum number (in,integer)
     nr  : number of radial mesh points (in,integer)
     r   : radial mesh (in,real(nr))
     vr  : potential on radial mesh (in,real(nr))
     p0  : m th energy derivative of P (in,real(nr))
     q0  : m th energy derivative of Q (in,real(nr))
     q1  : radial derivative of q0 (in,real(nr))
     hp0 : H applied to P (out,real(nr))
```

DESCRIPTION:

Applies the scalar relativistic radial Hamiltonian, H, to a radial wavefunction, P_l. This is an approximation since we assume P_l is a scalar wavefunction, normalisable to unity. A Hamiltonian which satisfies $HP_l = EP_l$ is given implicitly by

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

where V is the external potential, $M = 1 - V/2c^2$ and Q_l is obtained from integrating the coupled scalar relativistic equations. See the routine `rschrodint` for further details.

REVISION HISTORY:

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

4.1.145 rschrod (Source File: rschrod.f90)

INTERFACE:

```
 subroutine rschrod(n,l,np,nr,r,vr,eval,p0,q0)
```

INPUT/OUTPUT PARAMETERS:

```
     n    : principal quantum number (in,integer)
     l    : quantum number l (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)
     p0   : P component of the radial wavefunction (out,real(nr))
     q0   : Q component of the radial wavefunction (out,real(nr))
```

DESCRIPTION:

Finds the solution to the scalar relativistic radial Schrödinger equation for a given potential $v(r)$ and quantum numbers n 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 `P0` and `Q0` represent the radial functions multiplied by r. See routine `rschrodint`.

REVISION HISTORY:

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

4.1.146 reciplat (Source File: reciplat.f90)

INTERFACE:

```
 subroutine reciplat(avec,bvec,omega)
```

INPUT/OUTPUT PARAMETERS:

```
     avec  : real-space lattice vectors stored column-wise (in,real(3,3))
     bvec  : reciprocal lattice vectors stored column-wise (out,real(3,3))
     omega : unit cell volume (out,real)
```

DESCRIPTION:

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

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

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)
```

4.1.147 findsymlat (Source File: findsymlat.f90)

INTERFACE:

```
 subroutine findsymlat(eps,avec,tpolar,plrvl,nsymlat,symlat)
```

INPUT/OUTPUT PARAMETERS:

```
eps     : zero vector tolerance (in,real)
avec    : lattice vectors stored column-wise (in,real(3,3))
tpolar  : .true. if plrvl is a polar vector, .false. if it is an axial
          vector (in,logical)
plrvl   : polarisation vector (in,real(3))
nsymlat : number of lattice point group symmetries found (out,integer)
symlat  : lattice point group symmetries found (out,integer(3,3,48))
```

DESCRIPTION:

Finds the point group symmetries which leave the Bravais lattice and a polarisation vector, $\mathbf{p}$, invariant. If `tpolar` is `.true.` then $\mathbf{p}$ is assumed to be a polar vector, otherwise it is axial and invariant under inversion. Let A be the matrix consisting of lattice vectors in columns, then

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

is the metric tensor. Any 3×3 matrix S with elements -1, 0 or 1 is a symmetry of the lattice if the following conditions hold:

$$S^{\mathrm{T}} g S = g,$$
$$\kappa S \mathbf{p} = \mathbf{p},$$

where $\kappa = 1$ if $\mathbf{p}$ is polar, and $\kappa = |S|$ if $\mathbf{p}$ is axial. Vectors which are within a distance ϵ of each other are considered to be equal. The first matrix in the set returned is the identity.

REVISION HISTORY:

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

4.1.148 findsymcrys (Source File: findsymcrys.f90)

INTERFACE:

```
subroutine findsymcrys(tsymctr,eps,nspecies,natoms,nsymlat,symlat,ld,atposl, &
 tpolar,plrvl,nsymcrys,symcrys)
```

INPUT/OUTPUT PARAMETERS:

```
tsymctr  : .true. if the crystal should be shifted to the optimal symmetry
           center (in,logical)
eps      : zero vector tolerance (in,real)
nspecies : number of species (in,integer)
natoms   : number atoms for each species (in,integer(nspecies))
nsymlat  : number of lattice point group symmetries (in,integer)
symlat   : lattice point group symmetries (in,integer(3,3,48))
ld       : leading dimension (in,integer)
atposl   : atomic positions in lattice coordinates
```

```
               (inout,real(3,ld,nspecies))
    tpolar   : .true. if plrvl are polar vectors, .false. if they are axial
               vectors (in,logical)
    plrvl    : polarisation vector at each atom site in lattice coordinates
               (in,real(3,ld,nspecies))
    nsymcrys : number of crystal point group symmetries found (out,integer)
    symcrys  : crystal point group symmetries found (out,integer(3,3,48))
```

DESCRIPTION:

Finds the largest number of point group symmetries which leave the crystal structure, including atomic positions and polarisation vectors at each site, invariant. If `tpolar` is `.true.` then the polarisation vectors are taken to be polar, otherwise they are axial and invariant under inversion. All atomic positions as well as the mid-points between two and three atoms are checked as possible symmetry centers. The atomic positions are then shifted so that the new symmetry center lies at $(0, 0, 0)$. The routine requires the symmetries which leave the Bravais lattice invariant. See routine `findsymcrysctr`.

REVISION HISTORY:

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

4.1.149 findsymcrysctr (Source File: findsymcrysctr.f90)

INTERFACE:

```
 subroutine findsymcrysctr(eps,nspecies,natoms,nsymlat,symlat,ld,atposl,tpolar, &
  plrvl,ctrvl,nsymcrys,symcrys)
```

INPUT/OUTPUT PARAMETERS:

```
    eps      : zero vector tolerance (in,real)
    nspecies : number of species (in,integer)
    natoms   : number atoms for each species (in,integer(nspecies))
    nsymlat  : number of lattice point group symmetries (in,integer)
    symlat   : lattice point group symmetries (in,integer(3,3,48))
    ld       : leading dimension (in,integer)
    atposl   : atomic positions in lattice coordinates (in,real(3,ld,nspecies))
    tpolar   : .true. if plrvl are polar vectors, .false. if they are axial
               vectors (in,logical)
    plrvl    : polarisation vector at each atom site in lattice coordinates
               (in,real(3,ld,nspecies))
    ctrvl    : symmetry center in lattice coordinates (in,real(3))
    nsymcrys : number of crystal point group symmetries found (out,integer)
    symcrys  : crystal point group symmetries found (out,integer(3,3,48))
```

DESCRIPTION:

For a given symmetry center, this routine finds those point group symmetries which leave the crystal structure, including atomic positions and polarisation vectors at each site, invariant. If `tpolar` is `.true.` then the polarisation vectors are taken to be polar, otherwise they are axial and invariant under inversion. The routine requires the symmetries which leave the Bravais lattice invariant. Vectors which are within a distance ϵ of each other are considered to be equal.

REVISION HISTORY:

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

4.1.150 findeqatoms (Source File: findeqatoms.f90)

INTERFACE:

```
subroutine findeqatoms(eps,nspecies,natoms,nsymlat,symlat,ld1,atposl,tpolar, &
 plrvl,ld2,nsymeqat,symeqat,tvleqat)
```

INPUT/OUTPUT PARAMETERS:

```
eps      : zero vector tolerance (in,real)
nspecies : number of species (in,integer)
natoms   : number atoms for each species (in,integer(nspecies))
nsymlat  : number of lattice point group symmetries (in,integer)
symlat   : lattice point group symmetries (in,integer(3,3,48))
ld1      : leading dimension 1 (in,integer)
atposl   : atomic positions in lattice coordinates (in,real(3,ld1,nspecies))
tpolar   : .true. if plrvl are polar vectors, .false. if they are axial
           vectors (in,logical)
plrvl    : polarisation vector at each atom site in lattice coordinates
           (in,real(3,ld1,nspecies))
ld2      : leading dimension 2 (in,integer)
nsymeqat : number of symmetries which map the equivalent atoms
           (out,integer(ld2,ld2,nspecies))
symeqat  : symmetries in symlat which map the equivalent atoms
           (out,integer(48,ld2,ld2,nspecies))
tvleqat  : translation vectors in lattice coordinates corresponding to each
           symeqat applied about origin (out,real(3,48,ld2,ld2,nspecies))
```

DESCRIPTION:

This routine finds equivalent atoms and related symmetries. Let $\{(\mathbf{r}_i, \mathbf{p}_i) : i = 1, 2, \ldots\}$ be the set of all atomic positions and polarisation vectors in the crystal. If `tpolar` is `.true.` then the polarisation vectors are taken to be polar, otherwise they are axial and invariant under inversion. Atom β is equivalent to atom α if they are of the same species,

$$(\mathbf{r}_\alpha, \mathbf{p}_\alpha) = (S\mathbf{r}_\beta + \mathbf{t}, \kappa S\mathbf{p}_\beta)$$

and

$$\{(\mathbf{r}_i, \mathbf{p}_i) : i = 1, 2, \ldots\} = \{(S\mathbf{r}_i + \mathbf{t}, \kappa S\mathbf{p}_i) : i = 1, 2, \ldots\},$$

for some lattice point symmetry S and translation $\mathbf{t}$, where $\kappa = 1$ if $\mathbf{p}$ is polar, and $\kappa = |S|$ if $\mathbf{p}$ is axial. Vectors which are within a distance ϵ of each other are considered to be equal.

REVISION HISTORY:

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

4.1.151 genkpts (Source File: genkpts.f90)

INTERFACE:

```
subroutine genkpts(eps,nsym,sym,ngridk,bvec,vkloff,nkpt,ikmap,ivk,vkl,vkc,wkpt)
```

INPUT/OUTPUT PARAMETERS:

```
eps    : zero vector tolerance (in,real)
nsym   : number of symmetry matrices (in,integer)
sym    : symmetry matrices (in,integer(3,3,nsym))
ngridk : k-point grid size (in,integer(3))
bvec   : reciprocal lattice vectors (in,real(3,3))
vkloff : offset of k-point grid in lattice coordinates (in,real(3))
nkpt   : total number of k-points (out,integer)
ikmap  : map from grid to k-point set
         (out,integer(0:ngridk(1)-1,0:ngridk(2)-1,0:ngridk(3)-1))
ivk    : integer coordinates of the k-points
         (out,integer(3,ngridk(1)*ngridk(2)*ngridk(3)))
vkl    : lattice coordinates of each k-point
         (out,real(3,ngridk(1)*ngridk(2)*ngridk(3)))
vkc    : Cartesian coordinates of each k-point
         (out,real(3,ngridk(1)*ngridk(2)*ngridk(3)))
wkpt   : weights of each k-point (out,real(ngridk(1)*ngridk(2)*ngridk(3)))
```

DESCRIPTION:

This routine is used for generating k-point or q-point sets. The set is reduced with the symmetries in the array `sym`. If no reduction is required then `nsym` should be set to 1. In lattice coordinates the vectors are given by

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

where i_j runs from 0 to $n_j - 1$ and $0 \leq \mathbf{v}_{{\rm off};j} < 1$ for $j = 1, 2, 3$. The array `ikmap` contains the map from the k-point integer coordinates to the reduced index.

REVISION HISTORY:

```
Created August 2002 (JKD)
```

4.1.152 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)
```

4.1.153 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)
```

4.1.154 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(n))
cf : cubic spline coefficients (out,real(3,n))
```

DESCRIPTION:

Calculates the coefficients of a clamped 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_{ij} are determined such that

$$y_i(x) = f_i + c_{i1}(x - x_i) + c_{i2}(x - x_i)^2 + c_{i3}(x - x_i)^3,$$

is the interpolating function for $x \in [x_i, x_{i+1})$.

REVISION HISTORY:

```
Created October 2004 (JKD)
Improved speed and accuracy, April 2006 (JKD)
```

4.1.155 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)
```

4.1.156 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)
```

4.1.157 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. Coefficients of large angular momenta (greater than `lmaxinr`) are set to zero on the inner part of the muffin-tin. See routines `rfinterp` and `spline`.

REVISION HISTORY:

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

4.1.158 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 (out,real(3,n))
```

DESCRIPTION:

Given function f defined on a set of points x_i then if $m \geq 0$ this routine computes the mth derivative of f at each point. If $m < 0$ the anti-derivative of f given by

$$g(x_i) = \int_{x_1}^{x_i} f(x)\,dx$$

is calculated. If $m = -1$ then an accurate integral is computed by fitting the function to a clamped cubic spline. See routine `spline`.

REVISION HISTORY:

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

4.1.159 mixer (Source File: mixer.f90)

INTERFACE:

```
subroutine mixer(init,beta0,betamax,n,nu,mu,beta,f,d)
```

INPUT/OUTPUT PARAMETERS:

```
init    : .true. if the mixer should be intitialised (inout,logical)
beta0   : default 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 μ^i and output vector ν^i of the ith self-consistent loop, this routine generates the next input vector to the loop using an adaptive mixing scheme. The jth component of the output vector is mixed with a fraction of the same component of the input vector:

$$\mu_j^{i+1} = \beta_j^i \nu_j^i + (1 - \beta_j^i)\mu_j^i,$$

where β_j^i is set to β^0 at initialisation and increased by the same amount if $f_j^i \equiv \nu_j^i - \mu_j^i$ does not change sign between loops. If f_j^i does change sign, then β_j^i is reset to β^0. Note that the array `nu` serves for both input and ouput, and the arrays `mu`, `beta` and `f` are used internally and should not be changed between calls. The routine must be initialised before the first iteration and is thread-safe so long as each thread has its own independent set of storage variables. Complex arrays may be passed as real arrays with n doubled.

REVISION HISTORY:

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

4.1.160 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)
```

4.1.161 rotaxang (Source File: rotaxang.f90)

INTERFACE:

```
subroutine rotaxang(eps,r,det,v,th)
```

INPUT/OUTPUT PARAMETERS:

```
eps : zero vector tolerance (in,real)
r   : rotation matrix (in,real(3,3))
det : matrix determinant (out,real)
v   : normalised axis vector (out,real(3))
th  : rotation angle (out,real)
```

DESCRIPTION:

Given a rotation matrix

$$R(\hat{\mathbf{v}},\theta)=\begin{pmatrix}\cos\theta+x^2(1-\cos\theta) & xy(1-\cos\theta)+z\sin\theta & xz(1-\cos\theta)-y\sin\theta\\ xy(1-\cos\theta)-z\sin\theta & \cos\theta+y^2(1-\cos\theta) & yz(1-\cos\theta)+x\sin\theta\\ xz(1-\cos\theta)+y\sin\theta & yz(1-\cos\theta)-x\sin\theta & \cos\theta+z^2(1-\cos\theta)\end{pmatrix},$$

this routine determines the axis of rotation $\hat{\mathbf{v}}$ and the angle of rotation θ. If R corresponds to an improper rotation then only the proper part is used and `det` is set to -1.

REVISION HISTORY:

```
Created Decmeber 2006 (JKD)
```

4.1.162 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), Phys. Rev. Lett. 45, 566 (1980)].

REVISION HISTORY:

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

4.1.163 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), Phys. Rev. Lett. 45, 566 (1980)].

REVISION HISTORY:

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

4.1.164 xc_pbe (Source File: xc_pbe.f90)

INTERFACE:

```
subroutine xc_pbe(n,kappa,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)
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); 78, 1396(E) (1997)]. The parameter κ, which controls the large-gradient limit, can be set to 0.804 or 1.245 corresponding to the value in the original article or the revised version of Y. Zhang and W. Yang [Phys. Rev. Lett. 80, 890 (1998)].

REVISION HISTORY:

```
    Modified routines written by K. Burke, October 2004 (JKD)
```

4.1.165 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
```

4.1.166 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 of Armiento-Mattsson 05 functional (to be published) for one point.

REVISION HISTORY:

```
Created April 2005 (RAR)
```

4.1.167 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)
```

4.1.168 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), Phys. Rev. Lett. 45, 566 (1980)]. (Clean room implementation from paper)

REVISION HISTORY:

```
    Created April 2005 (RAR)
```

4.1.169 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, 329-359 (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)
```

4.1.170 xc_xalpha (Source File: xc_xalpha.f90)

INTERFACE:

```
subroutine xc_xalpha(n,rho,exc,vxc)
```

INPUT/OUTPUT PARAMETERS:

```
n   : number of density points (in,integer)
rho : charge density (in,real(n))
exc : exchange-correlation energy density (out,real(n))
vxc : exchange-correlation potential (out,real(n))
```

DESCRIPTION:

X_α approximation to the exchange-correlation potential and energy density [J. C. Slater, Phys. Rev. 81, 385 (1951)].

REVISION HISTORY:

```
Modified an ABINIT routine, September 2006 (JKD)
```

4.1.171 vnlwfv (Source File: vnlwfv.f90)

INTERFACE:

```
subroutine vnlwfv(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. These wavefunctions are used for calculating the non-local Coulomb matrix elements.

REVISION HISTORY:

```
    Created November 2004 (Sharma)
```

4.1.172 vnlrho (Source File: vnlrho.f90)

INTERFACE:

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

USES:

```
 use modmain
```

INPUT/OUTPUT PARAMETERS:

```
    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
             (out,complex(lmmaxvr,nrcmtmax,natmtot))
    zrhoir : interstitial charge density (out,complex(ngrtot))
```

DESCRIPTION:

Calculates the complex overlap charge density from two input wavefunctions:

$$\rho(\mathbf{r}) \equiv \Psi_1^*(\mathbf{r})\Psi_2(\mathbf{r}).$$

Note that the muffin-tin wavefunctions are provided in spherical coordinates and the returned density is in terms of spherical harmonic coefficients. See also routine `vnlrhomt`.

REVISION HISTORY:

```
    Created November 2004 (Sharma)
```

4.1.173 vnlrhomt (Source File: vnlrhomt.f90)

INTERFACE:

```
subroutine vnlrhomt(is,wfmt1,wfmt2,zrhomt)
```

USES:

```
use modmain
```

INPUT/OUTPUT PARAMETERS:

```
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
         (out,complex(lmmaxvr,nrcmtmax))
```

DESCRIPTION:

Calculates the complex overlap density in a single muffin-tin from two input wavefunctions. See routine `vnlrho`.

REVISION HISTORY:

```
Created November 2004 (Sharma)
```

4.1.174 genwiq2 (Source File: genwiq2.f90)

INTERFACE:

```
subroutine genwiq2
```

USES:

```
use modmain
```

DESCRIPTION:

The Fock matrix elements

$$V^{\rm NL}_{ij{\bf k}}\equiv\sum_{l{\bf k}'}\int\frac{\phi^*_{i{\bf k}}({\bf r})\phi_{l{\bf k}'}({\bf r})\phi^*_{l{\bf k}'}({\bf r}')\phi_{j{\bf k}}({\bf r}')}{|{\bf r}-{\bf r}'|}\,d{\bf r}\,d{\bf r}'$$

contain a divergent term in the sum over ${\bf k}'$ which behaves as $1/q^2$, where ${\bf q}\equiv{\bf k}-{\bf k}'$ 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

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

where the integral is over the small parallelepiped centered on $\mathbf{q}_i$, so that integrals over the first Brillouin zone of the form

$$I = \int_{\rm BZ} \frac{f(\mathbf{q})}{q^2}\, d\mathbf{q}\,,$$

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)
```

4.1.175 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

$$D^{\mathbf{q}}_{ij} \rightarrow D^{\mathbf{q}}_{ij} - \sum_{k=1}^{3} \omega^0_k v^0_{k;i} v^0_{k;j}$$

for all $\mathbf{q}$, where ω^0_k is the kth eigenvalue of the $\mathbf{q}=0$ dynamical matrix and $v^0_{k;i}$ the ith component of its eigenvector. The eigenvalues are assumed to be arranged in ascending order. This ensures that the $\mathbf{q}=0$ dynamical matrix has 3 zero eigenvalues, which the uncorrected matrix may not have due to the finite exchange-correlation grid.

REVISION HISTORY:

```
Created May 2005 (JKD)
```