RSPt Manual, v1.0

John Wills, Per Andersson, Torbjörn Björkman, Igor Di Marco
Oscar Grånäs, Patrik Thunström, Yaroslav Kvashnin

July 31, 2023

Contents

1 What is RSPt?

What is RSPt? RSPt is a Open Source project and a code for band structure calculations. RSPt (Relativistic Spin Polarized (test)) is very robust and flexible and can be used to calculate the band structure and total energy for all elements, and combinations thereof, over a wide range of volumes and structures. The Full-Potential Linear Muffin-Tin Orbital (FP-LMTO) method allows for very small basis sets and fast calculations. Compared to LMTO-ASA there are no restrictions on the symmetry of the potential in FP-LMTO. RSPt allows for multiple energy sets (i.e. valence and semi-core states) with the same angular quantum numbers but different principal quantum number.

The code can be used for spin-polarized and/or spin-orbit calculations with several LDA and GGA functionals implemented. The symmetry generator can be used to automatically create input files which can be edited if necessary. No material library is needed as all input data is created on the fly.

The flexibility and stability of the basis set come at a price. The FP-LMTO basis set is site dependent and because of that it is very difficult to implement linear response and the stress tensor. We are working on that but as for now structural relaxations and phonons must be calculated in an indirect way using e.g. structure distortions and diagonalizing of the dynamical matrix, respectively.

This manual is not a short list of keywords, but rather a document where we try to convey many years of experience of using RSPt. The program is very flexible with a huge number of parameters that the user can change for good and bad. It was developed and used for many years by a small set of users who passed on the knowledge on how to use the program from scientist to scientist. You can, if you know how, calculate almost any crystal material with RSPt but with all that freedom to tweak the finer details and parameters of the methods used comes a risk of doing things horribly wrong. This perhaps over talkative text is our way of trying to show you the right direction.

This manual is a living document. We try to add more information explaining old and new functionality. If you have a question that is not answered in the manual you can always ask the Forum at http://www.rspt.net/. Section 2 is highly technical and perhaps a bit tough for the novice solid state scientist but reading it really helps you understand what is going on in the program and what the input really means.

2 The machinery behind FP-LMTO

This section describes a particular implementation of a full-potential electronic structure method using Linear Muffin-Tin Orbitals (LMTO's) [1, 2] as basis functions. There have been several "FP-LMTO" implementations [3, 4, 5, 6, 7]. There are many aspects to an electronic structure method. This paper is focused on those aspects which enable a full potential treatment. Relatively small details pertaining to full-potential methods will be discussed while larger details having to do with, for example, relativity will not be.

The emphasis of a variational full-potential method is somewhat different from that of a method such as the LMTO-ASA method. The emphasis of the former is on the completeness of the basis while in the latter it is in the phys-

ical content (and interpretability) of the basis. These concepts are, of course, intimately related, but the emphasis is different.

2.1 Notation

Papers on electronic structure methods unavoidably carry a high overhead in functional symbols and indices. It is simplest to define here, without motivation, the special symbols and functions that will be used in this paper, for future reference. These special functions (although not necessarily the symbols used here) have been used extensively in LMTO documentation and are largely due to Andersen.[1]

2.1.1 Spherical harmonics:

$$\mathcal{Y}_{\ell m}(\hat{\vec{r}}) \equiv \mathrm{i}^{\ell} Y_{\ell m}(\hat{\vec{r}}) \tag{1}$$

$$C_{\ell m}(\hat{\vec{r}}) \equiv \sqrt{\frac{4\pi}{2\ell+1}}\, Y_{\ell m}(\hat{\vec{r}}) \tag{2}$$

$$\mathcal{C}_{\ell m}(\hat{\vec{r}}) \equiv \mathrm{i}^{\ell} C_{\ell m}(\hat{\vec{r}}) \tag{3}$$

where Y is a spherical harmonic.[8]

2.1.2 Bessel functions:

$$\mathcal{K}_{\ell}(\kappa, r) \equiv -\kappa^{\ell+1} \begin{cases} n_\ell(\kappa r) - \mathrm{i} j_\ell(\kappa r) & \kappa^2 < 0 \\ n_\ell(\kappa r) & \kappa^2 > 0 \end{cases} \tag{4}$$

$$\mathcal{K}_{L}(\kappa, \vec{r}) \equiv \mathcal{K}_{\ell}(\kappa, r)\mathcal{Y}_L(\hat{\vec{r}}) \tag{5}$$

$$\mathcal{J}_{\ell}(\kappa, r) \equiv j_\ell(\kappa r)/\kappa^{\ell} \tag{6}$$

$$\mathcal{J}_{L}(\kappa, \vec{r}) \equiv \mathcal{J}_{\ell}(\kappa, r)\mathcal{Y}_L(\hat{\vec{r}}) \tag{7}$$

where L denotes ℓm and n_ℓ and j_ℓ are spherical Neumann and Bessel functions, respectively.

2.1.3 Geometry:

For computational purposes, the crystal is divided into non-overlapping spheres surrounding atomic sites (*muffin-tin spheres*) where the charge density and potential vary rapidly and the *interstitial* region between the spheres, where the charge density and potential vary slowly. This is the *muffin-tin* geometry used as an idealized potential and charge density in early electronic structure methods (KKR and APW). Here, the division is a computational one, and does not restrict the final shape of the charge density or potential. In the muffin-tin spheres, the basis functions, electron density, and potential are expanded in spherical waves; in the interstitial region, the basis functions, electron density, and potential are expanded in Fourier series.

There are many relevant considerations in choosing muffin-tin radii. Assuming all expansions are taken to convergence, the density and potential depend on the muffin-tin radii only through the dependence of basis functions on

the radii. As discussed below, basis functions have a different functional form inside the muffin-tin spheres, and the choice of muffin-tin radius affects this crossover. Hence, assuming the Hamiltonian is the same inside and outside the spheres (the treatment of relativity may affect this as discussed below), the muffin-tin radii are variational parameters and the optimum choice minimizes the total energy. If the basis is large enough however (suitably complete within and without the spheres), the energy is insensitive to the choice of radii. A reasonable choice results from choosing radii that are within both the minimum in charge density and the maximum in potential along a line between nearest neighbors. Relativistic effects are usually taken into account only in the muffin-tin spheres, in which case the Hamiltonian depends on the radii; hence when relativistic effects are important, the radii are not variational parameters.

In what follows, lattice positions are vectors $\vec{R} = R\vec{n}$, integer multiples of a basis R. Atomic positions in the unit cell are denoted by $\vec{\tau}$. A set of atomic positions invariant under the point group of the lattice are said to be of the same symmetry type, t. Similarly, in the reciprocal lattice, vectors are $\vec{g} = G\vec{n}$ for the reciprocal basis $G = 2\pi R^{-T}$. Brillouin zone (or reciprocal unit cell) vectors are denoted by $\vec{k}$.

2.1.4 Symmetric functions:

Within the muffin-tin region, functions invariant are expressed in harmonic series. If $f(\vec{r})$ is such a function, at site τ

$$f(\vec{r})\Big|_{r_\tau < s_\tau} = \sum_h f_{ht}(r_\tau) D_{ht}(\mathcal{D}_\tau \hat{\vec{r}}_\tau) \tag{8a}$$

$$D_{ht}(\hat{\vec{r}}) = \sum_m \alpha_{ht}(m) \mathcal{C}_{\ell_h m}(\hat{\vec{r}}) \tag{8b}$$

In Equation (8a), $\mathcal{D}_\tau$ is a transformation to a coordinate system local to site τ; the local coordinates of sites of the same type are related by an element of the crystal point group that takes one site into another. Expressed in this way, the functional form of D_{ht} (Equation (8b)) depends only on symmetry type.

In the interstitial region, symmetric functions are expressed in Fourier series:

$$f(\vec{r})\Big|_{r\in\mathcal{I}} = \sum_{\mathcal{S}} f(\mathcal{S}) D_{\mathcal{S}}(\vec{r}) \tag{9a}$$

$$D_{\mathcal{S}}(\vec{r}) = \sum_{g\in\mathcal{S}} e^{i\vec{g}\cdot\vec{r}} \tag{9b}$$

The sum in Equation (9a) is over symmetry stars $\mathcal{S}$ of the reciprocal lattice.

2.2 Basis Set

2.2.1 Interstitial

In the interstitial region (symbolically $\mathcal{I}$) between the muffin-tin spheres, bases are Bloch sums of spherical Hankel or Neumann functions:

$$\psi_i(\vec{k},\vec{r})\Big|_{\vec{r}\in\mathcal{I}} = \sum_R e^{i\vec{k}\cdot\vec{R}} \mathcal{K}_{\ell_i}(\kappa_i, |\vec{r}-\vec{\tau}_i-\vec{R}|) \mathcal{Y}_{\ell_i m_i}\big(\mathcal{D}_{\tau_i}(\vec{r}-\vec{\tau}_i-\vec{R})\big) \tag{10}$$

The rotation $\mathcal{D}_\tau$ in (10) takes the argument into a coordinate system local to each site τ. The purpose of this will be made evident later. The function on the right hand side of Equation (10) is sometimes called the envelope function .

Notice the parameters, specifying a basis function, inherent in this definition. They are the site τ in the unit cell on which the spherical wave is based, the angular momentum parameters ℓ and m of the spherical wave with respect to its parent cell, and the kinetic energy κ^2 of the basis in the interstitial region. The angular momentum parameters specifying the basis set are chosen to represent the atomic states from which crystal eigenstates are derived. In the LMTO-ASA, it is usual to include ℓ bases one higher than the highest relevant band. In the method described here, this is rarely necessary, possibly because of the multiplicity of bases with the same angular momentum parameters. It is usual to use "multiple κ" basis sets, having all parameters except the tail parameter the same.

There appears to be no simple algorithm for choosing a good set of interstitial kinetic energy parameters. Schemes such as bracketing the relevant energy spectrum have been proposed.[7] The optimum set would minimize the total energy. This can be done but is time consuming even for relatively simple systems. It seems, however that parameter sets obtained in this way for simple systems in representative configurations can give good results when used for related systems over a broad pressure range. Thus good sets are arrived at through some experimentation. The choice can be important as it's possible to pick a set of parameters that will give very bad results, and the parameter set used in any new calculation should be always checked for stability.

2.2.2 Muffin Tins

In the muffin-tin spheres , bases are linear combinations of spherical waves matching continuously and differentiably to the envelope function at the muffin-tin sphere. The envelope function $\mathcal{K}$ may be expanded in a series of spherical Bessel functions about any site except its center. A basis function on a muffin-tin sphere in the unit cell at $\vec{R} = 0$ is therefore

$$\begin{aligned}
\psi_i(\vec{k},\vec{r})\Big|_{r_\tau=s_\tau} = \sum_R e^{i\vec{k}\cdot\vec{R}} \sum_L \mathcal{Y}_L(\mathcal{D}_\tau\hat{\vec{r}}_\tau)\Big(& \quad \mathcal{K}_\ell(\kappa_i,s_\tau)\delta(R,0)\delta(\tau,\tau_i)\delta(L,L_i) \\
& + \quad \mathcal{J}_L(\kappa,s_\tau)B_{L,L_i}(\kappa_i,\vec{\tau}-\vec{\tau}'-\vec{R})\Big) \\
= \sum_L \mathcal{Y}_L(\mathcal{D}_\tau\hat{\vec{r}}_\tau)\Big(& \quad \mathcal{K}_\ell(\kappa_i,s_\tau)\delta(\tau,\tau_i)\delta(L,L_i) \\
& + \quad \mathcal{J}_L(\kappa,s_\tau)B_{L,L_i}(\kappa_i,\vec{\tau}-\vec{\tau}',\vec{k})\Big)
\end{aligned} \tag{11}$$

where $\vec{r}_\tau \equiv \vec{r} - \vec{\tau}$ and B is equivalent to the KKR structure constant . [9] The unitary transformation applied to B rotates components into site-local coordinates from the left and right.

Equation (11) is compactly expressed by defining a two-component row vector K so that

$$\mathrm{K}_\ell(\kappa,r) \quad = \quad (\mathcal{K}_\ell(\kappa,r),\mathcal{J}_\ell(\kappa,r)) \tag{12}$$

and a two component column vector S so that

$$\mathrm{S}_{L,L'}(\kappa,\vec{\tau}-\vec{\tau}',\vec{k}) \;=\; \begin{pmatrix} \delta(\tau,\tau')\delta(L,L') \\ B_{L,L'}(\kappa,\vec{\tau}-\vec{\tau}',\vec{k}) \end{pmatrix} \quad . \tag{13}$$

Then the value of a basis function on a muffin-tin boundary is expressed simply as

$$\psi_i(\vec{k},\vec{r})\Big|_{r_\tau=s_\tau} \;=\; \sum_L \mathcal{Y}_L(\mathcal{D}_\tau\hat{\vec{r}}_\tau)\mathrm{K}_\ell(\kappa_i,s_\tau)\mathrm{S}_{L,L_i}(\kappa_i,\vec{\tau}-\vec{\tau}',\vec{k}) \tag{14}$$

The radial part a basis function inside a muffin-tin sphere is a linear combination of atomic like functions ϕ and their energy derivatives $\dot{\phi}$ [1, 2] matching continuously and differentiably to the radial function K in Equation (14). Collecting ϕ and $\dot{\phi}$ in a row vector

$$\mathrm{U}(e,r) \equiv \Big(\phi(e,r),\dot{\phi}(e,r)\Big) \quad , \tag{15}$$

a simple case of this matching condition may be expressed as $\mathrm{U}(e,s)\Omega(e,\kappa) = \mathrm{K}(\kappa,s)$ and $\mathrm{U}'(e,s)\Omega(e,\kappa) = \mathrm{K}'(\kappa,s)$, where Ω is a matrix of order 2.

The use of these radial functions in the method described here is different than that used by most other methods, however. For the broadest utility, a basis set must be flexible enough to describe energy levels derived from atomic states having different principle quantum numbers but the same angular momentum quantum number. For example, describing the properties of elemental actinides at any pressure requires a basis with both $6p$ and $7p$ character. Similarly, an adequate calculation of the structural properties of transition metal oxides requires both semi-core and valence s and p states on the transition metal ions. The description of the evolution of core states from localized to itinerant under pressure also requires multiple principle quantum numbers per ℓ value. It is usual in LMTO-based methods to perform calculations for the eigenstates and eigenvalues of "semi-core" and valence states separately, using a different basis set, with a single set of energy parameters $\{e_\ell\}$, for each "energy panel". This approach fails when energy panels overlap, and has the disadvantage that the set of eigenvectors is not an orthogonal set. The problem of "ghost bands" also arises.[2]

In the method described here, bases corresponding to multiple principle quantum numbers are contained within a single, *fully hybridizing* basis set. This is accomplished simply by using functions ϕ and $\dot{\phi}$ calculated with energies $\{e_{n\ell}\}$ corresponding to different principal quantum numbers n to describe the radial dependence of a basis in the muffin-tin spheres. The Hamiltonian matrix for an actinide, for example, will have elements $\langle\psi_{6p}|H|\psi_{7p}\rangle$ and the overlap matrix elements $\langle\psi_{6p}|\psi_{7p}\rangle$, We may formally express the radial part of basis i in a muffin-tin sphere by the function $f(r) = \sum_n a_i(n\ell)\mathrm{U}(e_{n\ell},s)\Omega(e_{n\ell},\kappa_i)$ but in practice it is sufficient to restrict the coefficients by $a_i(n\ell) = \delta(n,n_i)$ so that the basis set (although not eigenvectors) will have pure principal quantum number "parentage". This method of expanding the energy range of a basis set has been used (and reported) extensively. Representative calculations in which this method was essential are described in Reference [10].

Thus another parameter specifying a basis function is the set of energy parameters $\{e_{t\ell}\}$ that will be used to calculate the radial basis functions $\phi_{t\ell}$ and

$\dot{\phi}_{t\ell}$ used to express the basis function in muffin-tin spheres of each symmetry type. A basis function in a muffin-tin sphere is therefore

$$\psi_i(\vec{k},\vec{r})\Big|_{r_\tau<s_t} = \sum_L^{\ell\le\ell_m} \mathrm{U}_{tL}(e_i,\mathcal{D}_\tau\vec{r}_\tau)\Omega_{t\ell}(e_i,\kappa_i)\mathrm{S}_{L,L_i}(\kappa_i,\vec{\tau}-\vec{\tau}',\vec{k}) \quad (16)$$

where e_i means "use the energy parameter $e_{n\ell}$ corresponding to the principal quantum number specified for basis i" and

$$\mathrm{U}_{tL}(e,\vec{r}) \equiv \mathcal{Y}_L(\hat{\vec{r}})\mathrm{U}_{t\ell}(e,r) \quad . \quad (17)$$

The necessary cutoff in angular momentum has now been made explicit. The 2×2 matrix Ω matches U to K continuously and differentiably at the muffin-tin radius. Specifically, Ω is specified by

$$\begin{pmatrix}\phi_{t\ell}(e,s_t) & \dot{\phi}_{t\ell}(e,s_t)\\ \phi'_{t\ell}(e,s_t) & \dot{\phi}'_{t\ell}(e,s_t)\end{pmatrix}\Omega_{t\ell}(e,\kappa) = \begin{pmatrix}\mathcal{K}_\ell(\kappa,s_t) & \mathcal{J}_\ell(\kappa,s_t)\\ \mathcal{K}'_\ell(\kappa,s_t) & \mathcal{J}'_\ell(\kappa,s_t)\end{pmatrix} \quad (18)$$

In principle, and as programmed, each $(\tau\ell\kappa)$ basis can use its own unique energy set. It is more usual to use a common energy set for a set of basis states giving rise to bands of similar energy within the scope of a particular calculation. The calculation of energies in an energy parameter set is discussed below.

A parameter introduced in (16) is the angular momentum cutoff ℓ_m. In most cases, a converged total energy is achieved with values $\ell_m \sim 6-8$. Note that since a basis set generally contains functions based on spherical waves with $\ell \le 3$, the KKR structure constant in (13) is rectangular.

2.3 Matrix Elements

2.3.1 Muffin-Tin Matrix Elements

The potential in a muffin-tin at $\vec{\tau}$ has an expansion in linear combinations of spherical harmonics invariant under that part of the point group leaving $\vec{\tau}$ invariant:

$$V(\vec{r})\Big|_{r_\tau<s_t} = \sum_h v_{ht}(r_\tau)D_{ht}\big(\mathcal{D}_\tau\hat{\vec{r}}_\tau\big) \quad (19a)$$

$$D_{ht}(\hat{\vec{r}}) = \sum_m \alpha_{ht}(m)\mathcal{C}_{\ell_h m}(\hat{\vec{r}}) \quad . \quad (19b)$$

The utility of referring bases and potentials in muffin-tin spheres to site-local coordinates is apparent in (19a). If the site local coordinates of sites are constructed so that $\mathcal{D}_{\tau'} = \mathcal{D}_\tau\mathcal{Q}^{-1}$ for some $\mathcal{Q}$ such that $\mathcal{Q}\vec{\tau} = \vec{\tau}'$, then the harmonic functions D_{ht} depend only on the symmetry type, rather than on each site. The normalization for the spherical harmonic in (19a) ($\mathcal{C} = \sqrt{4\pi/(2\ell+1)}\mathcal{Y}$) is chosen so that $v_{ht}(r)$ is the potential when $\ell_h = 0$.

Combining (16) and (19a), the potential matrix is

$$\begin{aligned}\big\langle\psi_i\big|V\big|\psi_j\big\rangle\Big|_{mt} = &\sum_\tau\sum_L \mathrm{S}^\dagger_{L,L_i}(\kappa_i,\vec{\tau}-\vec{\tau}_i,\vec{k}) \quad (20)\\ \times &\Big(\sum_h\sum_{L'}\Omega^T_{t\ell}(e_i,\kappa_i)\big\langle U^T_{t\ell}(e_i)\big|v_{ht}\big|U_{t\ell}(e_j)\big\rangle\Omega_{t\ell}(e_j,\kappa_j)\\ &\big\langle L\big|D_{ht}\big|L'\big\rangle\,\mathrm{S}_{L,L_j}(\kappa_j,\vec{\tau}-\vec{\tau}_j,\vec{k})\Big) \quad .\end{aligned}$$

The matrix element of the D_{ht} is a sum over Gaunt coefficients:

$$\begin{aligned} \langle L | D_{ht} | L' \rangle &= \sum_{m_h} \alpha_{ht}(m_h) \mathcal{G}\big(\ell',\, m';\, \ell,\, m;\, \ell_h,\, m_h\big) \\ \mathcal{G}\big(\ell',\, m';\, \ell,\, m;\, \ell_h,\, m_h\big) &= \int \mathcal{Y}_{\ell' m'} \mathcal{Y}^*_{\ell m} \mathcal{C}_{\ell_h m_h} \end{aligned}$$

In electronic structure methods using muffin-tin orbitals, the muffin-tin energy parameters $\{e_\ell$ are usually taken from "ℓ-projected average energies". With multiple energy sets, this is a reasonable choice provided that the basis set, which uses separate sets, gives rise to bands well separated in energy. The ℓ-projected charge, integrated over a muffin-tin sphere, is a sum over cross terms between energy sets

$$Q_\ell = \sum_{ij} Q_\ell(e_i, e_j)$$

and must be made diagonal in some approximation for the resulting energy- and ℓ-projected energies and charges to be representative.

Another criterion, particularly useful for states using different sets not well separated in energy or for states not having significant occupation is to maximize the completeness of the basis. To accomplish this, the energy parameter for the low energy state $e_\ell(1)$ can be set to a set of projected energy averages, and the energy parameters for the same ℓ in higher energy sets may be chosen so that the radial function has one more node and the same logarithmic derivative at the muffin-tin radius, hence

$$\int_0^s r^2 dr\ \phi_\ell(e_1, r)\phi_\ell(e_i, r) = 0 \quad , \quad i > 1 \quad . \tag{21}$$

Although this usually generates energy parameters out of the range of occupied states (since the logarithmic derivative of semi-core states is usually large in magnitude and negative), this choice seems to give a total energy close to the minimum with respect to this parameter. This is an example of the difference mentioned in the introduction in emphasis between an accurate "basis-set" method and a method motivated by a physical model.

The convergence of the harmonic expansion of the potential in a muffin-tin sphere (19a) depends, of course, on the basis, atomic constituents, and geometry. Using harmonics through $\ell_{h_{max}} = 6$ is usually sufficient, and it has never been necessary to go beyond $\ell_{h_{max}} = 8$.

2.3.2 Interstitial Matrix Elements

Overlap and Kinetic Energy: The interstitial overlap matrix can be easily obtained from an integral over the interstitial surface (the only non-zero contributions, in a crystal periodic in three dimensions, come from the surfaces of the muffin-tin spheres) and the kinetic energy is proportional to the overlap:

$$\begin{aligned} \int_{\mathcal{I}} \psi_i^\dagger(\vec{r}) \psi_j(\vec{r}) &= -(\kappa_j^2 - \kappa_i^2)^{-1} \int_{\mathcal{I}} \big(\psi_i^\dagger \nabla^2 \psi_j - (\nabla^2 \psi_i^\dagger)\psi_j\big) \\ &= (\kappa_j^2 - \kappa_i^2)^{-1} \sum_\tau s_t^2 \int d\Omega_\tau W(\psi_i^\dagger, \psi_j) \end{aligned} \tag{22}$$

where $W(f,g) = fg' - f'g$. Basis functions on muffin-tin spheres are given in (14), hence

$$\langle\psi_i|\psi_j\rangle\Big|_{\mathcal{I}} = \sum_\tau s_t^2 \sum_L \qquad \mathrm{S}^\dagger_{L,L_i}(\kappa_i, \vec{\tau}-\vec{\tau}_i, \vec{k}) \\ \times \qquad \frac{W\big(\mathrm{K}^T_\ell(\kappa_i, s_t), \mathrm{K}_\ell(\kappa_j, s_t)\big)}{\kappa_j^2 - \kappa_i^2} \mathrm{S}_{L,L_j}(\kappa_j, \vec{\tau}-\vec{\tau}_j, \vec{k}) \quad (23)$$

In the limit $\kappa_j^2 \to \kappa_i^2$, the evaluation of (22) requires the derivative with respect to κ^2 of the structure constant.

Potential Matrix Elements: The greatest difference between LMTO-based full-potential methods is in the way the matrix elements of the potential are calculated over the interstitial region . The method being described here uses a Fourier representation of basis functions and the interstitial potential to calculate these matrix elements. Other approaches for computing these elements are described in the literature. [4, 5]

A Fourier transform of the basis functions described in Section 2.2 would be too poorly convergent for practical use. However, the evaluation of the interstitial potential matrix requires only a correct treatment of basis functions and potential in the interstitial region. This degree of freedom can be used to design "pseudo basis-set", equal to the true basis in the interstitial region although not in the muffin-tin spheres, and have a Fourier transform which converges rapidly enough for practical use. We define this pseudo basis set by

$$\tilde{\psi}_i(\vec{k}, \vec{r})\Big|_{\vec{r}\in\mathcal{I}} = \sum_R e^{i\vec{k}\cdot\vec{R}} \tilde{\mathcal{K}}_{\ell_i}(\kappa_i, |\vec{r}-\vec{\tau}_i-\vec{R}|) i^\ell Y_{\ell_i m_i}(\vec{r}-\vec{\tau}_i-\vec{R}) \quad (24a)$$

$$\tilde{\mathcal{K}}_\ell(\kappa, r) \equiv \mathcal{K}_\ell(\kappa, r), \;\; r > s, \;\; s \le s_\tau \quad (24b)$$

Since rapid Fourier convergence is the criterion for constructing the pseudo-basis, it is useful to consider the Fourier integral of a Bloch function with wave-number $\vec{k}$:

$$\tilde{\psi}(\vec{g}) = -\frac{1}{\mathrm{V}_c(|\vec{k}+\vec{g}|^2 - \kappa^2)} \int_{\mathrm{V}_c} d^3r\, e^{-i(\vec{k}+\vec{g})\cdot\vec{r}} \big(\nabla^2+\kappa^2\big)\tilde{\psi}(\vec{r}) \quad (25)$$

where V_c is the unit cell volume. Equation (25) is obtained by casting $\nabla^2+\kappa^2$ on the plane wave then doing two partial integrations; surface terms vanish due to periodicity. From (25) it is evident that the Fourier integral of a pseudo-basis satisfying the first criterion (equal to the true basis in the interstitial region) may be obtained from integral over muffin-tin spheres. If in addition, the pseudo-basis is different from a Hankel function only in it's parent sphere, the Fourier integral is a finite integral over a single muffin-tin sphere. The problem then is to find a function $\tilde{\psi}$ such that $(\nabla^2+\kappa^2)\tilde{\psi}$ has a rapidly convergent Fourier integral, vanishes outside a radius less than or equal to the parent muffin-tin radius for the basis, and has a value and slope equal to $\mathcal{K}$ at this radius.

A good choice for such a function is obtained by solving

$$\big(\nabla^2 + \kappa^2\big)\, \tilde{\mathcal{K}}_\ell(\kappa, r)\mathcal{Y}_L(\hat{\vec{r}}) = -c_\ell \Big(\frac{r}{s}\Big)^\ell \Big[1 - \Big(\frac{r}{s}\Big)^2\Big]^n \mathcal{Y}_L(\hat{\vec{r}})\Theta(s-r) \quad (26)$$

for a radius $s < s_{t_i}$, and with with c_ℓ chosen to match on to $\mathcal{K}$ at s. This is easily done analytically. The resulting Fourier transform is

$$\tilde{\psi}_i(\vec{k}+\vec{g}) = \frac{4\pi}{\mathrm{V}_c} \frac{Y_{L_i}(\vec{k}+\vec{g}) e^{-\mathrm{i}(\vec{k}+\vec{g})\cdot\vec{\tau}_i}}{\left(|\vec{k}+\vec{g}|^2 - \kappa_i^2\right)} |\vec{k}+\vec{g}|^{\ell_i} \frac{\mathcal{J}_N(|\vec{k}+\vec{g}|, s)}{\mathcal{J}_N(\kappa_i, s)} \tag{27}$$

where $N = \ell_i + n_i + 1$. The subscript i has been purposely left off N and s (see below).

These coefficients converge like $1/g^{n+4}$, provided $\mathcal{J}_N(|\vec{k}+\vec{g}|, s)$ achieves it's large argument behavior, and n can be chosen to optimize convergence. Weinert [11] used an analogous construction as tool to solve Poisson's equation. He proposed a criterion for the convergence of the Fourier series (27) which amounts to choosing the exponent n in Equation (27) so that $|\vec{k}+\vec{g}_{max}|s$ would be greater than the position of the first node of $\mathcal{J}_{\ell+n+1}$. We find this criterion to be useful provided anisotropy in reciprocal space is accounted for. This is accomplished by using the minimum reciprocal lattice vector on the surface of maximal reciprocal lattice vectors, rather than simply using g_{max}.

Notice that this criterion is a criterion for $N = \ell + n + 1$. The basis Fourier components are simplified, and the amount of information stored reduced, by simply using a single argument for all bases; *i.e.* all bases use the same value of N. It is also possible to use a single radius s, less than or equal to the smallest muffin-tin radius, since the only requirement is on the pseudo bases in the interstitial region. In practice, a few radii are desirable if large and small atoms are present in the same calculation, since small radii give less convergent Fourier coefficients. In any event, no more than a few radii are necessary to handle systems with many atoms. Notice also that local coordinates have been left out of (27). The resulting potential matrix may be easily rotated to local coordinates at the end of the calculation.

As expressed in (27), the Fourier components are products of phases $e^{-\mathrm{i}(\vec{k}+\vec{g})\cdot\vec{\tau}}$, which scale like the number of atoms squared (the size of the reciprocal lattice grid grows linearly with the number of atoms), and a function of lattice vectors and a few parameters, which scales linearly with the number of atoms. The phase factors are simple to calculate by accumulation and need not be stored.

The potential in the interstitial region is similarly obtained from a "pseudo-potential" $\tilde{V}$ that equals the true potential in the interstitial region and has rapidly converging Fourier coefficients:

$$V(\vec{r})\Big|_{\mathcal{I}} = \tilde{V}(\vec{r})\Big|_{\mathcal{I}} \tag{28a}$$

$$\tilde{V}(\vec{r}) = \sum_{\mathcal{S}} \tilde{V}(\mathcal{S}) D_{\mathcal{S}}(\vec{r}) \tag{28b}$$

$$D_{\mathcal{S}} = \sum_{\vec{g}\in\mathcal{S}} e^{\mathrm{i}\vec{g}\cdot\vec{r}} \tag{28c}$$

The sum in Equation (28b) is over stars $\mathcal{S}$ of the reciprocal lattice.

Integrals over the interstitial region are performed by convoluting the potential with an interstitial region step function and integrating over the unit cell:

$$\langle\psi_i|V|\psi_j\rangle_{\mathcal{I}} = \langle\tilde{\psi}_i|\tilde{V}|\tilde{\psi}_j\rangle_{\mathcal{I}} = \langle\tilde{\psi}_i|\theta_{\mathcal{I}}\tilde{V}|\tilde{\psi}_j\rangle_c \quad .$$

The potential matrix element is calculated by convoluting the convoluted potential with a basis, and performing a direct product between convoluted and unconvoluted bases. If basis functions are calculated n^3 reciprocal lattice vectors, the interstitial potential will be calculated on $(2n)^3$ vectors. The convolution is exact if it is carried out on a lattice containing $(4n)^3$ vectors. The size of the set of reciprocal lattice vectors necessary to converge the total energy using this treatment of the interstitial region varies from between $\sim$ 150 – 300 basis plane waves per atom, depending on the smoothness of the potential and the convergence required.

Another way of integrating over the interstitial region, more usual in site-centered methods, is to integrate Fourier series over the unit cell and subtract the muffin-tin contributions with pseudo-bases and pseudo-potential expressed as an expansion in spherical waves. The convolution has an advantage in acting with a single representation, and, given a finite representation for bases and potential, the convolution may be done exactly.

Empty spheres are never used with this scheme. Bases, and the charge density and potential are calculated as accurately as necessary using the scheme described above and a basis set expanded with tail parameters and energy sets has proven to be flexible enough to accurately describe the contribution of the electronic states in the interstitial region.

2.4 Charge Density

When a solution to the wave equation at every physical energy is available, the charge density may be obtained from a set of energy-dependent coefficients. The spherically symmetric charge density in a muffin-tin sphere, coupled with an $\ell - projected$ density of states, is an example. In a variational calculation, as is being described here, all that is available is a (variational) solution to the wave equation at a set of discreet energies, and the charge density must be obtained simply from the square of the eigenvectors, or equivalently from expectation values of occupation numbers.

Having calculated a set of eigenvalues and eigenvectors $\mathcal{A}$ of the generalized eigenvalue problem, the charge density in the interstitial region is

$$\tilde{n}(\vec{r})\Big|_{\mathcal{I}} = \sum_{\mathcal{S}} \tilde{n}(\mathcal{S}) D_{\mathcal{S}}(\vec{r}) \tag{29a}$$

$$\tilde{n}(\mathcal{S}) = \frac{1}{N_{\mathcal{S}}} \sum_{\vec{g}\in\mathcal{S}} \sum_{nk} w_{nk} \frac{1}{\mathrm{V}_c} \int_{\mathrm{V}_c} d^3r\, e^{-\mathrm{i}\vec{g}\cdot\vec{r}} \Big| \sum_i \tilde{\psi}_i(k,\vec{r}) \mathcal{A}_i(nk) \Big|^2 \tag{29b}$$

where $N_{\mathcal{S}}$ is the number of vectors in the reciprocal lattice star $\mathcal{S}$. The square of the wave function is obtained by convoluting the Fourier components of ψ with $\mathcal{A}$, Fourier transforming, and taking the modulus.

In the muffin-tin spheres the charge density is

$$n(\vec{r})\Big|_{r_\tau<s_t} = \sum_h n_{ht}(r_\tau)D_{ht}(\mathcal{D}_\tau r_\tau) \quad (30a)$$

$$n_{ht}(r) = \sum_{e\ell}\sum_{e'\ell'} U_{t\ell'}(e_{i'},r)M_{ht}(e\ell,e'\ell')U^T_{t\ell}(e_i,r) \quad (30b)$$

$$M_{ht}(e\ell,e'\ell') = \frac{2\ell_h+1}{4\pi}\sum_{m_h m m'}\alpha^*_{ht}(m_h)\mathcal{G}\big(\ell,\,m;\,\ell',\,m';\,\ell_h,\,m_h\big) \times \sum_{nk} w_{nk}\mathcal{V}_{\tau\ell m}(e)\mathcal{V}^\dagger_{\tau\ell' m'}(e') \quad (30c)$$

$$\mathcal{V}_{\tau\ell m}(e) = \sum_i \delta(e,e_i)\Omega_{t\ell}(e,\kappa_i)\mathrm{S}_{\ell m,\ell_i m_i}(\kappa_i,\vec{\tau}-\vec{\tau}_i,\vec{k})\mathcal{A}_i(n\vec{k}) \quad (30d)$$

The process of calculation is evident in the sequence of equations.

2.5 Core States

Core states, even spherically symmetric complete shells, contribute non-muffin-tin components to the interstitial region and to muffin-tin spheres of other surrounding sites. Whether it is essential to include this contribution depends on the size of the contribution, and any sizeable contribution implies that there are states being treated as localized which aren't localized within the scope of the calculation. Nevertheless, confining states to the core is often useful, and including the core contribution to the full potential is not difficult. One possibility, the one used in this method, is to fit the part of the core electron density to a linear combination of Hankel functions, and expand this density in the interstitial region as a Fourier series and in the muffin-tin spheres in a harmonic series, in the same way the basis functions are treated.

2.6 Potential

2.6.1 Coulomb Potential

The Coulomb potential is obtained by first calculating the Coulomb potential in the interstitial region, then, using the value of the interstitial potential on the muffin-tin sphere, calculating the potential in the spheres by a numerical Coulomb integral of the muffin-tin electron density for each harmonic.

The interstitial Coulomb potential is calculated in a way similar to that suggested by Weinert [11]. Express the electron density as

$$n(\vec{r}) = \tilde{n}(\vec{r}) + \sum_{R\tau}\big(n(\vec{r})-\tilde{n}(\vec{r})\big)\Theta(s_t-r_\tau) \quad (31)$$

where $\tilde{n}$ is the squared modulus of the pseudo-eigenvectors, which is equal to the true electron density in the interstitial region. The first term on the right-hand side of (31) has, by construction, a convergent Fourier series. The second term is confined to muffin-tin spheres. To calculate the Coulomb potential in the interstitial region, this term may be replaced by any density also confined to the muffin-tin spheres and having the same multipole moments. If a charge

density satisfies these requirements and also has a convergent Fourier series, the Coulomb potential in the interstitial region may be easily calculated from the combined Fourier series. Such a charge density can be constructed in a similar way to that detailed for the pseudo-bases. Construct a pseudo charge-density satisfying

$$\tilde{n}^{(p)}(\vec{r}) = \sum_{R\tau}\sum_{h} \tilde{n}^{(p)}(ht, r_{R\tau}) D_{ht}(\mathcal{D}_\tau \hat{\vec{r}}_{R\tau}) \tag{32a}$$

$$\tilde{n}^{(p)}_{ht}(r) = c_{ht}\Big(\frac{r}{s_t}\Big)^{\ell_h}\Big(1-\Big(\frac{r}{s_t}\Big)^2\Big)^n \Theta(s_t - r) \tag{32b}$$

$$0 = \int_\tau d^3r\, r_\tau^\ell D^*_{ht}(\mathcal{D}_\tau \hat{\vec{r}}_\tau)\big(\tilde{n}^{(p)}(\vec{r}) - n(\vec{r}) + \tilde{n}(\vec{r})\big) \quad . \tag{32c}$$

This charge density has Fourier components

$$\begin{aligned}\tilde{n}^{(p)}(\vec{r}) = &\sum_\tau\sum_h e^{-\mathrm{i}\vec{g}\cdot\vec{\tau}}(-i)^{\ell_h} D_{ht}(\mathcal{D}_\tau \vec{g}) \frac{4\pi}{\mathrm{V}_c} \frac{(Q_{ht}\{n\} - Q_{ht}\{\tilde{n}\})}{s^{\ell_h+n+1}} \\ &\times \frac{\big(2(\ell_h+n+1)+1\big)!!}{(2\ell_h+1)!!} g^{\ell_h} \mathcal{J}_{\ell_h+n+1}(g, s_t)\end{aligned} \tag{33}$$

where the multipole moments Q are defined by

$$Q_{ht}\{n\} = \frac{2\ell_h+1}{4\pi}\int_{s_t>r_\tau} r_\tau^{\ell_h} D_{ht}(\hat{\vec{r}}_\tau) n(\vec{r})\, d^3r_\tau \tag{34}$$

The Fourier components $\tilde{n}^{(p)}(\vec{r})$ converge like $1/g^{n+2}$ provided $j_{\ell+n+1}$ attains it's asymptotic form. The exponent n is chosen using the same considerations as for the pseudo-basis set.

The Coulomb potential in the interstitial region is then given by

$$\begin{aligned} V_c(\vec{r})\Big|_{\mathcal{I}} &= \tilde{V}_c(\vec{r})\Big|_{\mathcal{I}} \\ &= \sum_{g\neq 0} \frac{4\pi e^2\big(\tilde{n}(g) + n^{(p)}(g)\big)}{g^2} e^{\mathrm{i}\vec{g}\cdot\vec{r}} \end{aligned} \tag{35}$$

From the Coulomb potential in the interstitial region follows the Coulomb potential on the surface of the muffin-tin spheres. The coulomb Potential inside the muffin-tin spheres is

$$\begin{aligned} V^{(c)}(\vec{r})\Big|_{r_\tau<s_t} = &\sum_h D_{ht}(\mathcal{D}_\tau \hat{\vec{r}}_\tau)\Big[e^2\int_0^{s_t} \frac{r_<^{\ell_h}}{r_>^{\ell_h+1}} \frac{4\pi r'^2 n_h(r)}{2\ell_h+1} dr' \\ &+ \Big(V_h^{(c)}(s) - \frac{e^2}{s^{\ell_h+1}}\int_0^s \frac{4\pi r'^{\ell_h+2} n_h(r')}{2\ell_h+1} dr'\Big)\Big(\frac{r}{s}\Big)^{\ell_h}\Big] \end{aligned} \tag{36}$$

where

$$V^{(c)}_{ht}(s_t) \equiv \frac{2\ell_h+1}{4\pi}\int_{r_\tau=s_t} d\hat{\vec{r}} D^*_{ht}(\mathcal{D}_\tau \hat{\vec{r}}) V^{(c)}(\vec{r}) \tag{37}$$

is the harmonic component of the potential on a sphere boundary.

2.6.2 Density Gradients

Gradients of the electron density are needed for the evaluation of gradient corrected density functionals. These functionals depend on invariants (with respect to the point group) constructed from density gradients (*e.g.* $|\vec{\nabla}n|^2$). This reduces computation significantly in the muffin-tin spheres, for if f and g are invariant functions (*i.e.* $f(\vec{r}) = \sum_h f_h(r) D_h(\hat{\vec{r}})$), and $d = \vec{\nabla} f \cdot \vec{\nabla} g$, then $d(\vec{r}) = \sum_h d_h(r) D_h(\hat{\vec{r}})$ with

$$\frac{4\pi r^2}{2\ell_h+1} d_h(r) \quad = \quad \sum_{h,h'} \sum_{k,k'=\pm 1} f_h^{(k)}(r) g_{h'}^{(k')}(r) I(kk';hh') \tag{38}$$

where the set of parameters I is easily calculable from $3j$ and $6j$ coefficients and integrals over the harmonic functions D_h, and

$$f_h^{(k)} \quad = \quad \frac{4\pi}{2\ell+1} \begin{cases} rf' - \ell_h & k = 1 \\ rf' + \ell_h + 1 & k = -1 \end{cases} \tag{39}$$

and similarly for g.

Gradients of the interstitial charge density, represented as a Fourier series, are poorly represented by differentiating the series term by term. A stable representation of the density gradient that converges well is obtained by defining the derivative as the difference between adjacent grid points, divided by twice the grid spacing as suggested by Lanczos.[12] This is equivalent to differentiating, term by term, the Lanczos-damped series for the charge density.

3 Installation of RSPt

Important. *Compiling and running high performance code is* highly *system dependent. The program package comes with sample settings for a number of systems and these may, and then again may not, be helpful. Below follow descriptions of critical settings, but these will not necessarily help you accomplish those settings on your system. Remember that your local support is always the number one source of information regarding things like libraries, compilers etc.*

The RSPt package not only contains the FORTRAN and C RSPt engine, but also a large number of supporting programs, most of these written in C. These programs are in several sub-directories, consult the Makefile if you want to dig deeper into this, but it should not be necessary to do so.

When the tar ball (rsptnnn.tar, where nnn is the release number) you downloaded from http://www.rspt.net/ is unpacked, these files should be in the rsptnnn directory:

- Makefile
- RSPTmake.inc
- atom/
- bin/

- binC/
- cub/
- documentation/
- include/
- inputs/
- lib/
- rsptDir/
- RSPTmakes/
- runRspt/
- sym/
- testsuite/

The name of (path to) the directory where these files are will be needed later and we call it **RSPTHOME/**. It could be useful to set RSPTHOME in your shell profile (that is, your **.profile**, **.bashrc**, **.login .cshrc** or **.tcshrc** file depending on which shell you run) and add `$RSPTHOME/bin` to your PATH now.

For this implementation you will need a C compiler , e.g. *gcc* or *icc*, and a FORTRAN90 compiler accepting Cray pointers , e.g. *gfortran* or *ifort*. It is recommended that you use a version of *gcc* and *gfortran* newer than 4.1. Older versions including the earlier 4.1 contains some serious bugs and will not compile RSPt. *G77* will also not compile RSPt. You will also need BLAS and LAPACK (e.g. Intel MKL or Atlas), and FFTW-3 (http://www.fftw.org).

In RSPTHOME:

1. Edit **RSPTmake.inc**. This file is used to set some system variables used by the compiler. The order is not important but any duplicate entry will overwrite the previous definitions. RSPt is written both in FORTRAN and C. The two languages uses slightly different naming conventions. Most FORTRAN compilers, but not all, adds an underscore to the names of the subroutines and functions and folds all names to lower case. C preserves the case and do not add an underscore. There are several different ways to handle this situation in RSPt an explicit solution was chosen, where FORTRAN callable functions in the C part of the code have an underscore added to their names.

If you see a lot of missing references to RSPt functions (in contrast to library functions from BLAS, LAPACK or FFTW, then you have made a mistake in the definition of the library names and/or paths, see below) in the link step you probably have used a FORTRAN compiler which don't add an underscore by default. Read your local documentation to find out how your FORTRAN compiler behaves and which option adds an underscore if that is not the default behavior.

- FCOMPILER: is the name of the FORTRAN compiler.

- FHOME: should be the top of the FORTRAN compiler home directory (e.g. **/opt/intel/Compiler/11.0/074**). Can be used to simplify the path to the compiler and/or run-time libraries. If the full path to the compiler or libraries is given in FCOMPILER and FORTRANLIBS, FHOME can be left empty.
- FCOMPILERFLAGS: options passed to the FORTRAN compiler, e.g. regarding performance or debugging.
- FCPPFLAGS: if you are going to compile a plain serial version of RSPt you can leave this empty, if not you'll need to put the pre-processor options here. The different options are explained in section 4.4.
- FTARGETARCH: used for cross compiling, otherwise leave empty.
- FORTRANLIBS: all the libraries your FORTRAN compiler depends on. This is necessary because the loader is called from the C compiler. Look in $(FHOME)/lib and read the documentation for your FORTRAN compiler.
- F90COMPILER and F90COMPILERFLAGS: If you for some reason prefer to use a separate F77 compiler as FCOMPILER you can define your F90 compiler and its options here. As F77 is a subset of F90 you can use the F90 compiler in both places. In the Makefile the F90 compiler is used with both the FCOMPILERFLAGS and F90COMPILERFLAGS so you don't have to put common options at both places.
- CCOMPILER: the C compiler.
- CCOMPILERFLAGS: the compiler flags for the C compiler.
- CTARGETARCH: should be the same as FTARGETARCH.
- CPPFLAGS: should in principle be the same as FCPPFLAGS but as some FORTRAN compilers need extra options (often -Wp) to pass options to the pre-processor CPPFLAGS is kept as a separate entry.
- CLOADER: should be left empty unless some other loader then the one called by the C compiler is used.
- LAPACKLIB: the LAPACK library and any support libraries needed. Could be given either as absolute names or with LPATH_TO_LAPACK and -lNAME_OF_LAPACK.
- BLASLIB: the same for BLAS.
- FFTWLIB: the path and name of the FFTW library.
- EXTRALIBS: any extra library needed. See section 4.4.
- INCLUDEDIRS: Path to extra directories where files to be included might be found, e.g. **fftw3.h**. Use -I, like in -I/usr/local/include.

2. type *make*

If something goes wrong, check what you've done, correct any mistakes, type *make pristine* and try again. If you still have a problem, write up a detailed description of what went wrong and ask the Forum at the web page or the forum at the github repository.

If you don't get any error messages (you might get a few warnings depending on the choice of compilers) you can now find the executable *rspt*in $(RSPTHOME)/bin together with a lot of other utility programs.

There are a few small tests in the **testsuite/** directory set up to be run using *beetest*. Do *make test* in the **testsuite/** directory. You can see errors within the expected numerical precision (10^{-13}-10^{-15}) compared with the archived results depending on your specific platform.

Note for the experienced. *Recent versions of RSPt don't use the file* **para.glo** *and only needs to be compiled once.*

4 How to run RSPt

In order to run *rspt*you first need a set of input files. You can read more about how to create these files and what they contain in Section 5.1. You will most likely be wanting to turn pages back and forth between Sections 4 and 5 as you read, so you may as well put a mark there now. RSPt solves the LDA problem using an iterative process where it uses the solution from the last iteration as input for the next. The program *rspt*will only run for one cycle in the convergence process. There are a few different programs available in $(RSPTHOME)/bin to run *rspt*to an arbitrary level of convergence. The program *rspt*can of course be used by itself to run just one cycle, e.g. if you want to calculate the density of states (DOS) or optical properties from an already converged solution.

4.1 Basic ground state run

The most basic program is *runs*, which requires three arguments. The first argument is the *rspt*executable, the second the level of convergence, and the third the maximum number of iterations, e.g.

```
$(RSPTHOME)/bin/runs $(RSPTHOME)/bin/rspt 1e-12 40
```

See Section 5.2 for more information about the convergence parameter. If you are using a parallel version of *rspt*you need to encapsulate the *mpirun* command so that *runs* doesn't confuse the arguments to *mpirun* with the arguments to *runs*, e.g. "$(RSPTHOME)/bin/runs 'mpirun -np 4 $(RSPTHOME)/bin/rspt' 1e-12 40". The cycle will then stop when convergence is reached or at most after the maximum number of iterations.

Note that *rspt*allocates most of its variables dynamically but not all. The first fact means that the heap size must be increased for large calculations. This is usually done using *ulimit* or *limit*, depending on your choice of shell. If your calculation just dies without any error messages from the program, this might be the problem. Some of the variables are pointers to a big scratch array (old trick from the days before FORTRAN 90). The default size of that array is usually big enough but if the program stops saying "insufficient space" in the

end of the **out** file you must create the file **rsptsize** with one free format integer on the first line giving the size of the scratch array in mega-words. On most modern computers one mega-word is four megabytes.

4.2 Other runs

If RSPt is used to calculate some property as function of volume the utility program *runsl* can be used. The program reads a set of length scales from the file **lengths** which contains one length scale per line. *runsl* is used in the same way as *runs*.

There are similar programs that can read a list of strains from a file, (*runstrs*), or a combinations of lengths and strains, (*runstrsl*). There is also a program, *runsf*, that can relax a structure using forces. Note that the shape of the cell is fixed, it is just the atomic positions that are optimized. The program reads the file **runsf.inp** containing five lines where, in that order, the convergence tolerance for the optimization, the number of the first iteration, the number of the last iteration, the first mixing parameter and the prefix used to save files are given. Relaxation of the atomic positions can be combined with a set of length scales in *runslf*.

In the cases where more than one set of results are produced a directory called **results/** is created and the input files and the most important output files are saved in one archive (**runs.a**) for each result.

4.3 Memory store mode

Some clusters experience problems when reading/writing repeatedly from the hard drive, especially for parallel runs. To deal with this problem a memory store mode can be activated which saves the eigenvectors, density-, overlap- and hamilton-matrix to memory instead of continuously accessing to disk. The memory store mode can be activated by adding -DMEMORY_STORE added to FCPPFLAGS and CPPFLAGS before compiling the code.

4.4 Parallel RSPt

RSPt can be used in both a serial and several parallel modes. Without any pre-processor options the serial version is compiled. The code is parallel on three different levels. The most basic mode is parallel over k-points. This mode is suitable when running on a cluster with a slow interconnect as the communication overhead is small. The next mode is parallel over both k-points and bands. The band parallelization is more communication intensive and is only recommended on computers with a high-bandwidth/short-latency interconnect.

The third mode is threads parallel over the FFT, either using the regular CPU or an auxiliary GPU. This mode can only be used on shared memory nodes or on computers with a GPU. If a cluster is built from fat nodes band and/or k-point parallelization can be used between the nodes and threads within the node. FFT thread parallelization can in principle be used without the other two modes of parallelization but the gain is smaller and the overhead larger than for k-point parallelization so if the number of nodes is limited the latter is preferred.

The code is basically task parallel so the memory footprint per task is the same. Parallelization will buy you time but not space. This is usually not a problem as the memory footprint of RSPt is rather small to begin with compared to most plane-wave pseudo potential codes. If one requires memory saving for studying very large systems, then the peak memory can be strongly reduced by using MPI distributed FFTW.

4.4.1 Parallelization over k-points

To compile RSPt in a k-point parallel mode set FCPPFLAGS and CPPFLAGS to -DMPI and change the names of the FORTAN and C compilers to their MPI versions (e.g. *mpif90* and *mpicc*). You also have to add the FORTRAN MPI run-time library. This is again because the C compiler is used to link the program and by default it will only link the C MPI run-time library automatically. The FORTRAN run-time library is often call **libmpi_f77.a** but look in your local documentation to find the exact name and location.

It might be a good idea to only compile *rspt*with the parallel C and FORTRAN compilers, not all the utility programs. If you choose to do like this, after you have compiled the serial version of *rspt*, change the name of *rspt*to something else, like *rspt_s*. After that edit **RSPTmake.inc** and change directory to $(RSPTDIR)/**rsptDir/src/** and type *make clean; make*. This will compile a parallel version of *rspt*depending on the choice of pre-compiler options. As *make* always will produce a executable called *rspt*independent of the pre-compiler options it is a good idea to change the name of the new executable as well.

If you see errors in the link step referring to missing MPI functions whose names are ending with an underscore you missed to add the FORTRAN MPI runtime library. In principle FORTRAN could use the C MPI run-time library but due to name space conventions that would mess up a lot of other things.

Run RSPt with the commands and options defined by your local MPI installation. Often it would be something like

```
mpirun -np N $(RSPTHOME)/bin/rspt
```

where N is the number of copies of the program you want to run. If N is less than the number of k-points the k-points are distributed over the N tasks. If N is larger than the number of k-points the computational work for each k-point is distributed over more than one core or process. Because the parallelization over k-points is always more efficient than over bands RSPt always tries to use as many processes as possible for sharing the work over k-points. The process of dividing the tasks is automatic.

When RSPt is run in a k-point parallel mode some of the temporary and output files (see Section 5.2 will be duplicated for each copy of the program. These files will have the node number attached to the name. For the moment all copies of the program must have access to a common file system, i.e. all nodes must have access to RSPTHOME.

4.4.2 Parallelization over k-points and bands

If the systems under study are big, or if the number of k-points is small, it may be convenient to use also the parallelization over bands. The parallelization over bands does not require any additional compilation in addition to what is

described above, but is directly controlled when running the code. If a user runs RSPt for a system with n_1 k-points and uses $n_1 * n_2$ ranks, then the code will automatically assume that the user intends to use n_1 ranks for k-points parallelization and n_2 ranks for band parallelization. If the total number of ranks cannot be divided exactly by the number of k-points, then the code will stop. In this case, or if the user wishes to input the number of ranks to be used for the parallelization grid, then a file **CARTESIAN_TOPOLOGY** has to be placed into the simulation folder. This file consists of two lines. The first line contains the an integer number n_1 to use for k-point parallelization, while the second line contains an integer number n_2 to be used for band parallelization. If $n_1 * n_2$ is different than the total number of ranks, the code will stop.

4.4.3 Parallelization over FFT, using the CPU

The Fast Fourier Transform used by RSPt, FFTW3, supports thread level parallelization . If the FFTW3 library has been compiled with threads this can be used with the pre-compiler flag -DHAVE_FFTW3_THREADS. This option can be used together with the k-point parallel mode as well as with the serial mode. In the latter case use the regular FORTRAN and C compiler, not their MPI counterparts.

In the directory where RSPt will be run create a file **FFTW_MAX_THREADS** with an integer on the first line giving the maximum number of threads available to FFTW. Please note that depending on the exact version of FFTW3 this can mean different things in the parallel modes. In some versions this means the maximum number of threads for all copies of RSPt run by *mpirun* and in some the maximum number for any copy. There might also be a difference depending on if all copies are run on one physical node sharing the same memory and OS or if the tasks are shared by several nodes. The only way to be really sure is to read the local documentation and test the different configurations on your local computer.

The FFT parallel mode can only be used on shared memory machines and again the overhead is larger than for the k-point parallel mode. Use this option only if you have cores or processors to spare or if you for some reason cannot use MPI. Depending on the size of the Fourier mesh (see Section 5.1) the FFT might take a substantial part of the total execution time (in some extreme cases up to 95 % of the total time) and then this might be a useful option.

4.4.4 Parallelization over FFT, using the GPU

If the computer used for the calculation is equipped with a GPU from NVidia the FFT can be transferred to the GPU using the CUDA library. The CUDA FFT is invoked by using the precompiler (FCPPFLAGS and CPPFLAGS) option -DGPUFFT. The option CCOMPILER must be set to NVidias nvcc and the cufft library must be linked.

The performance for smaller transforms is dominated by the latency in the copy-in/copy-out process and the balance point is strongly hardware dependent but tests indicate that the GPU FFT is faster than the CPU FFT for sizes larger than $64 \times 64 \times 64$ for low range gaming style GPUs. The break-even point should be lower for dedicated GPUs from the Tesla family. The speed also depend strongly (and non-linearly) on the size of the transform.

Formally the GPU must at least have Compute Capability 1.3 but it is recommended to use card with CC from 2.0 and up. Read the CUDA documentation until you understand the difference between -arch and -code and the different passes which the compiler is doing. Using the GPU is not for the fainthearted but it could be a cheap fix for FFT dominated problems.

4.4.5 Parallelization over FFT, memory distributed

If one intends to study very large systems for which a huge Fourier mesh is required, it may be advantageous to use MPI distributed FFTW. The option -DMPI_FFTW tells the code that FFT in the convolution should be done through the FFTW3 MPI interface. With this flag no rank has to allocate the full FFT matrix, and the FFT should be faster for problems where the FFT grid is large.

This implementation can be complicated to use if the FFTW3 routines come from math libraries, such as MKL. For the latter, the FFTW3 wrapper has to be compiled. This should be done by setting MKL_FFTW_WRAPPER in the **RSPtmake.inc**. This flag should contain information on the compiler and the MPI libraries. The main issue here is that the base code in RSPt should use the sequential MKL implementation, while the wrappers should use the cluster MKL implementation. Thus one has to be careful of the order in which the libraries are placed in the command line. For MKL, this should be handled correctly with the flag MKL_FFTW_WRAPPER. Examples for these flags are:

```
${MKL_FFTW_WRAPPER}=intel openmpi
${MKL_FFTW_WRAPPER}=gnu intelmpi
```

This fix is probably going to work for most systems, but it may require some trial and error approach. Some other examples can be found in the RSPtmakes folder.

5 Files used by RSPt

RSPt uses a few input files, produces a lot of temporary files and some output files. All input files but one can be created automatically using default values that are useful in most cases. The files unique to DMFT will be described in Section 11. The utility program RPST (not to be confused with RSPt) than also can be used to create input files is described in Section 13.1. RSPt is usually run in a directory where the input files for a set of inputs are. We will call that directory **RUNDIR/** from here on.

5.1 Input files

Even though RSPt can do a lot of things it still can't guess what material you want to study. You need to define the Bravais lattice and the atomic positions and types at least.

5.1.1 symt.inp, old format

Create a directory called **sym/** in RUNDIR. In $(RUNDIR)/**sym/** you create a file called **symt.inp** which contains (apart from comments which starts with #):

- The Bravais lattice
- Bravais lattice. The three Cartesian components of the lattice vectors, one column for each vector. This is the only place where the vectors are written in column order , in the data file the vectors are written in row order. This is not because we don't know the difference between the ordering of multidimensional arrays in FORTRAN and C. The reason is a bit technical so just trust us. Note, for hcp and dhcp structures, the vectors must be given with high accuracy if the symmetry generator is going to be able to find all the symmetries. Try to keep the components close to unity or fractions thereof and set the length scale in the **data** file.
- The spin axis and a tag (l or w). Three components, either in lattice coordinates (tag=l), Cartesian coordinates (tag=a or c) or Wyckoff (tag=w). In the latter case a file called **l_to_w** with the nine components of the transformation matrix from lattice to Wyckoff coordinates must exists in the **sym/** directory. A non-zero spin axis will lower the symmetry accordingly and also tell the symmetry generator to prepare the other input files for a spin polarized calculation compatible with spin-orbit coupling. The spin axis can also be chosen as 0.0 0.0 0.0 for a non-spin polarized calculation.
- The total number of atoms. You need to put all atoms in the unit cell or supercell in by hand. RPST can construct supercells, see Section 13.1.
- The definitions of the atoms. One line for each atom, even if they are of the same type. Three coordinate components, the atomic number, a tag for the type of coordinate and an identity tag. The coordinates could either be lattice (coordinate tag=l), Wyckoff ((coordinate tag=w) or Cartesian (coordinate tag=a or c). If the file **offsets** exists with one line with a shift (three components) for each atom the positions will be shifted by the given displacement vector. This can be used for optimization of the atomic positions. The identity tag can be used to group atoms together and is given as a letter from a to z. Atoms with different atomic numbers are always treated as different and should be given different tags but there are cases where atoms with the same atomic number must be treated as different types. If a different tag is chosen for the different groups of atoms the symmetry will be lowered accordingly and the input files will be set up for this configuration.
- A strain matrix . The lattice can be strained without lowering the symmetry. Defined in the same way as the Bravais lattice but always in lattice coordinates. If no strain is going to be used, just write a three by three identity matrix.

Below is an example of **symt.inp** for UO_2:

```
# UO2, fluorite structure, AF
  5.000000000e-01  -5.0000000000e-01   0.0000000000e+00
  5.000000000e-01   5.0000000000e-01   0.0000000000e+00
  0.000000000e+00   0.0000000000e+00   1.0000000000e+00
#
0.0 0.0 1.0  l
#
6
    0.000     0.000     0.000   92  l  a
    0.500     0.500     0.500   92  l  b
    0.000     0.500     0.250    8  l  c
    0.000     0.500     0.750    8  l  c
    0.500     0.000     0.750    8  l  c
    0.500     0.000     0.250    8  l  c
#
1.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
```

When the file is prepared the symmetry generator will create all the necessary input files or the files that is used to create the input files. In the directory $(RUNDIR)/**sym/** run *symt* as

```
$(RSPTDIR)/bin/symt -all
```

which will create a few directories and files in **RUNDIR/**.

5.1.2 symt.inp, new format

The *symt* program now accepts a new format, based on data blocks, which facilitates the addition of new control instructions for the file generation. The behavior of the program is pretty similar to earlier, but there are some additional option available only with the new input format.

Here is the same input file as above, but in the new format. For descriptions of these data blocks, see above.

```
# UO2, fluorite structure, AF
latticevectors
  5.000000000e-01  -5.0000000000e-01   0.0000000000e+00
  5.000000000e-01   5.0000000000e-01   0.0000000000e+00
  0.000000000e+00   0.0000000000e+00   1.0000000000e+00
#
spinpol
#
spinaxis
0.0 0.0 1.0  l
#
atoms
6
    0.000     0.000     0.000   92  l  dn
    0.500     0.500     0.500   92  l  up
    0.000     0.500     0.250    8  l  a
    0.000     0.500     0.750    8  l  a
```

```
    0.500     0.000     0.750    8  l  a
    0.500     0.000     0.250    8  l  a
#
strainmatrix
1.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
```

Note that the mostly what has happened is that each block got a name to identify it. Comments are still signaled by the # character etc. In the new format, however, we can put the blocks in any order we like and even leave out some of the blocks. There is also a new feature in that the identity tags for the atoms can be used to generate a spin-polarized starting density, as described below. The code can obviously not provide any guess for what atomic species you want, so the `atom` block is mandatory, but defaults are provided for the lattice vectors (a simple cubic lattice), spin vector (just three zeroes, as above) and strain matrix (the default being to not use the strain format, just as in the old format). Note that *within* each block the format is fixed, that is, the `latticevectors` block must consist of nine floating point numbers.

List of all current options:

`latticevectors` A 3×3 matrix whose *columns* are the Bravais lattice vectors. Default: Unit matrix.

`spinaxis` Three numbers giving the axis followed by a single character that gives the coordinates. The choices are `l` for lattice, `c` for cartesian and `w` for Wyckoff coordinates (requires that a mapping is defined in a `l_to_w` block). Default: (0,0,0).

`atoms` First a single integer giving the number of atomic positions to be read. Then follows, for each position: Three numbers giving the coordinates of the atom, one integer giving the atomic number, one character telling which coordinate system the position was given in and finally an identity tag, which can be used to distinguish between otherwise equivalent atoms. You can put any string here, but avoid characters that has special meanings on your operating system, since it will be used in naming files. There are two tag strings which has a special meaning, namely `up` and `dn`, which will cause generation of spin-polarized starting densities with the majority up and down spin, respectively. There is no default for `atom`, this block must be provided.

`strainmatrix` A 3×3 matrix used to strain the Bravais lattice. This will also make the *symt* generate input files to *rspt*to use a strain matrix. Default: Unit matrix.

`lengthscale` A single number giving the overall length scale. Default: The RSPt default length scale, which is given by the equilibrium volumes for the separate elemental compounds. This will typically be too large.

`l_to_w` A 3×3 matrix giving the mapping from lattice to Wyckoff coordinates (i.e. if $\mathbf{a}$ is the vector with an atomic position in lattice coordinates and $\mathbf{w}$ is the same position in Wyckoff coordinates, then you

need to supply a matrix $\mathbf{M}$ such that $\mathbf{w} = \mathbf{Ma}$). Default: None. This *must* be given if `w` was specified for any of the coordinates given above (but only then).

`offsets` A set of displacements of each of the atomic positions given above. Each displacement is given by three numbers and you must give one displacement per atomic site. Warning! This block must appear *after* the `atom` block. Default: a set of zero displacements.

`mtradii` An integer specifying which method to use to determine the muffin-tin radii to be generated. The current choices are:

- **`0`** Put an estimate of the minimal radius such that you don't get too large core leakage.
- **`1`** Slater atomic radii.
- **`2`** Slater atomic radii, safeguarded by the requirement that the spheres must not overlap.
- **`3`** Algorithm from the Elk program: $R_i \propto 1+\zeta Z_i^{1/3}$. The radii are increased until the smallest distance between two spheres is 95% of touching. $\zeta = 0.25$

Choices 1 and 2 are not recommended, 3 seems to work well. The 0 option provides a useful piece of information and will probably never result in overlapping spheres. Default: `0`.

`lmax` Maximal ℓ to use for expansion of symmetric functions (density, potential etc.). Default: 8.

`spinpol` If keyword is present, generate a spin-polarized setup.

`fullrel` If keyword is present, generate a setup with spin-orbit coupling and relativistic symmetries using the spin axis given in the `spinaxis` block.

`spinpol_atomdens` If keyword is present, spin polarized starting atomic densities will automatically be produced. By default all atoms with d- or f- states will be set up to get a magnetic moment with the "down" channel as majority spin. The moment is set by setting up the state according to Hund's first rule. To control the direction of the moment on a specific atom, put `up` or `dn` as tags on the atoms, as described above. Note that the tag method of setting initial moments have precedence over this one, so if you set the atom tags for all atoms you want to polarize you do not need to set this switch.

5.1.3 atomdens

RSPt uses the atomic density for each atom to create a starting guess for the first iteration. The symmetry generator creates the necessary files in the directory $(RUNDIR)/**atom/**. For each kind of atomic type there is one file called **data_N.X**, where N is the atomic number and X is the identity tag. If $(RSPTDIR)/bin is not already in your PATH you must either add the directory now

or edit $(RUNDIR)/**atom/Makefile** and give the full path to the program *atom* in that file, once for each atomic type.

In principle you should never have to edit the file with the atomic data but if you choose to do so the format is as follows (line by line, oxygen in this example):

```
O
```

Comment, the type of atom.

```
8          60.        0.025   594      2
```

Atomic number, size of confining parabolic potential, logarithmic step size for mesh, number of mesh points, functional (see Section 5.1.6).

```
4        0.300     50      1.0e-09
```

Number of bases (the same as the number of lines below), mix ratio for iterative solver, maximum number of iterations, tolerance.

```
1    -1          2.      0      r
2    -1          2.      0      r
2     1          2.      0      r
2    -2          2.      0      r
```

For each base, the principal quantum number, relativistic quantum number κ, occupation, spin, status (r=relativistic, s=scalar relativistic). The occupation can be modified, either to create a certain electronic configuration or electrons can be removed to calculate ionization energies. The status can also be changed to create special electronic configurations. The end result of a bulk calculation is virtually independent of the configuration.

The Makefile will run the *atom* program with the right input data and add the results from the different atomic types to the file $(RUNDIR)/**atomdens** which is used by *rspt*. You should never have to edit this file.

5.1.4 symcof

The file $(RUNDIR)/**symcof** is a symbolic link to the file **symt.out** in the directory $(RUNDIR)/**sym/** which contains the symmetry coefficients and Euler rotations . This file should never be edited. In this file you can find the number of symmetry operations (second line), followed by the symmetry operations (three lines and a comment for each), the number of atomic types and harmonic functions, the atomic positions and Euler rotations for each type followed by the components of the harmonic functions and at the end of the file a list of the dimensions of the harmonics.

5.1.5 spts and tetra

The files **spts** and **tetra** (see below) are where the mesh in k-space is stored . The symmetry generator creates the directory $(RUNDIR)/**bz/** with the two files **cub.inp** and **cub.doc**. In **cub.doc** you find recommendations regarding the choice of the number of mesh points in each direction. The length of the vectors in reciprocal space is inversely proportional to the length in real space

so for some non-cubic cells it could be wasteful to have the same number of mesh points in all directions.

The file **cub.inp** contains the lattice vectors and a translation map. The map can be edited . If you are at the level where you do that kind of calculations you don't need any explanation on how to do that. For the rest of you, don't edit the map.

Change directory to $(RUNDIR)/**bz/** and run the program *cub* as "$(RSPTDIR)/bin/cub". For the name of the group file, keep the default (**symcof**) or enter the name of the file if you chose to use some other file (you might have more than one **symcof** file for different symmetries). Then enter the number of mesh points in each reciprocal direction. Remember the recommendation from **cub.doc**. The total number of mesh points needed depends on many things, e.g. the size of the Brillouin zone, the complexity of the Fermi surface and the precision needed. There are some hints in the output files if the mesh is not dense enough, see Section 5.2.1. The only way to be sure is to test for convergence.

The center of the mesh can be shifted way from the Γ point by giving three numerators and three denominators, one for each reciprocal lattice vector. The denominators can not be set to 0. If you are not going to shift the mesh set the numerators to 0 and the denominators to some non-zero value. If the number of mesh points and the shift are correct answer y to the next question, if not answer no and reenter the values. If you need tetrahedrons (small blocks of reciprocal space used in the so called tetrahedron integration method, see Section 5.1.6) answer yes to the next question, otherwise no. Finally the program will ask if you want to quit. If you have created all the meshes you need, answer yes. If not, answer no and go back and create some more.

The program will create a number of files for each mesh size. The files **cub.k.N** (or **cub.k.N_no** if no shift is applied) contains the so called k-points, i.e. reciprocal mesh points, for mesh size N. In the same way are the tetrahedrons stored in **cub.t.N** and **cub.t.N_no**. Copy or make a symbolic link of the **cub.k_N** file you want to use to $(RUNDIR)/spts. Do the same with **cub.t_N** to the file $(RUNDIR)/tetra. Both **spts** and **tetra** are fixed format files.

The file **spts** has the following structure:

```
(i6)
     1
(2i12)
           8          128       8 8 8 1 1 1 2 2 2
```

It begins with a format statement and an integer. The actual integer is not used but it is kept for backward compatibility reasons. The next line is another format statement and it is followed by the number of k-points in the IBZ, the total number of k-points in the BZ, the number of mesh points in each direction and the applied shift. Then follows a list of all k-points with the coordinates and the reciprocal weights.

```
  .062500000000000  .062500000000000  .062500000000000     8
  .187500000000000  .187500000000000 -.062500000000000    24
  .312500000000000  .312500000000000 -.187500000000000    24
  .437500000000000  .437500000000000 -.312500000000000     8
  .312500000000000  .062500000000000  .062500000000000    24
```

```
.437500000000000  .187500000000000 -.062500000000000    24
.187500000000000  .187500000000000  .187500000000000     8
.312500000000000  .312500000000000  .062500000000000     8
```

The file **tetra** follows a similar pattern but instead of the coordinates of the corners defining the tetrahedrons it just contains the numbers (from the file **spts** of the k-points in the four corners and again the reciprocal weight.

5.1.6 data and dataForm

The file **data** is the main input file to *rspt*. It contains all the information needed for the calculation except for the atomic density, symmetry coefficients and mesh points. The file is in fixed format. All the information in the file is created by the symmetry generator but you might have to edit the file to adopt it to your needs. The directory $(RUNDIR)/**dta/** is created by the symmetry generator and in that directory you will find a lot of files. These files are pieces of the data file that are put together by the Makefile in the $(RUNDIR)/**dta/** directory. The files **filename_form** are skeleton files used by *make*.

You can either edit the **data** file directly or the different files in $(RUNDIR)/dta but any change in **data** will be overwritten by *make data*. A common practice is to replace some parameter with a unique string that is easy to substitute using regexp . This can be used if you need to change some parameter other than the ones covered by the different versions of the *runs* programs. A shell script can e.g. be used to test different muffin-tin radii using *sed* in a loop.

Some parameters are marked with a dot. That means that the parameter is not usually used except by developers or advanced users. The value can and should be left as it is.

In alphabetical order, line by line:

- **bravais_lattice**: a format statement and the transpose of the Bravais lattice defined in symt.inp

```
(/ (3f18.0))
R transpose
 .500000000000000  .500000000000000  .000000000000000
-.500000000000000  .500000000000000  .000000000000000
 .000000000000000  .000000000000000 1.00000000000000
```

- **brillouin_zone_integration**: used for choosing the method for Brillouin zone integration.

```
(i6)
    1
(/ 2f12.0, i6)
      W(Ry)         dE/W      .
      0.015           4.      0
```

  The first integer chooses method: 0 = temperature (Fermi) smearing, 1 = tetrahedral interpolation (default), 2 = Gaussian smearing.

```
(/ 2f12.0, i6)
      W(Ry)         dE/W      .
      0.015           4.      0
```

The next three lines are only used for temperature and Gaussian smearing. The first number is the width of the smearing. In the case of temperature smearing it is the temperature times the Boltzmann constant. The next number is the cutoff of the smearing (for Gaussian smearing) in standard deviations (in other words, in units of the smearing width). The smallest possible temperature smearing is 6.33×10^{-6} Ry (way less than 1 K).

The flag prnt(6) (see below) modifies the integration method. If prnt(6) = .true. (default .false.) Blöchl's correction is not used in the case of tetrahedral interpolation and an adaptive smearing width will be used in the case of Gaussian smearing. If adaptive smearing is used the information is stored in the file **smeardata** between iterations .

- **contour_plot_data**: not used by default, used for plotting the charge density in a plane cut in the unit cell . See Section 6 for a description.
- **element_N.X**: the atomic configuration for each type of atom. N is the atomic number and X is the identification tag used in **symt.inp**.

```
(// i6, 2f12.0, 5x, a1 )
    n            S           dx coord
  623  .693835720        0.025     v
  623  2.80000000        0.025     a
```

A format statement followed by the number of radial mesh points inside the muffin-tin, the muffin-tin radius, the logarithmic step size of the mesh and a tag related to the scaling of the muffin-tin radius. The default value is "a" = absolute value no scaling, "r" = scaling by lattice length scale parameter and "v" = scaling by volume per atom. Notice the two leading slashes in the format statement. This means that the first two lines after the format statement are not read. If you prefer to use the first definition of the muffin-tin radius move the second slash to the end of the format statement.

```
(/f6.0, i6, f12.0, 2f6.0, 2x, 2i1, f2.0, 11x, a1//(2i6, f12.0, i12))
    Z    nc      Sinf/S     . Sws/S  ....     ScoFlag
 92.    24          1.    1.    1.  10.0          v
```

The first number is the atomic number and the second the number of core states to be read below. The third number is the radius used in the core state calculation. The last tag, the `ScoFlag` controls the meaning of the core state radius, "a" = absolute radius in the units of the muffin-tin radius, "r" = radius scaled by lattice length scale in units of the muffin-tin radius, "v" = radius scaled by the volume per atom in units of the muffin-tin radius, "m" = equal to the muffin-tin radius. The core state radius must be equal to or larger than the muffin-tin radius. If the radius is smaller it will be set to the muffin-tin radius. The numbers with dots over are not used but they are kept for compatibility reasons. The last number, `Sws/S`, is the ratio between the radius used to extrapolate the potential mesh and the muffin-tin radius. You should very rarely have to change these numbers.

```
 n      k          occ         flag
 1     -1           2.            0
 2     -1           2.            0
 2      1           2.            0
 2     -2           4.            0
 3     -1           2.            0
 3      1           2.            0
 3     -2           4.            0
 3      2           4.            0
 3     -3           6.            0
 4     -1           2.            0
 4      1           2.            0
 4     -2           4.            0
 4      2           4.            0
 4     -3           6.            0
 4      3           6.            0
 4     -4           8.            0
 5     -1           2.            0
 5      1           2.            0
 5     -2           4.            0
 5      2           4.            2
 5     -3           6.            2
 6     -1           .0            0
 6      1           .0            0
 6     -2           .0            0
```

For each core state are the principal quantum number , the relativistic kappa quantum number , the occupation and a flag given. If the flag is set to 2 a SIC correction will be applied to that core state. It is recommended that a few empty core states are added.

```
14 Bases
 0      1      1
 0      1      2
 0      1      3
 1      1      1
 1      1      2
 1      1      3
 2      1      1
 2      1      2
 3      1      1
 3      1      2
 0      2      1
 0      2      2
 1      2      1
 1      2      2
```

Here we start to see some of the unique parts of FP-LMTO, the ℓ-projected basis functions . The first line is the number of basis functions to be read. For each basis function the ℓ quantum number, the energy set and tail

are read. The data set above includes three s-states belonging to the first energy set with tail 1, 2 and 3 associated to them respectively. Next come three p-states, two d-states and two f-states, all belonging to the first energy set. Last come two s-states and two p-states belonging to the second energy set. Note that there can be more than one basis functions with the same ℓ and tail as long as they belong to different energy sets. There can also be more than one basis function with the same ℓ within a energy set as log as the tail differs.

The default configuration uses two energy sets but you can use any reasonable number of sets. You can also use any number of tails but three to five tails is a common choice. The default setup can handle most cases but there is no good rule how to choose basis functions if the default setup is not good enough. You just have to test your configuration and see if it is flexible enough without becoming linear dependent. See the section about the **out** file below.

```
  7     7     6     5     5     6     7     8     9
 20    21     0     0     0     0     0     0     0
  7     7     6     5     5     6     7     8     9
 20    21     0     0     0     0     0     0     0
  6     6     6     5     5     6     7     8     9
  0     0     0     0     0     0     0     0     0
  6     6     6     5     5     6     7     8     9
  0     0     0     0     0     0     0     0     0
```

Finally we have the section where the principal quantum number for each ℓ and energy set are defined. For each energy set and spin (in this case two, we have a spin polarized configuration) one line gives the principal quantum number for ℓ going from 0 to `lmax` . Here we see that in the first energy set we have 7s, 7p, 6d, 5f and so on for spin up (and then once again for the second spin). The second energy set has 6s, 6p, 6d, 5f and so on. The principal quantum numbers for the same ℓ in different energy sets must not be the same for states with explicit basis functions defined above. In the example above we can have 6d in both sets because there are no ℓ=2 basis functions in the second energy set but the s- and p-states must have different principal quantum numbers (7 and 6 respectively) as we have ℓ=0 and ℓ=1 basis functions in both energy sets.

The second line for each energy set and spin defines how the basis functions should be constructed or constrained, the so called flags. The default value is 2X for the first energy set (which in the default setup is the valence energy set) for the s- and p-states. X is the l quantum number. This means that the basis functions should be orthogonalized to the basis functions in energy set 2 (which in the default setup is the pseudo core) with the ℓ=X. In our example the 7s states should be orthogonal to the 6s states in the second energy set and the 7p states should be orthogonal to the 6p states.

The flags can be set to a few other values by the advanced user.

- **fourier_grid**: the grid for the fast Fourier transformation and the choice of diagonalizer.

```
(/ 3i6, 4x, 2i1, 2i6)
  ft1   ft2   ft3    ..  diag     .
   32    32    45    00     1     0
```

The number of mesh points needed in each direction depends on several parameters as the Fourier grid is used to define both the number of basis functions in the interstitial as well as the quality of the actual transformation. A longer vector will need more grid points than a short so the three numbers might be different and a complex structure with several atoms will need more grid points than a simple structure. The convergence with regards to the number of grid points can be checked in the **out** file, see Section 5.2.1.

The possible choices for the diagonalizer is 0-3. The result should be the same to within the numerical precision.

- diag=0: The Hamiltonian is transformed to a tridiagonal matrix and solved using the method by H. Bowdler et. al. "The QR and QL Algorithms for Symmetric Matrices", H. Bowdler et. al., Numer. Math. 11, 293 (1968).
- diag=1 (default): diagonalize the Hamiltonian using the LAPACK solver ZHPEV.
- diag=2: diagonalize the Hamiltonian using the LAPACK solver ZHEEV.
- diag=3: diagonalize the Hamiltonian using the LAPACK solver ZHEEVD. Uses another method to calculate the eigenvectors than ZHEEV that should be faster but it is reported to be unstable in some rare cases.

- **header**: controls the cutoff in ℓ, the number of different atomic types, number of valence electrons, the XC functional, mixing factor, warping and whether the calculation should include spin-orbit interaction and spin polarization

```
(/ 2i6, f6.0, 2i6, 2f6.0, 4l6, i6)
 lmax ntype  zval     . icorr     .  pmix   win   wmt f-rel sp-po     .
    8     3  48.0     6    24    1.  .200     t     t     f     t     1
```

  - `lmax`: the cutoff in ℓ for the basis function expansion. Should be the same as in *symt*. `lmax`=8 is (almost) always enough. If the occupation is very low for the highest ℓ, `lmax` can be lowered to speed up the calculation but test first. Note, through the folding of the interstitial basis functions into the muffin-tins the occupation can be non-zero even for states with ℓ higher than the highest ℓ used in an explicit basis function i.e. one defined in the element_N part of the **data** file.
  - `ntype`: the number of symmetry groups. There will be at least one for each kind of atom but there can be more than one for similar atoms belonging to different symmetry sub-groups. The total number of types is calculated by *symt* and can be found in the file **symcof**.
  - `zval`: the number of valence (and pseudo core) electrons in the unit cell. Should be the same as `zwin` in **panel_data**. Technical: there are a few very special situations where the two can be set to different values, therefor both are kept in the **data** file.

- `icorr`: the XC functional. The second digit can be either
  - 0= exchange only
  - 2= Hedin-Lundqvist: vonBarth-Hedin interpolation, RPA scaling
  - 3= Ceperly-Alder parametrized by Perdew and Zunger
  - 4= Perdew and Wang 1992

  and the first digit
  - 0= LDA
  - 1= PW 91 gradient corrections
  - 2= PBE 96 gradient corrections
  - 3= AM05 Note, AM05 only with Perdew and Wang 1992
  - 4= PBEsol gradient corrections
  - 5= Revised PBE gradient corrections
- `pmix`: the mixing factor used in the Broyden mixing. The default value might be to high for complicated systems. If the total energy oscillates, try a smaller mixing.
- `win` and wmt: technical, warping of the basis
- `f-rel`: spin-orbit interaction
- `sp-po`: spin polarization

- **length_scale**: the length used to scale the lattice vectors, in atomic units. The default value is the equilibrium length for the elements and a weighted combination (that must be checked and corrected) for all other materials.
- **panel_data**: controls the behavior of the tails.

```
(/ i6, 3x, 3i1, i6, 2f6.0, f12.0 /)
 ntls   ... nsets  zwin     .   log(epst)
    3   111     2  48.0    .0        -38.
        T^2   ...  ?   ... sc t^2 = a/S^2         + b
(f12.0, f5.0, i1, i3, 3x, 3i1, l3, f6.0, i3, 6f12.0)
 .300000000   .01  0   101  f    .0  0          .0
-2.30000000   .01  0   101  f    .0  0          .0
-.600000000   .01  0   101  f    .0  0          .0
(3f6.0)
   .0    1.    1.
```

After one line with the format and one line with comments the number of tails, the number of energy sets, the interstitial charge and a convergence parameter are set. The number of tails must match the number of lines giving the tail energies below. All tails must not be used in the definition of the atoms. The default number is often good but there are cases where you might need to add one or more tails, see Section 5.2.1.

The number of energy sets depends on if you need a pseudo core or not. The default setup is with a pseudo core. You can have more than two energy sets but the number must be the same for all atomic types. Note that you don't need to have explicit basis functions for all sets. E.g. for

hydrides the hydrogen pseudo core can be left with no basis functions but the the lines in the **element** file must still be there.

The number of electrons, `zwin`, should almost always match the number `zval` in the **header** file. There is a technical difference between the two numbers for the advanced user to explore. Finally we have a convergence parameter controlling the matching between the interstitial and muffin-tin basis functions. A higher absolute value results in more functions being used in the matching (but not in the actual calculation). The default value is a minimum and if you are in doubt, increase the absolute value.

Again there are two lines with formats and comments. After that there are `ntls` lines defining the tails. The first number is the tail energy followed by a real written together with an integer (see the format, the default values are 0.0 and 1, not 0.01). The real parameter controls how the structure functions are created and can be left as 0.0. The integer is not used by default. The next four integers control how the tail energy behaves. The first integer of the four defines the behavior and the next three give the finer details. The default value, 0, means a constant energy and it will give very good results in almost all cases as long as the tail energies are chosen with some care. One possible exception is EOS calculations at extremely high compressions*. In that case, see the source file **sentls.F** for details.

The next parameter, a logical, controls whether the tail energy should be scaled with the length scale. Finally the real, integer and final real parameter is used when the tail energy is set to a polynomial, see above.

After a final format statement follow three real parameters. The first two set the scaled min and max cutoff when eigenvalues are connected to tails after the diagonalization and the third is a scaling factor used in setting the tails. You don't have to bother about this unless you are a very advanced user.

- **prnt_array_data**: the content of this file is a 72 character long string of flags which controls most of the features .

```
    .    1    .    2    .    3    .    4    .    5    .    6    .    7
fftfffffftfffffffffftftfffffffftffffftfffffffftfffffffffffffftfffffttffff
```

The flags can be either true (t) or false (f), leading to the following behavior. The definitive answer to what a prnt flag does is the source code file drspt.F where all changes to the flags are documented.

```
c Descriptions of prnt_array flags along with their default values as
c generated by symt. Two default values indicate that different values
c may be generated by symt depending on choice of setup.
c
c def  |   flag     |  Description
c------|------------|-----------------------------------------------------
c    F   prnt( 1  ) => print new potentials
c    F   prnt( 2  ) => print eigenvalues
c  T     prnt( 3  ) => write mixed potentials on disk
c    F   prnt( 4  ) => print mixed potentials, strained parameters, ...
c    F   prnt( 5  ) => only calculate eigenvalues, then stop
```

*In that case you will also have to think about the choice of functional. Most functionals don't take the kinetic energy into account when calculating the XC energy and that contribution can be big at TPa pressure. AM05 is an exception.

```
c    F   prnt( 6  ) => if tetrahedra, don't apply Blochl's correction
c                      if smearing, use adaptive smearing width
c    F   prnt( 7  ) => muffin-tin density from density matrix
c    F   prnt( 8  ) => interstitial density from density matrix
c  T     prnt( 9  ) => continue if interstitial dens is < 0 or complex
c    F   prnt(10  ) => print charge densities
c    F   prnt(11  ) =>  Average density matrix over spins to get LDA
c                      instead of LSDA in a CSC-DMFT run.
c    F   prnt(12  ) => print eigenvectors after back transformation
c    F   prnt(13  ) => zero spin-orbit matrix elements on p states
c    F   prnt(14  ) => restart jacobian matrix with pmix
c    F   prnt(15  ) => no hybridization through spin-orbit matrix
c                      elements between PQN sets
c                      -> basisPQNsetHybThroughSO = 0
c    F   prnt(16  ) =>
c    F   prnt(17  ) => calculate forces
c    F   prnt(18  ) =>
c  T F   prnt(19  ) => read strain matrix and strain lattice
c    F   prnt(20  ) => calculate angular points and print to file
c  T     prnt(21  ) => calculate and store gaunt coefficients
c    F   prnt(22  ) => controls mixing of potential and energies
c  T     prnt(23  ) => controls mixing of potential and energies
c    F   prnt(24  ) => Keep more things in memory. This flag makes the
c                      code avoid throwing away and later reinitialize
c                      some modules. This should be faster.
c    F   prnt(25  ) => Lanczos damp bases, new interstitial
c    F   prnt(26  ) => Lanczos damp bases, old interstitial
c    F   prnt(27  ) =>
c    F   prnt(28  ) =>
c    F   prnt(29  ) =>
c    F   prnt(30  ) =>
c    F   prnt(31  ) =>
c  T     prnt(32  ) => calculate pseudo-atom density exponents
c    F   prnt(33  ) => generate starting potential
c    F   prnt(34  ) => set pseudo-atom density exponents to
c                      Madelung constants (disabled)
c    F   prnt(35  ) => print Ewald (Madelung) constants
c    F   prnt(36  ) => Plot Fermi surface in XCrySDen format
c    F   prnt(37  ) => write plot of V and N between neighbors
c  T     prnt(38  ) =>
c    F   prnt(39  ) => Perform SIC calculation
c    F   prnt(40  ) => MLM?
c    F   prnt(41  ) => MLM?
c    F   prnt(42  ) => Orbital polarization
c    F   prnt(43  ) => LDA+U (soon)
c    F   prnt(44  ) =>
c  T     prnt(45  ) => check arrangement of interstitial potential
c    F   prnt(46  ) => if (prnt(66)) write contour plot of atomic
c                      densities
c    F   prnt(47  ) =>
c    F   prnt(48  ) => use exponents read from data for pseudo-atom
c                      charge densities
c    F   prnt(49  ) => stop if radius or dx mismatch reading pot
c    F   prnt(50  ) => allow unequal number of electrons and protons
c                      (uniform background added).
c    F   prnt(51  ) => calculate dielectric matrix
c    F   prnt(52  ) => stop before density (just do eigen block +
c                      eigenvalue postprocessing)
c    F   prnt(53  ) =>
c    F   prnt(54  ) =>
c    F   prnt(55  ) =>
c    F   prnt(56  ) =>
c    F   prnt(57  ) =>
c    F   prnt(58  ) => if (prnt(51) don't include interstitial
c    F   prnt(59  ) => if (prnt(51) don't include muffin tins
c    F   prnt(60  ) => plot core orbitals
c  T     prnt(61  ) => output to 'out' instead of STDOUT
c    F   prnt(62  ) => manually set number of eigenvalues to keep
c    F   prnt(63  ) =>
c    F   prnt(64  ) => if prnt(65) also output pdos
c    F   prnt(65  ) => output dos and nos
c    F   prnt(66  ) => calculate charge density contour plot
c  T     prnt(67  ) => try to diagonalize the eparm matrix
c  T     prnt(68  ) => eigenvectors, eigenvalues, and weights to
```

```
c                       permanent files, rather than scratch.
c    F   prnt(69  ) => non-basis enu's = average over basis enus
c    F   prnt(70  ) =>
c    F   prnt(71  ) =>
c    F   prnt(72  ) =>
```

- prnt(1-4): controls what to write to files, does not affect the actual calculation.
- prnt(5): if you only need the eigenvalues you can stop after they have been calculated.
- prnt(6): controls how the BZ integration should be performed. The Blöchl correction should never be switched off unless you really know what you are doing. If you use Gaussian smearing the adaptive smearing can give you a better description of the Fermi surface.
- prnt(7-8): there are different ways to calculate the density but this is more a technical issue than a practical. See the source code if you need to change prnt(7) and prnt(8).
- prnt(9): The density should never be less than zero in a physical system but due to the numerical methods used this may happen. Set prnt(9) to T if you for some reason want to stop if this happens.
- prnt(10): controls whether the charge density should be printed. If true the core density will be written to the file **cordens** and the valence density to **valdens**. This is not the same as a charge density contour plot (see prnt(66)).
- prnt(11): Symmetrize the density matrix over spin. This has the effect of making a self-consistent LDA+DMFT run use LDA rather than LSDA.
- prnt(12): the Hamiltonian matrix is Cholesky decomposed before the diagonalization. This will destroy the ordering of the eigenvectors but it can be restored by a back transformation. prnt(12) controls whether the eigenvectors should be written to file or not after the back transform. This flag is similar to prnt(5) but the result is pretty printed to the file **eigenvectors**.
- prnt(13): RSPt uses a lm-basis set and not a jj-set. In some very special cases this can cause some problems with the description of the spin-orbit coupling for pseudo core p-states. You can turn the spin orbit interaction off for the pseudo core p-states if you ever experience this problem.
- prnt(14): technical stuff, don't change this flag unless you know what you are doing.
- prnt(15): if you know what a Primary Quantum Number (PQN) set is you are allowed to change prnt(15).
- prnt(16): not used.
- prnt(17): RSPt can calculate non spin polarized forces (not yet with spin-orbit coupling) including the IBS contribution with high accuracy. The result in Ry/Bohr is written to the file **forces**. Note, the force calculation is rather time consuming so converge first with prnt(17) set to F and then do one iteration with prnt(17) set to T.

- prnt(18): not used.
- prnt(19): controls whether the strain matrix should be read and used. If set to T the strain matrix must be included in the **data** file and it can not be included when set to F.
- prnt(20): controls whether the angular points should be written to the file **apts**. If the file does not exist and prnt(20) is set to F it will be changed to T.
- prnt(21): not used, gaunt coefficients are always calculated. Kept for compatibility reasons.
- prnt(22-23): technical stuff controlling how the energy parameters should be updated. Don't touch unless you know what you are doing.
- prnt(24): Controls the memory allocation for a few big arrays. T will give a faster code with a bigger memory foot print and F a slower and smaller code.
- prnt(25-26): The Lanczos method can be used to damp unneeded bases in the interstitial (which can be calculated in two ways).
- prnt(27-31): not used.
- prnt(32): the initial potential used in the first iteration can be generated from a pseudo atom in different ways, see also prnt(48).
- prnt(33) : a new potential from pseudo atoms can be created even if the file **pot** with an old potential exists, see also prnt(32) and prnt(48).
- prnt(34): not used, see prnt(32) instead.
- prnt(35): controls whether Madelung constants should be written to the **out** file. Does not affect the actual calculation.
- prnt(36): controls whether the Fermi surface should be plotted, see Section 6.6.
- prnt(37): controls whether the potential and density along lines connecting nearest neighbors should be printed to the files **vplot.N** and **dplot.N** (N over spin channels) respectively. Very useful when choosing muffin-tin radii.
- prnt(38): not used.
- prnt(39): Self-interaction correction (SIC)
- prnt(40-41): MLM. MLM is in the code. Read the source code if you ever plan on using it.
- prnt(42): orbital polarization
- prnt(43): place holder for LDA+U (almost done).
- prnt(44): not used.
- prnt(45): technical, controls whether some checks should be performed on the potential.

- prnt(46): if prnt(66) and prnt(46) are both true the atomic densities will be written to the files **atomic_density_contour** and **atomic_magnetization_contour**. The format of the output can be IDL (output=1, default), gnuplot (output=2) or xyz (output=3). For now output must be set in the source code file **denplt2.F**.
- prnt(47): not used.
- prnt(48): see prnt(32).
- prnt(49): potential files can be reused in some cases even though the setup has been changed. prnt(49) controls whether the radial grid should be adapted to a new muffin-tin radius or if the program should stop.
- prnt(50): controls whether explicit charge neutrality should be enforced. The default behavior is to stop if the the number of electrons defined in **header** and **panel_data** (valence) and **element_N** (core) differs from the total number of electrons expected from the atomic configuration. If prnt(50) is set to T you can choose a different value for the number of valence or core electrons. A compensating jellium background, positive or negative, will then be added automatically. This can e.g. be used to simulate a system where the temperature is higher than the binding energy of the highest occupied states if the kinetic energy of the jellium background is taken into account.
- prnt(51): controls whether the dielectric matrix should be calculated. Either of the interstitial or muffin-tin regions can be excluded, see prnt(58) and prnt(59). The calculation is somewhat cumbersome and a high number of k-points is needed for good results so it is recommended that prnt(51) is set to T for one iteration starting with a highly converged solution.
- prnt(52): in some cases only the eigenvalues and eigenvectors are needed. Set prnt(52) to T to stop after the possible calls to the force, optical, bcoop or DMFT calculations and before the density is calculated.
- prnt(53-57): not used.
- prnt(58): If prnt(51) is set to T and the dielectric matrix is calculated the interstitial region is excluded.
- prnt(59): If prnt(51) is set to T and the dielectric matrix is calculated the muffin-tin region is excluded.
- prnt(60): controls whether the core orbital density should be printed to file. If set to T the densities for each type and orbital are written to the files **core_orbital.N.M** where N is over types and M over orbitals.
- prnt(61): The file **out** is the default output file. The output can be redirected to STDOUT instead by setting prnt(61) to F. This assumes that file sequence number 6 is reserved for STDOUT on your computer.
- prnt(62): obsolete but not removed. Mxinp eigenvalues can be disregarded in the calculation to reduce the size of the memory footprint.

This used to be an issue long ago when computers had much less memory. Mxinp is defined in the file **fourier_grid** where it is the last number with a default value of 0.

- prnt(63): not used.
- prnt(64): If prnt(65) is set to T and the density of states is calculated and printed prnt(64) controls whether the partial density of states also should be calculated. The partial density of states is printed to the files **PDos.1.N.M**, where N is over types and M over spin, **PDos.1.int.M** and **PDos.1.hea**. See also Section 6.4.
- prnt(65): If set to T the density of states is printed to the file **Dos.1.0**, see Sections 6.4 and 6.4.1.
- prnt(66): controls whether a charge density contour plot should be printed. See section 6.1.
- prnt(67): technical. Controls the way the eparm matrix behaves.
- prnt(68): the default behavior is to keep the eigenvalues and eigenvectors in the permanent files **va1**, **ve1** and **wt1**. If the eigenvalues and eigenvectors are not going to be used in some way after an iteration is finished the files can be opened as scratch files instead. Scratch files are in most cases, depending on the compiler and operating system, removed automatically when the program which created the files is done.
- prnt(69): technical. Really technical.

The default choices are usually very good. Some of the flags will include features that are only meaningful for a converged solution and might take a long time to finish and should therefor only be used for single iterations at the end of the calculation.

- **spin_data**: In the case of a spin polarized calculation (non-zero spin axis in **symt.inp** the file **spin_data** is created.

```
(/ 3f18.0, 5x, a1 // f18.0, i24, f18.0)
                          axis                          coord
 .000000000000000  .000000000000000  1.00000000000000     1
   field(Ry/bohr)   field modulation star    spin mix ratio
             .000                      1              .10
```

The file contains the spin axis , an optional external field that can be used to nudge the solution towards a spin polarized state, a technical parameter and a parameter that controls the spin mixing. The spin is mixed separately from the density and the mixing parameter can be changed depending on the convergence of the spin state.

- **strain_matrix**: Depending on prnt(19) the strain matrix must be included or not. The default value is T and the strain matrix is the identity matrix. The strain matrix can be used e.g. when calculating elastic constants.

5.1.7 fixmom.inp, momfix.dat

If either of the files **fixmom.inp** or **momfix.dat** are present, a fixed spin moment calculation will be performed. The file **fixmom.inp** is intended to be the start file written by the user and **momfix.dat** is generated by the program and contains additional information stored between iterations. The files have the same format and may contain exactly the same things. The format is a set of named blocks of data and with the exclamation mark character (!) signalling that the rest of the line is a comment.

A basic **fixmom.inp** contains just information about what moment to fix and the desired value in a `constraint` block:

```
constraint
 0
-0.50
```

This constraint fixes the total moment to -0.5 μ_B. The first number in the constraint block says what to constrain according to

0 Constrain the total moment.

-1 Constrain the interstitial moment.

n Any positive integer: the moment inside the muffin-tin of type n.

This is followed by a real number that specifies what value to constrain the moment to.

The next type of block is the `PID-data` block. It contains information about the PID controller (see Section 12.1).

```
 PID-data
           3
  0.000000000000000E+000  0.000000000000000E+000  0.000000000000000E+000
  5.000000000000000E-003  1.000000000000000E-003  5.000000000000000E-002
```

The first integer again gives what type of constraint this data block refers to. The second line is the current controller output (e_n, $\sum_{i=1}^{n} e_i$ and $(e_n - e_{n-1})$ from Equation (55)). The third line is the different gains (K_P, K_I and K_P).

The last type of block is the `field` block.

```
 field
           -1
  1.999811597295014E-004 -1.490893901588703E-004
```

It contains (again) an integer specifying to which constraint this field belong, followed by a line with the current value of the constraining field and the energy correction to be added this iteration.

As previously indicated, only the `constraint` block is needed in **fixmom.inp**, but the two other blocks may also be put there if the user has some guess of good starting values to put in.

5.2 Output files

5.2.1 out and out_last

The name of the main output file is **out**. Here all information about the last electronic iteration is written. If you use one of the *runs* programs, the file **out_last** will be the **out** file of the second to last iteration. RSPt is a relatively talkative code, so there is very much information in there. The information that a user is ultimately interested in such as total energy will of course end up near the end of the file, and most of the rest are technical parameters that tell you things about the calculation. Here follows a list of some of the parameters that are important to check to see whether the calculation produces useful results. In this example the **out** file comes from a calculation of UO_2.

Important parameters

Nearest neighbors A good thing to first search the **out** file for is the string `Nearest neighbors`. This will take you to a section that looks something like this

```
 Nearest neighbors:
    (closest site within .777817 * L)
   t  t(N)              d/L             2S/d     d(B)     d(A)
   1     3   .43301270189221   .94817645965300  4.53502  2.39982
         3   .43301270189221   .94817645965300  4.53502  2.39982
         3   .43301270189221   .94817645965300  4.53502  2.39982
         3   .43301270189221   .94817645965300  4.53502  2.39982
         2   .70710678118654   .75617858538974  7.40565  3.91890
         1   .70710678118654   .75617858538974  7.40565  3.91890
   2     3   .43301270189221   .94817645965300  4.53502  2.39982
         3   .43301270189221   .94817645965300  4.53502  2.39982
         3   .43301270189221   .94817645965300  4.53502  2.39982
         3   .43301270189221   .94817645965300  4.53502  2.39982
         1   .70710678118654   .75617858538974  7.40565  3.91890
         2   .70710678118654   .75617858538974  7.40565  3.91890
   3     1   .43301270189221   .94817645965300  4.53502  2.39982
         2   .43301270189221   .94817645965300  4.53502  2.39982
         3   .50000000000000   .57289179162550  5.23659  2.77108
         3   .50000000000000   .57289179162550  5.23659  2.77108
         3   .70710678118654   .40509567074450  7.40565  3.91890
         3   .70710678118654   .40509567074450  7.40565  3.91890
```

This table gives the information about the nearest neighbors of each type. The type index is the leftmost number in the table. For each type a number of atoms close in space are listed. Their index go in the second column. Column three, five and six contains the distance between these atoms in units of the length scale, Bohr radii and ångstroms, respectively. Column four contains a very important number. It gives the sum of the muffin-tin radii of the atom pair in units of the distance between the atoms. If this number is equal to or larger than 1, the muffin-tins overlap. This must *never* happen. If you find that this is the case, you must adjust the muffin-tin radii of the species that caused the overlap.

Energy parameters Go on to search for the phrase `Energy parameters`. Here is a somewhat truncated form of what you will find:

```
     Energy parameters ...                                    energy reset
                                                              | npr(mt) .ne. npr
                                                              | | e < e1(ws)
       t  e  l  spin  n mt          E       D(mt)  | | | e > e2
       1  1  0  down  7  7   2.4986612      -5.366  * - - *
                  up  7  7   2.4986612      -5.170  * - - *
             1  down  7  7   3.6050245      -3.397  * - - *
                  up  7  7   3.6050245      -3.263  * - - *
             2  down  6  6   0.6185360      -1.039  - - - -
                  up  6  6   0.6185576      -0.928  - - - -
             3  down  5  5   0.4933281      -1.474  - - - -
                  up  5  5   0.4937560      -0.601  - - - -
          2  0  down  6  6  -2.5855297    -102.007  - - - *
                  up  6  6  -2.5786879     -10.386  - - - *
             1  down  6  6  -0.7840840      -4.898  - - - *
                  up  6  6  -0.7784670      -3.897  - - - *
             2  down  6  6   0.5656665      -0.869  - - - -
                  up  6  6   0.5659293      -0.765  - - - -
             3  down  5  5   0.4537840      -0.878  - - - -
                  up  5  5   0.4550889      -0.235  - - - -
```

This contains information about the linearization energies (the `E` column) for each type (`t`), energy set (`e`), ℓ quantum number (`l`) and spin. To the right, there are four columns of either the symbol $\star$, if something has happened or $-$, if it hasn't. The most important thing to watch out for here is stars in the second column. That indicates that the basis function does not have the correct number of nodes for an ℓ state of the prescribed principal quantum number. If that happens you should put some constraint on your linearization energies to prevent this from occurring.

Diagonalization The next important block to check is the diagonalization block. Search for `complex diagonals` . In a perfect world the eigenvalues should be real but there is (almost, you could be the luckiest person in the world) always a small complex component.

```
*** warning in eigen: complex diagonals
  k vectors                    extremum
          2     ( 6.262E+00,  2.882E-12)
TIME: EIGEN:      0.038m in,      0.163m added,      0.202m total

Cpulog for EIGEN:
            FC+VI           FC junk              STRC           O+T Int
           .10961            .00003            .00521            .00068
           O+H Mt              Diag              <I>
           .03671            .00635            .00030
SPACE: EIGEN:      963408I in,      602496I added,     1565904I total


SPACE: eigen0:      959852I in,        4776I added,      964628I total

statistics
              zsk               emin                  z              emax
 .000000000000000 -2.68898059676820  47.9999999999999  .312487752963912


         low band         high band
```

```
               1                48
 Eigenvalue sum: -21.4940149262486
 fermi energy =  3.1248775296391E-01
       D(ef) =  0.0000000000000E+00
```

The complex component should be less than 10^{-8}-10^{-9}. Values higher than this are most often caused by a bad choice of basis functions. Try and change the one or more tail energies (to know which one takes some practice) or use different tail energies for different atomic types. The latter is most important for pseudo core states. You might also see complex diagonals if you have way too few k-points.

The two results `emin` and `emax` are the lowest and highest eigenvalue respectively and especially `emin` can be checked for inconsistencies. If you have a pseudo core state that is significantly lower in energy than the tail energy used for that state the solution might be less than optimal. Exactly which state the lowest eigenvalue belongs to can be inferred from other output in the file, see below.

`D(ef)` is the density of states at the Fermi energy and as we can see the example we are using is an insulator as expected.

Charge summary Further down in the **out** file there is a summary of the charge distribution. Search for `Muffin tin charge:` and `Interstitial charge:`. The muffin-tin charge is per atomic type and spin

```
Muffin tin charge:
  type  spin                Q
     1  down   4.5556024776667
     2  down   4.5169074330171
     3  down   2.5236735951909
 total  down   19.167204291447

     1    up   4.5169074370493
     2    up   4.5556024735240
     3    up   2.5236735952949
 total    up   19.167204291753
```

and the interstitial charge is only spin resolved (charge in the interstitial cannot be connected to a specific type).

```
Interstitial charge:
        spin                Q             dQ/Q              fix
        down  4.83279570855208 -.000366150001516  1.00036628411644
          up  4.83279570824664 -.000366150002216  1.00036628411714
```

The total charge for different types but of the same element (like type 1 and 2 above, both are U) should be about the same and the charge for each spin should be the same unless you have an anti-ferromagnetic solution. The error in the interstitial charge, `dQ/Q`, should be small like of the order of 10^{-4}.

Fourier transform convergence Between the muffin-tin and interstitial charge summary you can find the convergence summary for the Fourier transform .

```
Fourier transform parameters for panel  1:
radii and arguments:
1.5000000     5
2.7999999     6
```

The second argument is very important as it relates to the completeness of the basis function expansion in the interstitial. There is a highly technical but exact argument saying that this number should always be 6 or higher. Take our word for it, if the number is 5 or lower, go back and redo the calculation with a denser FFT mesh. It is good practice to check this number after one or two iterations so you don't waste time. In our example, the mesh is not dense enough.

ℓ-projected averages The next important thing to look for are the ℓ-projected averages. Search for `L-projected averages`. The ℓ-projected averages give occupancies and average energies for various parts of the basis sets. Below is a part of the output for the first type shown.

```
L-projected averages ...
  Panel  1 ...
     Muffin tins
       t  l     e  spin             E           dE/E             Q
       1  0     1  down      0.0668560    -29.1425241     0.0338447
                    up      0.0688046    -28.2884943     0.0332087
               2  down     -2.5720505     -0.0052406     1.0342438
                    up     -2.5109763     -0.0269663     1.0322616
            total  down     -2.4884312                    1.0680884
                    up     -2.4305694                    1.0654703
          1     1  down      0.1854683    -14.7246896     0.0074882
                    up      0.1835407    -14.8897210     0.0077316
               2  down     -0.7820887     -0.0025513     2.9238145
                    up     -0.7313629     -0.0644060     2.9147211
            total  down     -0.7796170                    2.9313028
                    up     -0.7289424                    2.9224527
          2     1  down      0.1009543     -5.1268941     0.3832865
                    up      0.1014686     -5.0960480     0.3618445
               2  down     -0.1676528      4.3740351     0.0182559
                    up     -0.1663758      4.4015114     0.0184390
            total  down      0.0887422                    0.4015424
                    up      0.0884815                    0.3802836
          3     1  down      0.1333291     -2.7000793     0.1198324
                    up      0.1347667     -2.6637842     0.1139211
               2  down     -0.0800869      6.6661419     0.0030145
                    up     -0.0681846      7.6743656     0.0028867
            total  down      0.1280921                    0.1228469
                    up      0.1297511                    0.1168078
          4     1  down      0.0939289      1.0460008     0.0212811
                    up      0.0958702      1.0428618     0.0213017
               2  down     -0.1330174      0.6388548     0.0003047
                    up     -0.1290178      0.6315067     0.0003049
            total  down      0.0907249                    0.0215858
                    up      0.0926971                    0.0216065
```

The occupancy of the respective ℓ and energy set should be rather close to what is expected. Note that the number of electrons, `Q`, is never exactly the integer value expected from a simplified atomic argument. The contributions to one muffin-tin from all other muffin-tins are added ("folded", see Section 5.1.6) up

to ℓ=lmax. Therefor you will also see some charge in states with no explicit basis function. If the total charge for a specific ℓ and pseudo core energy set is much higher than expected the respective valence energy set state is not orthogonal enough. In that case set the flag in the **data** file accordingly. If that don't solve the problem, increase the difference in energy for the different tails or add tails.

If the basis set is too bad you can get linear dependencies and the program will stop before the diagonalizer. If this is the case change the tails as above.

Boundary matching The Fourier transform mesh density affects not only the quality of the interstitial basis functions but also the matching between the interstitial and the muffin-tin. Search for `Charge densities at boundaries`. Below is the result for the first atomic type.

```
 Charge densities at boundaries
     t      h      n(down)      n(  up)
     1      1   .016512254   .016373907 ... mt
                .016529327   .016390979 ... int
            2  -.000015399   .000014496 ... mt
               -.000015421   .000014601 ... int
            3  -.016175596  -.015948797 ... mt
               -.016350326  -.016124726 ... int
            4  -.013668395  -.013481337 ... mt
               -.013809558  -.013623122 ... int
            5   .005804325   .005736162 ... mt
                .006116057   .006049323 ... int
            6  -.015339061  -.015194304 ... mt
               -.016165479  -.016025965 ... int
            7   .001648743   .001656595 ... mt
                .002076535   .002083619 ... int
            8   .000872605   .000885931 ... mt
                .001086161   .001095894 ... int
            9   .001328211   .001350358 ... mt
                .001663156   .001682514 ... int
```

The difference between the muffin-tin and the interstitial should be less than 10^{-3}-10^{-4} for the first few harmonics (h) for all types. If not, go back and redo the calculation with a denser mesh.

Core states The core states should behave and stay in the muffin-tin. Search for `core states`. The core state energies and occupation are printed for each type.

```
 type:     1
      z             rm        dx             s(i)            rn(i)
   92.0   0.4941728E-06 0.0250000   2.8000000( 623)   2.8000000( 623)
   n  k              e    occ    ctp     dq      g(s)    sicde    sicen
   1 -1 -8506.758586909  2.000  0.019 0.0000  0.000000
   2 -1 -1588.203987140  2.000  0.078 0.0000  0.000000
   2  1 -1531.451011226  2.000  0.080 0.0000  0.000000
   2 -2 -1250.254369893  4.000  0.093 0.0000  0.000000
   3 -1  -400.338296892  2.000  0.198 0.0000  0.000000
   3  1  -374.565887725  2.000  0.208 0.0000  0.000000
   3 -2  -309.780860042  4.000  0.230 0.0000  0.000000
```

```
   3  2  -268.970276804  4.000  0.248 0.0000  0.000000
   3 -3  -255.824589234  6.000  0.254 0.0000  0.000000
   4 -1  -101.391403790  2.000  0.419 0.0000 -0.000000
   4  1   -89.798052831  2.000  0.451 0.0000  0.000000
   4 -2   -72.885662144  4.000  0.499 0.0000  0.000000
   4  2   -54.421266663  4.000  0.580 0.0000 -0.000000
   4 -3   -51.311966444  6.000  0.594 0.0000 -0.000000
   4  3   -27.026855652  6.000  0.782 0.0000  0.000003
   4 -4   -26.216492657  8.000  0.802 0.0000  0.000004
   5 -1   -21.969218198  2.000  0.865 0.0000  0.000100
   5  1   -17.461556141  2.000  0.956 0.0000 -0.000256
   5 -2   -13.428517185  4.000  1.056 0.0000 -0.000853
   5  2    -6.834581287  4.000  1.356 0.0000  0.004844
   5 -3    -6.234731228  6.000  1.390 0.0000  0.006185
   5  3     0.060595093  2.000  2.292 0.0000 -0.085265 -.514652  .032835
   6 -1    -2.654976456  0.000  1.831 0.0000 -0.067771
   6  1    -1.343182910  0.000  2.181 0.0000  0.111254
   6 -2    -0.695826574  0.000  2.292 0.0000  0.170458
 core electrons: 80.0   fix =  1.00000000000000
     spin down:  40.0000000000000
     spin   up:  40.0000000000000
           mt:  80.0000000000000        leakage:  .000000000000000
     spin down:  .000000000000000                  40.0000000000000
     spin   up:  .000000000000000                  40.0000000000000
      >>>   no charge outside mt   <<<
```

The leakage should be less for all types. If not, try to increase the muffin-tin radius for the problematic type. Remember to increase the radius for all types of the same element. If the radius cannot be increased enough core states must be removed from the core and put in the pseudo core. The list above will tell which state to remove, i.e. the state with the highest energy. If ℓ>0, both jj-states must be removed.

Magnetic moment For spin polarized calculations the moment is printed. Search for `MAGNETIC MOMENTS`. You will find something like

```
  MAGNETIC MOMENTS

    Spin moments
      MT spin moments:
       type          moment
          1      -1.0866952
      Interstitial spin moment
             -0.0266597
      Total spin moment :   -1.1133548

    Orbital Moments
       type      l        orbital moment
          1      0           0.0000000
          1      1          -0.0067670
          1      2          -0.0093928
          1      3           0.0001165
          1      4          -0.0000148
          1      5           0.0000038
          1      6           0.0000004
          1      7           0.0000001
          1      8           0.0000000
          1  total          -0.0160539
```

```
Total orbital moment:   -0.0160539

Magnetic moment ...   -1.1294087 Bohr magnetons
```

The orbital moment section will only be printed if you also use spin-orbit coupling.

Note for the experienced. *Printed after the string* `Magnetic moment ...` *is now the* total *moment, so in a fully relativistic run it will be the sum of spin and orbital moments.*

Energy contributions At the end of the **out** file you find the summary of the different energy contributions. Search for `Energy Vector`.

```
Energy Vector
                  Eigenvalue sums
        Valence : -0.21494014926249E+02          -21.494 014 926 2
           Core : -0.71133165451495E+05      -71,133.165 451 495
                  Coulomb Energy
n(vc/2-vin)  mt : -0.39467169816317E+05      -39,467.169 816 317
n(vc/2-vin) int :  0.88136369704960E+01            8.813 636 970 4
       -zv(0)/2 : -0.33720233054332E+03         -337.202 330 543 3
          Total : -0.39795558509890E+05      -39,795.558 509 889
                  Exchange/Correlation
      n*exc int : -0.61715169291431E+01           -6.171 516 929 1
      n*exc  mt : -0.19777439007550E+04       -1,977.743 900 754 9
          Total : -0.19839154176841E+04       -1,983.915 417 684 1
                  SIC
core SIC contr. : -0.13134275280479E+00           -0.131 342 752 8
                  Total Energy
              E : -0.11293426473675E+06     -112,934.264 736 74
```

The second column is the same as the first but in another format. The core SIC contribution will only be printed if the correction is used, see Section 5.1.6.

Convergence summary Finally the summary of the convergence is printed. Search for `fsq`.

```
 Broydn:
    itr          fsq       length         ierr
      0  1.8317E-03        43404            1
```

Fsq is compound convergence parameter testing the convergence of several different entities, e.g. the potential and charge density. The level of convergence needed depends on what you are looking for. Simple EOS calculations can do with 10^{-10}-10^{-12} but a MAE or force calculation might need 10^{-14}-10^{-18}. A good test is to run to some convergence, check whatever property you are interested in, continue from your solution to a better convergence and see if the numbers are converging. Note, the quality of your calculation is not guaranteed by a good convergence. You can have a strongly converged but wrong solution. Always check the output for signs of problems as described above.

Things to grep for:

Total energy - `"E :"`

Magnetic moment - `Magnetic`

Fermi energy - `fermi`

Volume - `volume`

Timings - `TIME`

Things to search for:

Muffin-tin sizes - `2S/d`

Energy parameters - `energy`

Total charges - `charge summary`

ℓ-projected charges - `projected averages`

Energies summarized - `energy vector`

5.3 hist

Each time RSPt is invoked a summary of the output is written on the hist file. The `runs` and `runsl` programs modify the hist output from RSPt . Mainly cutting away the information from all but the last (and hopefully converged) run for each length in `length_scale`. They in turn produce the file `runsEnergyLog` with brief information about the iterations.

5.3.1 pot and eparm

After each iteration the mixed potential and energy parameters are stored in `out` and `eparm`. RSPt will automatically start from this input if these files are present. If they are not, the program will try to find `atomdens`, and restart from atomic densities.

5.3.2 jacob1 and jacob2

The `jacob1` file contains in/out energies/potentials. It is first read in as the old vectors and then the present vectors are printed out in every step. The first line is a formatting statement. The first number on the second line is the iteration in the Broyden scheme (not in the selfconsistent loop), the second number is how many vectors you are mixing. You mix the energies in one vector and then one potential vector for each mt harmonics and one for the interstitial stars. If you do a spin calculation you have the double amount of vectors, one set for energies/average potentials and one for energies/deviatoric potentials. The third line is the `fsq` value, note that also the energies are included in this measure of convergence.

Then follows all the vectors explicitly, the first one in each spin set is the energy vector and the last one is the interstitial potential vector. The ones in between are the mt potential vectors, one for each harmonics.

The first half of the numbers in each vector constitutes the input energy set/potential vector and the second half is the output - input energy set/potential vector.

The input potential is the potential that the new KS orbitals are calculated from. Those KS orbitals form a density and then a new potential, that then constitutes the output potential. When self-consistency is reached those should be the same, and the second half of numbers in each vector approach zero. Executing RSPt uses the input potential to solve the KS equations in order to obtain an output potential. The **jacob1** file thus only contains information from one step in the self-consistent loop.

Since using the out potential unmodified as new in potential in the next step in the self-consistent loop usually gives rise to bad convergence, the output potential is usually mixed in only to a certain degree and the main part of the new input potential is actually from the old input potential. In RSPt this is done according to the Broyden scheme. When several Broyden steps have been taken, the **jacob2** file is present among the output files. This file also contains information about previous steps in the Broyden loop.

To completely restart a calculation from atomic densities both jacob files should be removed, in addition to **pot** and **eparm**.

5.4 apts

`apts` stores the angular point grid for calculating the exchange-correlation energy and potential inside the muffin-tins.

6 Plotting output

In this section is described how to generate plot data for various quantities from *rspt*. It should be noted that *rspt*does not itself plot anything, but outputs plot data to various other programs, notably *xmgrace*, *gnuplot* and *IDL*.

6.1 Charge density plotting

There are several options for generating data for contour plots (or xyz plots) of the charge density in some plane that cuts the calculation cell. All of them start by setting the prnt flag 66 to `t` and to add the following to the end of the file `dta/form`:

```
#include "contour_plot_data"
```

Then type *make data*. This will insert the `contour_plot_data` file in the `data` file. The contour plot data section contains the following:

First format statements, a comment and an offset from the unit cell origin.

```
 (/ 3f18.0 // 3f18.0 / 3f18.0)
                  offset vector in lattice coordinates
  .000000000000000  .000000000000000  .000000000000000
```

Next comes two vectors defining a plane or, more precisely, a set of parallel planes (in lattice coordinates). The plane chosen will be the one that contains the point defined above by the offset vector.

```
          contour plane vectors in lattice coordinates
  1.00000000000000  .000000000000000  .000000000000000
  .000000000000000  1.00000000000000  .000000000000000
```

Then the number of points along the two vectors to define the plot mesh, and last an integer (1 or 0) that determines whether the core density should be added as well, or if only the valence density is to be plotted for the full density plot.

```
 (/ 2i18)
       intervals along plane vectors
               100               100
 (/ i12)
   add core?
            1
```

6.1.1 Plotting full charge, initial atomic charge and potential

The full charge and magnetization plots are generated whenever the flag 66 is set. The plot data is output to the files **cdpl.out** and **mdpl.out** for charge and magnetization, respectively. If also flag 46 is set, plots of the initial atomic densities will be generated. These are formatted for plotting using IDL (Interactive Data Language). A plot of the Kohn-Sham potential formatted for plotting with gnuplot is also output to files named **potential_contourN**, where the index N is 1 or 2, for the two spin channels. See Section 6.1.2 below for an explanation of *gnuplot* files.

6.1.2 Charge from some energy interval

To plot the charge only from states in some energy interval, just supply a file named **projdens.inp** in addition to the above steps for the full charge density. **projdens.inp** looks like this:

```
! Generate contour plot of the charge density according to
! Task = 0  : Full density
!        1  : Density from energy interval
!
! Output format = 0  : simple xyz format
!                 1  : IDL (Interactive Data Language) format
!                 2  : .dat and .plt file for gnuplot
!
! The energies are relative to the Fermi energy.
!
! Task    Output format   Emin     Emax      Energy unit
projdens
   1            2         -1.5     0.00           eV
```

The output files will be **projected_density** and **projected_magnetization**. For the projected density it is possible to choose output format. For the IDL format there is no file suffix, for xyz format the files end in **.xyz**, and for plotting with *gnuplot* you get one **.plt** and one **.dat** file. To plot using *gnuplot*, simply type

```
gnuplot projected_density.plt
```

and you will (hopefully) get a decent looking 3d plot. If something goes wrong or looks bad, you will need to look in the **.plt** file and edit things like axis ranges etc. by hand. An example of the output can be seen in Figure 1.

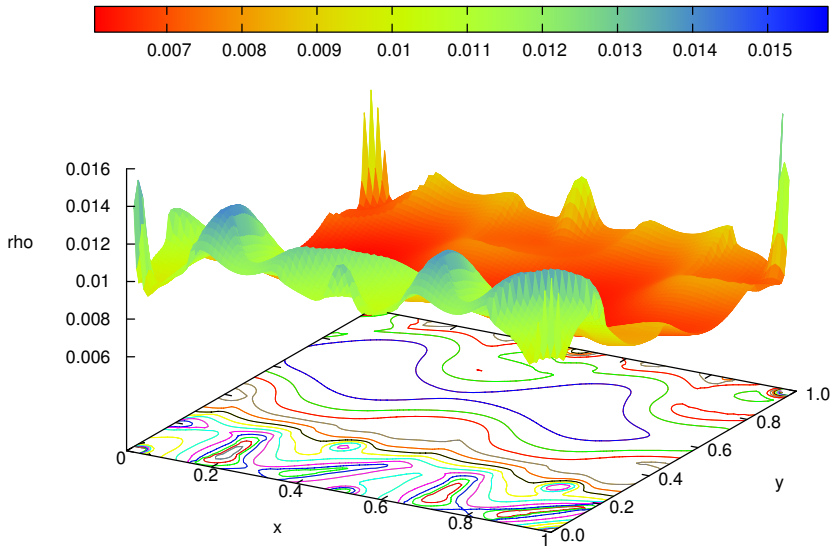

Figure 1: Plot of the charge density of Si from states with energies down to 1.5 eV below the Fermi level. The plot plane is defined by the vectors $(1, 1, -2)$ and $(0, 0, 2)$.

6.1.3 SIC state charge

If the `plot` flag in the **sic.inp** file is non-zero and `prnt(66)` is set, the charge density from the SIC states will be output as contour plots. Depending on whether the plot flag is set to 1, 2 or 3, the output format will be **.xyz**, *IDL* or *gnuplot*, as above.

6.2 Wave functions

If the flag prnt 60 is set to `t`, the code will output plot data for the core orbitals and the radial basis functions. The names of the files are for the core orbitals,

```
core_orbital.t.N.dat
```

where `T` labels the atomic type and `N` is the number of this function, as it appears in the **data** file. The file format is simple xy-data and the data output is r times the core orbital, $r\phi(r)$.

The radial basis functions are output to files with names,

```
rbf.t.l.e[.dn/.up].dat
```

Here `t` stands for type, `l` the ℓ quantum number and `e` is the energy set. The label `dn/up` is for spin down and up in a spin-polarized calculations. The format is $xy_{1\ldots6}$, where the different columns are r times:

1. the large component of the wave function
2. the small component of the wave function
3. the spin-orbit part of the large component.
4. energy derivative of the large component
5. energy derivative of the small component
6. energy derivative of the S-O part of the large component

If you want to understand details of what these numbers are you need to look up the scalar relativistic equation of Koelling and Harmon[13]. If you just want an idea of what the wavefunction looks like, just plot the first column (this is probably what you want). See Figure 2 for an example.

6.3 Charge and potential between neighbors

When the `prnt(37)` is set, the code will output plots of the charge density and potential between nearest neighbor pairs to the files **dplotX.dat** and **vplotX**. The 'X' is 1 or 2 for spin down and up respectively. The output is a file formatted for the 2D-plotting program XMGrace. This information can be used to set the muffin-tin radii to sensible values.

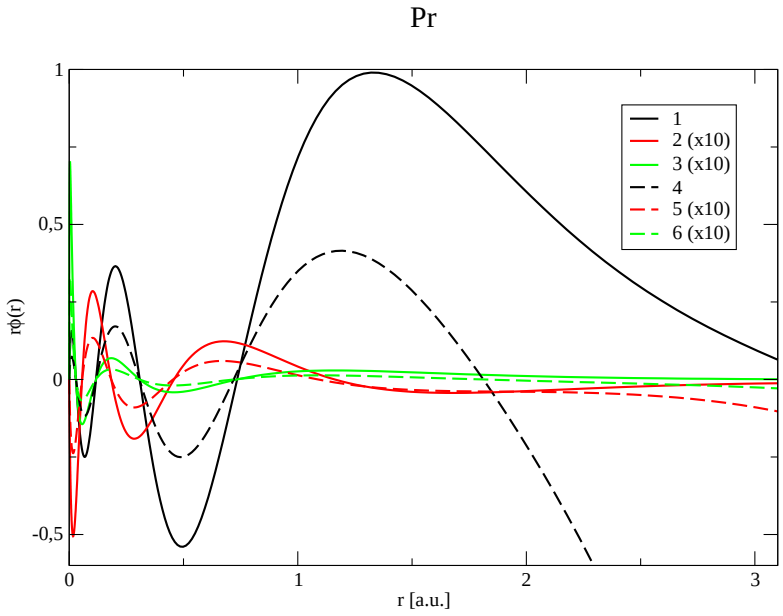

Figure 2: Example of basis function plot, in this case a $5s$ basis function in fcc Pr. The meaning of the numbers in the legends are given in the text. Note that some of them have been multiplied by 10 to be seen properly.

6.4 DOS and Partial DOS

RSPt can calculate both the total and partial density of states. If `prnt(65)` is set to `t` the program will calculate and print the density of states and the result is printed to the files **Dos.1.hea** and **Dos.1.0**. The first file contains a summary of the setup and the second the total density of states and the integrated number of states on a energy mesh from `emin` to `emax` with an increment of `de` (see below regarding the **dos.inp** file if you want to change these).

If also `prnt(64)` is set to T the ℓ-projected density of states will be calculated and printed. This can only be done if the tetrahedron method is used for the Brillouin zone integration. The result will be printed to the files **PDos.1.N.M**, where N is over types and M over spin, **PDos.1.int.M** and **PDos.1.hea**. The first set of files contains the ℓ-projected density of states for ℓ=0 to ℓ=min(3,lm) and a sum of the contributions from all higher ℓ, in states/Ry, the second set of files the interstitial contribution for each spin (in the interstitial all ℓ-character is lost) and finally the last file which contains a summary of the setup. See Figure 3 for an example. The total density of states for each spin-channel is available simply by summing up the projected constituents.

6.4.1 dos.inp

The behavior of the density of states (DOS) calculation can be controlled by the file **dos.inp**. The file contains keywords in the form keyword=value and the delimiter between keywords can be either space or comma. The possible keywords are given in Table 1.

Table 1: Explanations of the keywords in the **dos.inp** file.

Keyword	Type	Default	Effect
`quit`	integer	0	Quit after PDOS? 0 – return 1 – stop
`renormalize`	integer	0	0 – Site DOS = total 1 – Site + Int. DOS = total
`spinpro`	integer	`nspden`	Number of spin projections
`meshsize`	integer	calc. from `de`	specifies meshsize
`emin`	real	lowest eigenvalue	minimum energy
`emax`	real	highest eigenvalue	maximum energy
`de`	real	5 mRy	Energy increment
`type`	integer	`all`	type for which to apply `lmax`
`lmax`	integer	min(3,`lm`)	maximum ℓ projection

6.5 COOP, BCOOP

The Crystal Orbital Overlap Population (COOP) analysis is a tool for analyzing bonding. It can be loosely described as a density of states weighed by the (anti-) bonding character of the states at that energy: a positive number indicates bonding and a negative indicates antibonding states. Since the RSPt basis can

Si

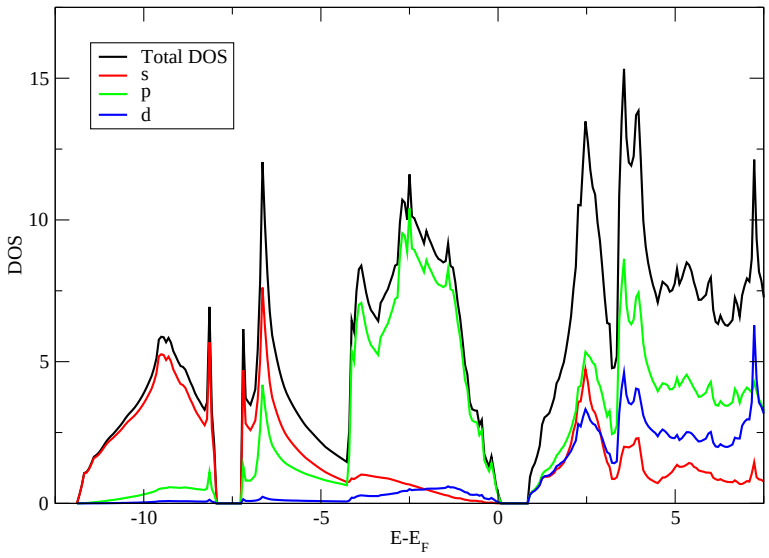

Figure 3: Total and partial densities of state for Si.

be very strongly overlapping, a modified version is implemented, Balanced COOP (BCOOP). The BCOOP is a function of the states in a bond, that is, a sum over a set of states from one atomic site to another. In RSPt, two types, l quantum numbers and spins are specified, which also allows for calculation of a partial BCOOP with respect to l and spin.

To perform a BCOOP calculation, simply put a **coop.inp** file in the run directory. The coop.inp file looks like this:

```
! coop.inp file for calculating the total COOP for Si
! in the diamond structure
-1  1.5  0.005          ! emin, emax, de
1                       ! number of lines
0   1  -1  0   1  -1  0  ! not_used, type1, l1, spin1, type2, l2, spin2
```

Comments can be written after an exclamation mark. The first line to be used specifies the energy mesh as lowest energy, highest energy and the energy increment. Next there follows a number of lines that specifies which contributions to the BCOOP that are to be summed. In this case the BCOOP is calculated between type 1 and 2. To sum over all l quantum numbers of a site, specify l=-1, and to sum over all spins, set spin=0, so in the example above the total BCOOP is calculated.

Presently, the code calculates not only the BCOOP but also a lot of other quantities such as the traditional COOP, COHP etc. The output files are put in the directory **coop_output/**. The files containing the BCOOP has names **nc.XX.out**, where XX is the number of the line specified in the **coop.inp**.

6.6 Fermi surface

To get a Fermi surface plot you need to:

1. Make a normal run to self-consistency with the settings you like.
2. Set `prnt(36)`. This makes the code output the Fermi surface in XCrySDen format.
3. Make sure that you have a **kmap** file, linked to **cub.N.g** in the **bz/** directory (this should have been done by the program *link_spts*).
4. Run *rspt*again.
5. Now the file **fermisurf.bxsf** has created in your run-directory. launch as `xcrysden -bxsf fermisurf.bxsf`

To view the plot you need XCrySDen installed, it is freely downloadable from www.xcrysden.org. The file is viewed simply by launching *xcrysden* with the command

```
xcrysden --bxsf fermisurf.bxsf
```

6.7 Band structure

To get a band structure plot you need to:

1. Make a normal run to self-consistency with the settings you like.
2. Set `prnt(2)`. This makes the code print the eigenvalues to file
3. Set `prnt(52)`. This makes the code stop after the eigenvalues have been calculated.
4. Generate a new **spts** file with k-points along the path you want to plot. This is done by the program *lineg1* or by the program *kpath*, as described below.

5. Save the old output file under another name.

6. Run *rspt*.

7. Now you have an **eigenvalues** file with the eigenvalues along the lines of your choice. Run the program *evconv* to generate file(s) with a plot friendly format of all the bands in the energy interval that you have specified in the **evconv.inp**. An example of **evconv.inp** file is:

```
# File for generating a band structure plot from an eigenvalue file.
# Note that the eigenvalues must have been generated along the path
# you want. The units of the reference energy is Ry. The unit of
# the min and max energies is determined by the scale factor.
#
#  reference energy (Ry)    scale factor   min energy   max energy
# (e.g. the fermi level)    (here Ry->eV)     (eV)         (eV)
   -2.2765409496228E-01       13.605692    -150.0000     50.0000
```

After having presented the whole procedure to generate the band plots, we now illustrate more in detail how to use the two programs to generate the **spts** file needed. The main difference between *lineg1* and *kpath* lays in the different treatment of the end points of each segment in the Brillouin zone. Another difference is that *kpath* can automatically give you a uniform k-point density along your path.

lineg1 requires to input separately the endpoints and the number of points to use for each segment in the Brillouin zone; *the endpoints appear once per segment in the output*; the program prints the k-points to screen, so you will have to cut and paste them into a new **spts** file (take care that the header should be moved to the top of the file!); a typical input file **lineg1.inp** is

```
# File for generating a set of lines through k-space for
# band structure plotting. A line is defined by first
# putting the number of points that you like, then the
# start and end vectors. The points are in units of the
# reciprocal lattice vectors. Considering an fcc lattice,
# the setting below would generate a path consisting
# of two lines, first one with 10 points from the gamma
# point to the X point, then one with
# 20 points from the X point to the L point.
#
# Note that in kpath each point is included one time
# per interval, i.e. in the following example the
# X point appears twice in the spts file, but the gamma
# point and the L point appear only once.
#
10
0.0 0.0 0.0
0.5 0.0 0.5
20
0.5 0.0 0.5
0.5 0.5 0.5
```

kpath requires you to specify the lattice vectors and the endpoints, and optionally the number of points for some of the segments. Unless an explicit number of k-points is given, the k-point density in

a segment is automatically set to match the first segment. Each endpoint appears only once per segment in the output. The program generates a file called **spts.band**, which you need to rename to **spts**; the typical input file **kpath.inp** is

```
# File for generating a set of lines through k-space for
# band structure plotting. First you we must write
# the bravais lattice vectors, i.e. the same vectors
# as in your symt.inp file.
# The following lines define the path of high-symmetry
# points to follow in the BZ according to
# k1 k2 k3 "Symmetry symbol" "[Number of k-points within the line]"
# Specifying the number of k-points is optional. If it is
# not specified it will be given the same k-point density
# as the first line (which is recommended).
#
# Note that in kpath each high symmetry point specified
# below is included only once in the spts file.
#
# Bravais lattice basis for fcc:
  0.00000000000000  0.50000000000000  0.50000000000000
  0.50000000000000  0.00000000000000  0.50000000000000
  0.50000000000000  0.50000000000000  0.00000000000000
#
# High symmetry k-points (in the reciprocal basis)
  0.50000000000000  0.00000000000000  0.50000000000000   X 60
  0.00000000000000  0.00000000000000  0.00000000000000   G
  0.50000000000000  0.50000000000000  0.50000000000000   L
  0.62500000000000  0.25000000000000  0.62500000000000   U
  0.50000000000000  0.00000000000000  0.50000000000000   X
  0.50000000000000  0.25000000000000  0.75000000000000   W
  0.50000000000000  0.50000000000000  0.50000000000000   L
  0.37500000000000  0.37500000000000  0.75000000000000   K
  0.50000000000000  0.25000000000000  0.75000000000000   W
```

6.7.1 fatbands.inp

If the file **fatband.inp** is present the code will perform a calculation of the band character. Currently 3 modes are implemented: 0 (default) Performs projection on type, energy set, l and m_l; 1 performs projection on type; 2 performs projection on type, energy set and l. The only input required is the mode number.

Example of **fatband.inp** file:

```
2
```

The output files are named according to the projection e.g. **fatbands.t01.e1.l2.m-1** for the projection on type 1, energy set 1, l-quantum number 2 and m_l-quantum number -1. Beware that the most verbose mode 0 creates vast amounts of files for large systems. The character is found in the third column, the magnitude is arbitrary and scaled to fit the plotted energy range. In XMGrace the bands are simply plotted as

```
xmgrace -settype xydy fatbands.tXX.eX.lX.mX
```

7 Orbital polarization

To perform a calculation with orbital polarization corrections, you need to set the `prnt` flag 42 to `t`. The orbital polarization is currently implemented so that the user is required to give a **orbpol.inp** file that contains information about which orbitals to correct. Suppose that we want to run PrO_2, which has two types of atoms, Pr that has f-electrons that we wish to correct and O that we should not correct. Then an **orbpol.inp** looks like this:

```
# Example orbpol.inp file for PrO2
# Types to correct
1  0
# l to correct
3
# energy set to correct
1
```

The # symbol comments out the rest of the line. The first 1 and 0 indicates that we wish to correct the first type but not the second. Then follows the ℓ and energy set to be corrected. Note that these are only given once per *corrected* type.

8 Optics

Quick information about the optics implemented in RSPt. The **optic.inp** is used when calculating dielectric tensors (set flag51 to true in data). If **optic.inp** does not exist and flag51 is true, then the values used in the simulation will be set to default. Typically **optic.inp** looks like:

```
ecut (Ry)  shift (Ry)  meshmax (Ry)  meshinc (Ry)
 80.       0.000          6.0         0.0010
interband isolator jdos!interband semicon jdos!
   t        f       f
eps (cutoff for matrix element size)
3.552714e-13
```

Here the variables have the following meaning:

interband if true, it determines if interband insulator is used when performing the simulation

semicon if true calculates new fermilevel in dielec2; otherwise a band cutting is performed

jdos does nothing [Igor: are we sure about this???]

ecut determines at what energy the band's are cut off

meshmax determines the maximum energy used for the number of meshpoints

meshinc used when determining the number of meshpoints; nmesh=meshmax/meshinc +2

shift used to determine x,y for the mesh size [Igor: are we sure this is not a rigid energy shift of the spectrum?]

By having deep-laying orbitals in the valence it is possible to calculate XAS and XMCD from these optics routines. Johan Schott has calculated $L_{2,3}$-edges with DFT for NiO and the agreement with the results using the XAS routines (described just below) is within the experimental broadening.

9 XAS and XMCD

How to calculate spectra simulating x-ray absorption spectroscopy (XAS) using DFT together with the dipole approximation, as implemented in RSPt, is described here. Three light polarizations are calculated, providing information for construction of x-ray magnetic circular dichroism (XMCD). By proving the **xmcd.inp**, x-ray absorption (XA) spectra for $j = l - \frac{1}{2}$ and $j = l + \frac{1}{2}$ core states are calculated and saved to the files **mdich.lmin** and **mdich.lplu**, respectively. The input format for **xmcd.inp** looks like:

```
de emax nquan lcore cortype
```

where the variables have the following meaning:

de step size on the generated energy mesh. Rydberg unit.

emax maximum energy, relative to the Fermi level, in the generated energy mesh. Rydberg unit.

nquan principal quantum number for the core-states to excite.

lcore angular momentum number for the core-states to excite.

cortype RSPt type for the core-states to excite.

For example, to calculate the $L_{2,3}$-edges for RSPt type 1 on a mesh with 1 mRy in energy spacing and up to 1.5 Ry above the Fermi level, the **xmcd.inp** looks like:

```
0.001 1.5 2 1 1
```

Note that one can use the Jupyter notebook **rspt2quanty.ipynb** to construct a discrete single particle Hamiltonian and print it in a Quanty-format. This can be useful when studying XAS, XMCD and RIXS. The discretization of the hybridization function requires some care, especially if only a few bath states are used.

10 SIC

Here is a description of how to use the self-interaction correction scheme as implemented in *rspt*. There is a very brief discussion stating a few key quantities that you will see during the run, followed by a description of the input file and the output in the **out** file.

10.1 Summary SIC as implemented in *rspt*

The SIC implemented in *rspt*is according to the Perdew-Zunger prescription[14]. The idea is to impose the correct behaviour of a one electron system by demanding that the interaction energy of any single electron state with itself be zero:

$$E^H[n_\alpha(\mathbf{r})] + E^{XC}[n_\alpha(\mathbf{r})] = 0. \tag{40}$$

Here E^H is the Hartree energy, E^{XC} is the exchange-correlation energy and alpha indexes any one electron state. To correct the LDA, we simply subtract the left hand side of the above expression, evaluated for some suitable set of single particle orbitals. This relation fulfilled by Hartree-Fock, and so SIC is one in the row of methods that strive to correct LDA by reintroducing some desired property of Hartree-Fock or another (LDA+U, orbital polarization, exact exchange ...). The above energy correction gives rise to a similar correction of the potential, since we must also have

$$V^H[n_\alpha(\mathbf{r})] + V^{XC}[n_\alpha(\mathbf{r})] = V_\alpha^{SI} = 0. \tag{41}$$

The potential correction is calculated assuming that the corrected state has the shape of a core state, and the correction to the Hamiltonian for the corrected ℓ-shell is given by

$$H = H^{LDA} - \sum_\alpha \Delta\epsilon_\alpha^{SIC} |\alpha\rangle\langle\alpha|, \tag{42}$$

where the state $|\alpha\rangle$ is now an explicit linear combination of m_ℓ and spin quantum numbers and the eigenvalue shift is given by the integral

$$\Delta\epsilon_\alpha^{SIC} = \int_0^{S_{core}} dr\, V_\alpha^{SI}(r) n_\alpha^{SIC}(r). \tag{43}$$

The functional in Equation (40) can then be evaluated by projecting out the density of the α states from the total density. It turns out that to minimize the energy of the SIC functional it is not suitable to use the LDA basis functions. Therefore the scheme is modified so that instead of determining the linearization energy in normal way, we choose an energy such that the basis function gets the same boundary condition as the core function used to determine the potential correction in Equation (41). There is also the option to use LMTO basis functions determined inside the muffin-tins (Equation (15)) from the corrected potential, $V^{LDA}(r) - V_\alpha^{SI}(r)$, and at the energy determined for the solution of the core problem, $E_\alpha^{\nu,SIC}$, for the α shell, but this setting is currently not recommended. It has so far not been found energetically favourable and it can be numerically unstable.

10.2 The SIC input

SIC is activated by setting `prnt(39)` to true. The input related to SIC is then read from the file **sic.inp**. A sample **sic.inp** file can be found in the **inputs/** directory. We here give another one, for the case of fcc Ce.

10.2.1 The sic.inp file

```
# sic.inp for fcc Ce.
#
# Type of eigenvalue shifts [1]
1
# Correct basis functions? [0]
0
# density projection method [8]
8
# energy window for projection, used if
# projection method is an odd number [-0.1, 0.1]
-.1   0.1
# plot flag
2
# verbose output? [0]
0
# Types that has SIC states
1
# l values to correct
3
# Energy set
1
# Tail [0]
0
# Number of corrected states
1
# Gamma 1 (A_2u)
# spin down
0.000000000000000 0.000000000000000
0.000000000000000 0.707106781186547
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 -.707106781186547
0.000000000000000 0.000000000000000
# spin up
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
0.000000000000000 0.000000000000000
```

Shift type - Which type of SIC to do. Presently there are two options, `1` and `2`. Shift type 1 calculates the SIC eigenvalue shifts from a core function and type 2 uses the spherical average of the valence density. One can also put `0`, in which case the eigenvalue shifts are not calculated but read from the file. Note that this is not a proper SIC

calculation and only exists for purely experimental purposes.

Basis function correction - `0` or `1`. Whether to calculate the basis functions using the corrected potential and at the energy of the SIC eigenvalue. Recommended setting is `0`.

Density projection method - `8` or `9`. How to project out the density from the SIC states. `8` just projects out the m_ℓ,spin linear combination given and `9` uses only states in the energy interval $[E_\alpha^{\nu,SIC} + E_{min}, E_\alpha^{\nu,SIC} + E_{max}]$. There has been more options and they may appear again, hence the numbers 8 and 9.

E_{min}**,** E_{max} - Defining the energy interval used when restricting the density calculation to states in some interval around $E_\alpha^{\nu,SIC}$. Note the sign convention. The E_{min} in the example file above gives states from the interval $[E_\alpha^{\nu,SIC} - 0.1, E_\alpha^{\nu,SIC} + 0.1]$. This interval is also set as default if E_{min} and E_{max} are both set to zero.

Plot flag - `0-3`. This flag determines if and how to produce plots of SIC quantities. If `0` is specified, no plots are produced. If some other number is set, the radial density and the SI potential from the core calculation is output. If `prnt(66)` is set, the code will output contour plots of the individual SIC state densities in an output format determined by the value of the plot flag. `1` is IDL format, `2` gnuplot format and `3` is simple xyz-format.

Verbose - `0` or `1`. You may tell the code to output more information to the **out** file and some additional plots are generated by the core routines. This is probably not useful for anything but debugging.

SIC types - List of #types integers. Indicate which types that has (`1`) or hasn't (`0`) corrected states.

SIC ℓ quantum numbers - List of #(types with SIC) integers. Which ℓ quantum numbers to correct on the respective sites given as a list of integers.

Energy set - List of #(types with SIC) integers. To which energy set do we apply the correction.

Tail - List of #(types with SIC) integers. For which tails do we apply SIC? `0` is the recommended option, and this applies the correction to all tails. If some other positive integer is given, only the basis functions with that tail are corrected. If a negative integer is given all basis functions *except* those with that tail are corrected.

Number of corrected states - List of #(types with SIC) integers. How many states are there to correct for this type?

Gamma's - The corrected states, explicitly defined as linear combinations of spherical harmonics for the two spin channels. In this particular example, an A_{2u} crystal field state has been selected to localize a single f-electron in Ce without breaking the cubic symmetry.

What is given here are the coefficients for the linear combination, so the angular part of this state is

$$\frac{i}{\sqrt{2}}(Y_{3,-2}(\hat{\mathbf{r}}) - Y_{3,2}(\hat{\mathbf{r}})). \tag{44}$$

There must be one such list of complex numbers for each state to be corrected.

10.2.2 Comments & considerations

- The muffin-tin radii should in general be chosen fairly large for the atoms which have corrected states.
- The default behavior for the basis function is to take an energy such that it ends up looking like a SIC corrected core function, but setting the basis function flag to some non-zero value still works as before. See documentation in Section 5.1.6 about the **element_N.X** file.
- Use only negative tails for SIC states. This gives a smoother convergence and is consistent with the idea that the state is localized. Sometimes the calculation will not converge at all if you use a positive tail.
- *Remember:* SIC is variational with respect to the choice of corrected orbitals. If some specification of the linear combinations of the α states give lower energy than some other, then that is (within the SIC formalism) the correct answer. Whether the correct SIC answer is right is a different matter.

10.3 SIC output

If we simply search the **out** file for the word `SIC` we will first find[†]:

```
 SIC parameters
 Shift type        :  1
 Projection method :  8
 Basis correction  :  F
 Verbose output    :  F
 SIC Plots         :  0
 type   l  eset  desic down  desic up  Enu down    Enu up
    1   3    1    -.6830087 -.6803427 -.0145307  .0292303
```

This restates some of the input, and then shows the result for various quantities calculated. Here we see the SIC shifts the $4f$ eigenvalue down by about 0.7 Ry, and that this puts the state at an energy of around -0.015 Ry.

If we search further down in the **out** file we find:

```
Densities, potentials and energy contributions from SIC states

 Charge density from state   1
  Parent muffin-tin :
     spin
     down       0.956003881
       up       0.000000000
```

[†]The examples here come from an **out** file associated with the above fcc Ce input file

This shows information about the projected density used for evaluating the energy correction in Equation (40). Note that the charge does not sum up to exactly one. This is normal behaviour, and the projected density will never be exactly one for each state. If the projected density starts to deviate much from one ($> 10\%$), it is an indication that something has gone wrong.

Search again and we come to the last bit of SIC output:

```
                  SIC
 kin energy corr :  0.68300878542373E+00            0.683 008 785 4
      Hartree mt :  0.69124567494590E+00            0.691 245 674 9
         E_xc mt : -0.65975607825568E+00           -0.659 756 078 2
        E_sic mt :  0.31489596690228E-01            0.031 489 596 6
                  Total Energy
               E : -0.17717545245968E+05      -17,717.545 245 967
            Other total energies
  LDA/GGA energy : -0.17717513756371E+05      -17,717.513 756 370
   Full-CD SIC E : -0.17717405419478E+05      -17,717.405 419 477
```

This is an extra section of the different energy contributions. Note that we only use the spherical component of the projected charge density when we apply the correction.

11 DMFT

The DMFT machinery of RSPt is controlled by the presence of the input file **green.inp**, and gives access to many different functionalities. The usage of the Green's functions makes the input slightly cumbersome, but also very flexible. In fact one of the nicest feature of the DMFT machinery is that many different electronic local quantities are accessible in a simple and natural way by means of only the **green.inp** file.

In the following section we will first present the files that are needed to make a DMFT simulation. Then we will give an overview of all the possible fields we can set up in the input file. Finally examples of how to run a few simple tasks are given. Please notice that the reader is supposed to be more or less familiar with the dynamical mean-field theory or at least with the Green's function formalism at finite temperature, e.g. with concepts such as the Matsubara frequencies (for a review of the basic formalism and equations used in RSPt one can look at the References [15] and [16]).

11.1 Initial RSPt setup

The starting point of a DMFT simulation is a reasonably converged RSPt calculation. In principle a computational theory should not depend on the starting point, but when using DMFT or LDA+U within the LMTO method the initial electron density defines the correlated basis functions, and therefore the Hamiltonian. Orientatively the user is safe when starting from a LDA (or GGA) simulation whose `fsq` value is equal or less than 10^{-8} for a spectrum calculation, note that if a charge self-consistent calculation is done this is of less importance, and the final `fsq` value is the indication of convergence.

The first thing to keep in mind is that the Green's functions used in the DMFT part are defined within the finite temperature Matsubara formalism. In order to assure a precise correspondence between the LDA and DMFT part of

RSPt *it is mandatory to use the thermal smearing* as smearing for the Brillouin zone integration.

Moreover, a consideration about the basis used in RSPt is needed. A DMFT simulation is much heavier of a standard LDA simulation, and therefore it is better to keep the basis set as small as possible. It is advised to avoid to use semi-core states (multiple energy sets) unless they are really necessary for obtaining the correct physics of the crystal. Moreover the user should keep in mind that the number of allowed tail energies of the states from which we are supposed to take the correlated orbitals is dictated by the parameter `basis_type` (see below).

11.2 Input file example

```
matsubara
1500 40 120 10                  ! nmats head body tail

inputoutput
.true. .true.                   ! readsig csc

convergency
1d-6 1d-4 999 999               ! ndelta sigma_acc maxiter maxsolveriter

cluster
1 eV UJ                         ! ntot [eV] [UJ] [Cf]
1 2 1 1 0 3.5 0.9               ! t l e site basis F0 [F2 [F4 [F6]]] or U [J]
1 1 0.4                         ! solver DC sigma_mix [tensmom_mag]
                                ! solver parameters
                                ! dc parameters


cluster
1 eV UJ                         ! ntot [eV] [UJ] [Cf]
2 2 1 1 0 3.5 0.9               ! t l e site basis F0 [F2 [F4 [F6]]] or U [J]
1 1 0.4                         ! solver DC sigma_mix [tensmom_mag]
                                ! solver parameters
                                ! dc parameters
```

11.3 Main keywords

As one can see in the previous section the input file is based on main keywords, followed by a block of free formatted data parameters. Every block can appear only once except the cluster block, which can appear several times. On the right hand side the parameters are named (after the comment symbol), and those names will be used for their description. If the name is within circular brackets, then the parameter is optional.

Here follows an overview of all the possible keywords, while a more detailed description is made in the next sections.

matsubara	sets the Matsubara mesh
energymesh	sets the details of the real energy mesh
convergency	sets the accuracy of the calculation
cluster	sets the correlated problem for a given site (or sites)
unitcell	sets a cluster for the whole unit cell
projection	sets the shape of the correlated orbitals

mixing	sets mixing parameters for self-energy and chemical potential
inputoutput	sets how to start or resume a calculation (disk access)
tensmom	activates the tensor moment decomposition
spectrum	prints out spectral properties, e.g. DOS and PDOS
carriers	prints out carriers concentrations
modelexchange	activates an external local exchange field
isoexch	activates the calculation of the J_{ij}'s
verbose	sets the level of verbosity of the output
debug	sets several debug options (mainly addressed to developers).

11.4 cluster block

The cluster block is the only required block in the **green.inp** file, and contains the whole description of the local problem that one wants to solve through DMFT. The cluster block is divided into several rows, and a typical example looks like:

```
cluster
1 eV UJ                          ! ntot [eV] [UJ] [Cf] [Id]
1 2 1 1 0 3.5 0.9                ! t l e site basis F0 [F2 [F4 [F6]]] or U [J]
1 1 0.4                          ! solver DC sigma_mix [tensmom_mag]
                                 ! solver parameters
                                 ! dc parametrescluster
```

or in a more complicated case

```
cluster
1 eV UJ Cf IdNi3d                ! ntot [eV] [UJ] [Cf] [Id]
1 2 1 1 0 6.5                    ! t l e site basis F0 [F2 [F4 [F6]]] or U [J]
6 -1 0.3                         ! solver DC sigma_mix [tensmom_mag]
6 0 -3 2.0 6 30.0                ! solver parameters
0.2                              ! dc parameters
```

Here a description of the fields:

ntot	total number of atoms contained inside the cluster; for standard single-site DMFT this number should be set to one.
eV	the U-matrix parameters given in the following line are assumed to be in eV, instead of Ry
UJ	the U-matrix parameters given in the following line are expected to be U and J, instead of the Slater integrals
Cf	Use an automatically generated optimal crystal field basis for your cluster. This works also in the presence of spin-orbit coupling.
Id	Give the cluster a custom label.
t	atomic type, referred to the **data** file, for which we desire to construct the set of local correlated orbitals

l — ℓ-number of the set of correlated orbitals

e — energy set of the set of correlated orbitals

site — site for a given type, referred to the **data** file, for which we want to construct the set of correlated orbitals; it is important only for multi-site clusters. The expert user may specify a certain tail by using negative numbers according to the syntax `site = site - tail*(total number of sites)`. The tail specification is by construction not used in the MT projection scheme.

basis — once the atomic-like basis set has been constructed (see below, in the projection block) the basis functions can be recombined in order to focus on different crystal properties; in the following we present all implemented possibilities by using σ for the spins, and s-p-d-f for the angular character. Note that in the presence of the "Cf" flag above, these orbitals will only serve as the starting point for the automatically generated crystal field basis.

- **0** — standard atomic-like basis of complex spherical harmonics, in the following order:
  $\sigma_1[s_0], \sigma_2[s_0]$
  $\sigma_1[p_{-1}, p_0, p_1], \sigma_2[p_{-1}, p_0, p_1]$
  $\sigma_1[d_{-2}, d_{-1}, d_0, d_1, d_2], \sigma_2[d_{-2}, d_{-1}, d_0, d_1, d_2]$
  $\sigma_1[f_{-3}, f_{-2}, f_{-1}, f_0, f_1, f_2, f_3], \sigma_2[f_{-3}, f_{-2}, f_{-1}, f_0, f_1, f_2, f_3]$
- **1** — t_{2g} or t_{2u} crystal field basis[‡], in the following order:
  $\sigma_1[p_y, p_x, p_z], \sigma_2[p_y, p_x, p_z]$
  $\sigma_1[d_{yz}, d_{xz}, d_{xy}], \sigma_2[d_{yz}, d_{xz}, d_{xy}]$
  $\sigma_1[f_{x(y^2-z^2)}, f_{y(z^2-x^2)}, f_{z(x^2-y^2)}], \sigma_2[f_{x(y^2-z^2)}, f_{y(z^2-x^2)}, f_{z(x^2-y^2)}]$
- **2** — e_g or t_{1u} crystal field basis, in the following order:
  $\sigma_1[d_{z^2}, d_{(x^2-y^2)}], \sigma_2[d_{z^2}, d_{(x^2-y^2)}]$
  $\sigma_1[f_{x^3}, f_{y^3}, f_{z^3}], \sigma_2[f_{x^3}, f_{y^3}, f_{z^3}]$
- **3** — options 2 and 1 together (full crystal field basis for the d-shell), in the following order:
  $\sigma_1[\texttt{orbitals\_2}, \texttt{orbitals\_1}], \sigma_2[\texttt{orbitals\_2}, \texttt{orbitals\_1}]$
  to give an example, the d-shell would be:
  $\sigma_1[d_{z^2}, d_{(x^2-y^2)}, d_{yz}, d_{xz}, d_{xy}], \sigma_2[d_{z^2}, d_{(x^2-y^2)}, d_{yz}, d_{xz}, d_{xy}]$
- **4** — a2u crystal field basis; order: $\sigma_1[f_{xyz}], \sigma_2[f_{xyz}]$

[‡]We must note that two conventions exist regarding the construction of the real harmonics of the f-shell: the cubic set (adopted here) and the general set. One can look for more information at the orbitron webpage, or elsewhere on the web.

5 options 4 and 1 together, in the following order:
σ_1[orbitals_4, orbitals_1], σ_2[orbitals_4, orbitals_1]

6 options 4 and 2 together, in the following order:
σ_1[orbitals_4, orbitals_2], σ_2[orbitals_4, orbitals_2]

7 options 4 and 2 and 1 together (full crystal field basis for the f-shell), in the following order:
σ_1[orbitals_4, orbitals_2, orbitals_1],
σ_2[orbitals_4, orbitals_2, orbitals_1]

8 JJ basis, in the following order:
s $2[J = \frac{1}{2}]$
p $2[J = \frac{1}{2}], 4[J = \frac{3}{2}]$
d $4[J = \frac{3}{2}], 6[J = \frac{5}{2}]$
f $6[J = \frac{5}{2}], 8[J = \frac{7}{2}]$

F0, ... here one should list the U-matrix parameters as specified in the previous input line; as default the code expects the Slater parameters F^0 F^2 and F^4 for d-electrons and F^0 F^2 F^4 and F^6 for f-electrons in Ry. If some values are not given, they will be calculated via the radial Slater integrals. The values of F^2 and F^4 are empirically reduced by 18% (8%) and 12% (3%) in d-orbitals (f-orbitals) due to screening effects, respectively, unless the debug keyword "Noscreening" is given. If the option UJ is used, the code will expect instead values for U and J, which will be converted to Slater parameters using fixed atomic ratios. If J is not given, all parameters except F^0 will be again calculated as described above. If the option eV is specified, all values are expected to be in eV but will be immediately converted to Ry in the formatted output.

solver method used for the solution of the Anderson impurity model

0 no solver

1 SPTF solver

2 LDA+U or Hartree-Fock solver

3 CT-QMC solver in density-density approximation (for developers only)

4 CT-QMC solver in four index U-matrix (not implemented yet)

5 Hubbard I solver

6 exact diagonalization solver

	-1	The CPA solver. It produces a self-energy that reproduces the average Green's function of two or more clusters. (Still in testing)
	-2	so-called carbon-copy solver. It copies the self-energy from one cluster to the other. No checks are performed, so this solver is only for expert users.

DC double counting option.

-1	solver-dependent double counting (see solver description)
0	no double-counting
1	interpolation double-counting (PRB **67**, 153106)
2	fully localized limit (FLL)
3	around mean-field (AMF)
4	interpolation double-counting with fixed interpolation constant (first dc_parameter; x=0 corresponds to FLL, x=1 corresponds to AMF), and optionally a fixed occupation number (second dc_parameter)
10	double-counting correction, which fixes the impurity occupation, such that: $\mathrm{Tr}_{\omega,m}(G_{imp}) = \mathrm{Tr}_{\omega,m}(G_0)$ (PRB **77**, 205112)
11...14	same as 1...4, but the density matrix is now constructed from G_0 instead of G.

sigma_mix mixing parameter for the convergence of the self-energy

tensmom_mag a weight applied to each term in the tensor moment decomposition. A zero valued weight implies that no tensor moments decomposition will be applied to this cluster.

solver_parameters the solver parameters depend on the chosen solver, and therefore is going to be described below, together with the solvers

dc_parameters extra double counting parameters are needed for the double counting options -1 and 4. The number (and meaning) of those parameters is solver-dependent for double counting option -1, as described below for each solver. For option 4, instead, one can specify two parameters; the first one corresponds to the interpolation constant x (see PRB **67**, 153106), while the second one corresponds to a fixed occupation.

The cluster blocks in **green.inp** are given per (cluster) type. The self-energy is automatically copied to the other (cluster) sites.

11.5 Example 1: running a simple LDA+U calculation

Using the DMFT interface for running LDA+U simulations is a particularly instructing exercise. A good test case is the prototypical anti-ferromagnetic charge-transfer insulator NiO, whose converged RSPt files are contained in the testsuite . In the same folder is a suitable **green.inp** file for an LDA+U simulation:

```
cluster
1 Ry                        ! ntot [eV] [UJ] [Cf]
1 2 1 1 0 0.59 0.60 0.38    ! t l e site basis F0 [F2 [F4 [F6]]] or U [J]
2 2 1.0                     ! solver DC sigma_mix [tensmom_mag]

cluster
1 Ry                        ! ntot [eV] [UJ] [Cf]
2 2 1 1 0 0.59 0.60 0.38    ! t l e site basis F0 [F2 [F4 [F6]]] or U [J]
2 2 1.0                     ! solver DC sigma_mix [tensmom_mag]
```

From this example it is clear that the only needed blocks are the two cluster blocks for the two atomic types of the anti-ferromagnetic unit cell. All the other fields in the other blocks are set up using defaults values. The next step is to run *rspt*and let the calculator complete the the simulation.

Now it is the moment of having a look at the **out** file in order to have an idea of the basic information print by the DMFT code (more detailed information is accessible through the verbose flags). The beginning of the DMFT part itself is clearly marked in the **out** file:

```
 +---------------------------- BRIANNA --------------------------------+
```

All the different printouts associated to the various parts of the DMFT cycle are marked with the appropriate headers, e.g.

```
 ************************************************
     Formated and processed input data
 ************************************************
 matsubara
      1     1     0     0                        ! nmats nmats_head nmats_log nmats_tail

 energymesh
   1001  -1.00   1.00 0.010                      ! nemesh emin emax eim

 inputoutput
   T  T                                          ! readsig csc

 convergency
  1.00E-06 1.00E-04 999 999                      ! ndelta sigma_acc maxiter maxsolveriter

 projection
   1                                            ! basis_type

 mixing
   3  0.15000  0.00000                           ! mix_method max_slope sigdiff_factor

 cluster
   1                                            ! ntot [eV] [UJ] [Cf]
   1  2  1  1  0  0.5900  0.6000  0.3800         ! ID F0 [F2 [F4 [F6]]] or U [J]
   2  2  1.0000  0.0000                          ! solver_type DC sigma_mix [tensmom_mag]

 cluster
   1                                            ! ntot [eV] [UJ] [Cf]
   2  2  1  1  0  0.5900  0.6000  0.3800         ! ID F0 [F2 [F4 [F6]]] or U [J]
   2  2  1.0000  0.0000                          ! solver_type DC sigma_mix [tensmom_mag]
```

In particular here it is visualized the formatted input file, containing all the parameters which are going to be used in the DMFT code. In the present

LDA+U example only the cluster blocks have been specified in the **green.inp** file, therefore several other blocks have been set up to default values for the problem. The meaning of all these blocks is going to be described in the following sections. Now we focus on the initialization of the cluster object, which is printed a few lines later in the **out** file:

```
*********************************************
     Initialization of the clusters
*********************************************

*********************************************
      ID 0102010100     Initial local properties
*********************************************


      ID 0102010100 Local overlap matrix before normalization (close to 1)
Real part:
 0.98  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.98  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.98  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.98  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.98  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.98  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.98  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.98  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.98  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.98
Imag part:
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
```

The first thing to notice is the ID tag, which specifies completely which correlated atom the code is referring to: 0102010100 stays for type 01, ℓ=02, energy set 01, site 01 for the type 01, and finally plain atomic-like basis in complex spherical harmonics (using the same convention for the field basis in the cluster block. In principle the same electronic shell of a given atom can be approached with two different basis to obtain additional information, e.g. projection on more physically meaningful orbitals. What follows is a complex matrix[§] characterizing the orthonormalization of the correlated orbital basis. Ideally one should obtain the real identity matrix, but due to our construction of the orbitals (see below the projection block) not always this condition can be fulfilled. However, for NiO the user can be quite happy, since the values are very close to the identity. In general one should try to avoid calculations where the diagonal elements are less than 0.90 and the off-diagonal elements are bigger than 0.05.

Then the LDA+U cycle starts. At the end of the first iteration and brief summary is printed, specifying the current chemical potential, corresponding occupation and also how converged in the self-energy (static potential in this case).

```
############### Iteration nr.   1 ##############
```

[§]For sake of visualization the number of digits printed for this and the following matrices has been reduced with respect to the real **out** file.

```
cycle: The chemical potential is taken from the LDA calculation.
Griffin: The local dmtx is projected from the global dmtx
***************** Green's function summary ****************
It :  1,         mu =           0.628438174482
Total occupation    =          31.999999999986
Sigdiff             =           0.000000000000
************************************************************
```

Then the local dmtx is printed:

```
*********************************************
     Local properties
*********************************************

      ID 0102010100     Local dmtx: rho
Real:
  0.904  0.000  0.000 -0.000  0.026  0.000  0.000  0.000  0.000  0.000
  0.000  0.895  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
  0.000  0.000  0.912 -0.000  0.000  0.000  0.000  0.000  0.000  0.000
 -0.000  0.000 -0.000  0.895 -0.000  0.000  0.000  0.000  0.000  0.000
  0.026  0.000  0.000 -0.000  0.904  0.000  0.000  0.000  0.000  0.000
  0.000  0.000  0.000  0.000  0.000  0.709 -0.000  0.000  0.000  0.215
  0.000  0.000  0.000  0.000  0.000 -0.000  0.637  0.000  0.001 -0.000
  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.780  0.000  0.000
  0.000  0.000  0.000  0.000  0.000  0.000  0.001  0.000  0.637  0.000
  0.000  0.000  0.000  0.000  0.000  0.215 -0.000  0.000  0.000  0.709
Imag:
  0.000 -0.000 -0.021 -0.000  0.000  0.000  0.000  0.000  0.000  0.000
  0.000  0.000 -0.000  0.035  0.000  0.000  0.000  0.000  0.000  0.000
  0.021  0.000 -0.000  0.000 -0.021  0.000  0.000  0.000  0.000  0.000
  0.000 -0.035 -0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
 -0.000 -0.000  0.021 -0.000  0.000  0.000  0.000  0.000  0.000  0.000
  0.000  0.000  0.000  0.000  0.000 -0.000  0.000 -0.176  0.000  0.000
  0.000  0.000  0.000  0.000  0.000 -0.000  0.000  0.000  0.287 -0.000
  0.000  0.000  0.000  0.000  0.000  0.176 -0.000 -0.000 -0.000 -0.176
  0.000  0.000  0.000  0.000  0.000 -0.000 -0.287  0.000 -0.000 -0.000
  0.000  0.000  0.000  0.000  0.000 -0.000  0.000  0.176  0.000 -0.000
```

This matrix generates the LDA+U potential, which is not printed using the default output settings. In the standard LDA+U implementations once the effective potential has been calculated, the basic electronic structure machinery is used for solving the modified density functional problem. However the present LDA+U implementation is a small part of the DMFT project, and therefore the DMFT machinery is used instead, which requires a nested cycle to converge the LDA+U potential. Once full convergence has been reached, an additional iteration is needed for dumping to file the density matrix, which will be used by RSPt for creating a new **pot** file. This last iteration is summarized as follows

```
***************** Green's function summary ****************
It : 11,         mu =           0.628438174482
Total occupation    =          32.000000000000
Sigdiff             =           0.000089653214
Spin moment (mu_B)  =           0.000000000000
Total energy        =           9.007880605973
RSPt eigen energy   =           8.649149479342
Energy correction   =           0.358731126632
DMFT eigen energy   =           8.785251324114
Global GM energies  =           0.222629281859
************************************************************
```

One can see that some additional quantities are printed: the total spin moment associated to all the valence electrons, and several energies. The most important value is the `Energy correction`, i.e. the contribution to be added to the RSPt total energy (e.g. from the **hist** file) to obtain the final LDA+U energy.

The last important feature to focus on is that at every iteration the local density matrix in the correlated orbital basis is calculated, leading to some additional information:

```
ID:0202010100 c   Occupation:                        7.812017E+00
ID:0202010100 c   Energy [Tr(HG)]:                   4.945956E+00
                                                        Cartesian        Spin axis
ID:0202010100 c   Spin moment [Sx] (mu_B):           0.000000E+00     0.000000E+00
ID:0202010100 c   Spin moment [Sy] (mu_B):           0.000000E+00     0.000000E+00
ID:0202010100 c   Spin moment [Sz] (mu_B):           1.741119E+00     1.741119E+00
ID:0202010100 c   Total Spin moment [S] (mu_B):     1.741119E+00     1.741119E+00
ID:0202010100 c   Orbital moment [Lx] (mu_B):      -4.427569E-13    -4.427569E-13
ID:0202010100 c   Orbital moment [Ly] (mu_B):       4.423129E-13     4.423129E-13
ID:0202010100 c   Orbital moment [Lz] (mu_B):      -5.600461E-16    -5.600461E-16
ID:0202010100 c   Total Orbital moment [L] (mu_B):  6.258392E-13     6.258392E-13
ID:0202010100 c   Total moment [Jx=Lx+Sx] (mu_B):  -4.427569E-13    -4.427569E-13
ID:0202010100 c   Total moment [Jy=Ly+Sy] (mu_B):   4.423129E-13     4.423129E-13
ID:0202010100 c   Total moment [Jz=Lz+Sz] (mu_B):   1.741119E+00     1.741119E+00
ID:0202010100 c   Total moment [J=L+S] (mu_B):      1.741119E+00     1.741119E+00
ID:0202010100 c   Direction of S (Cartesian):       0.00000000  0.00000000  1.00000000
ID:0202010100 c   Direction of J (Cartesian):      -0.00000000  0.00000000  1.00000000
```

The meaning of the fields is straightforward, apart from the energy, whose discussion is out of the scope of this example. All the quantities are referred to the projection onto the correlated orbitals, in the present case the Ni d-shell of the atomic type 02. In this output all the components of the magnetic moments are printed with respect to both the spin axis reference frame (where the magnetization is basically aligned along z) and the cartesian reference frame. The directions of the relevant magnetic moments are also printed. If there is a finite orbital moment, i.e. if spin-orbit coupling is present, the angle between it and the spin moment is also visualized. Notice that sometimes, for some special settings, this information may be shown for every iteration of the DMFT cycle, even when the chemical potential is not converged. *The values that have a well-defined physical meaning are those for the right chemical potential*, which are labeled by a **c** just after the ID tag.

11.6 unitcell block

The unitcell block specifies that an additional cluster block should be produced, containing all the orbitals in the entire unit cell. Note that it can be very costly to project upon all the atoms in the unitcell, since also their off-diagonal elements are taken into account. The unitcell block looks like

```
unitcell
3                ! basis, [Cf]
```

and the fields have the following meaning:

basis the default local basis, used for ALL orbitals. Be careful if you use anything else than 0.

Cf rotate the orbitals into the best crystal field basis for the FULL unitcell.

The unitcell block is replaced by the generated cluster in the formatted input.

11.7 matsubara block

The matsubara block defines the mesh of the Matsubara frequencies used in the DMFT code. In the LDA+U example such a mesh is trivially reduced to

a single point, since the self-energy is energy independent (static effective potential). However when the self-energy becomes dynamical more points are needed. In RSPt two Matsubara meshes are used. The first mesh is the standard linear Matsubara mesh $\omega_n = \pi T(2n-1)$, where T is the temperature and n is an integer going from 1 to ∞. This is the physical mesh defined in the textbooks, and is used for the proper many-body parts. The second mesh is a linear-logarithmic mesh, and is used for the computationally expensive routines, i.e. the creation of the global Green's function. In such a mesh the first few frequencies are sampled linearly, since the Green's function is varying strongly. Then follows a logarithmic body, where the Green's function is changing slowly. Finally the first few frequencies are again linear, since they are important for the correct asymptotic behavior. Transformation from one mesh to the other one is made by means of interpolation (cubic spline by default, or Laurent spline if the debug flag is activated).

The Matsubara block looks like

```
matsubara
1500 40 120 10                  ! nmats head body tail
```

and the fields have the following meaning:

nmats number of Matsubara frequencies in the linear mesh

head number of linear Matsubara frequencies at the beginning of the linear-logarithmic mesh

body number of logarithmic Matsubara frequencies in the body of the linear-logarithmic mesh

tail number of linear Matsubara frequencies at the end of the linear-logarithmic mesh

When the user wants to set up the Matsubara meshes manually, two guidelines should be taken in mind. First of all `nmats` should be such that $\omega_{\texttt{nmats}}$ is at least 2 or 3 Ry, if the self-energy has a standard shape. Given that the spacing between the frequencies is proportional to the temperature, the lower the latter one is, the higher `nmats` should be. Once the linear mesh has been set-up, the linear-logarithmic mesh should contain enough points to ensure a good interpolation without introducing severe numerical errors. One way to check if the number of Matsubara points are sufficient, is to run a few 1-shot calculations with different number of points in the head and body but starting from the same self-energy. If the number of particles come out correctly in the first iteration then it is a strong indication that you have a converged linear-logarithmic mesh.

11.8 inputoutput block

The inputoutput block regulates the access to the disk and is used to start or resume a calculation. Typically it looks like

```
inputoutput
.true. .true.           ! readsig csc
```

Here the meaning of the fields

readsig read the **sig** file, and after that any **sig-ID** files, and use the loaded self-energy as the starting point of the DMFT cycle, instead of starting from a zero self-energy. The self-energies in the **sig-ID** files are only added to clusters without a self-energy but with a matching ID.

csc after the DMFT cycle has been converged, make one additional iteration for creating the global density matrix, save it on disk, and then use it to generate a new **pot** file

The flag `csc` should be set to true if the user wants to generate a new electron density including the DMFT changes. For simple "one-shot" calculations, such a flag should be set to false.

11.9 convergency block

The convergency block is used for the definition of the criteria determining whether a simulation is converged or not. Typically it looks like

```
convergency
1d-6 1d-4 999 999              ! ndelta sigma_acc maxiter maxsolveriter
```

Here the meaning of the fields

ndelta convergence criterion for the number of valence electrons

sigma_acc convergence criterion for the self-energy function

maxiter maximum number of DMFT iterations before quitting the program

maxsolveriter maximum number of solver iterations before quitting the program

A simulation is considered converged if the difference between the calculated number of electrons in two consecutive DMFT iterations is smaller than `ndelta`, and if the difference between the calculated self-energies in two consecutive solver iterations is smaller than `sigma_acc`. Notice that these parameters are referred to the one-shot DMFT cycle, since the converge of the CSC cycle is handled by the DFT part of RSPt, e.g. by means of the *runs* programs. It is important to stress that the numerical precision attainable with this DMFT implementation is very high, and almost comparable to the standard RSPt precision. However small values of the convergency parameters can result into a big computational effort, and therefore it is important that the user keeps his aims in mind. For example if the user wants to address the spectral properties of a strongly correlated material, values such as `ndelta`=10^{-4} and `sigma_acc`=10^{-3} can be sufficient, while total energy calculations may require a more demanding set up.

11.10 mixing block

The mixing block determines which method the code will use to mix the self-energy. It also contains the parameters used to control the search for the chemical potential. Typically it looks like

```
mixing
3  0.15  0.2                    ! mix_method max_slope sigdiff_factor [green_mu]
```

Here the meaning of the fields:

mix_method There are currently 3 self-energy mixing methods

- **1** Linear mixing
- **2** Broyden mixing: The self-energy is converted to a vector and mixed using the Broyden method. This method can in rare cases produce unphysical self-energies if it severely overshoots a trend.
- **3** Static Broyden mixing: The static part of the self-energy ($\Sigma(\infty)$) is mixed using the Broyden method. The dynamical part is mixed using the same mixing coefficients. This method is less aggressive than pure Broyden mixing and is less likely to produce an unphysical self-energy, but the procedure can sometimes get stuck if the static part of the self-energy converges before the dynamical part.

max_slope This parameter restricts the step size used in the search for the chemical potential. If you have a slow but monotonic convergence you can try to increase this value. If the code overshoots the correct number of electrons at each iteration you can try to decrease this value.

sigdiff_factor If the program is run without charge self-consistency it will try to converge both the self-energy and the number of electrons at the same time. The value of `sigdiff_factor` is multiplied with the current value of the self-energy difference (Sigdiff) to give an additional charge convergence criteria. Set `sigdiff_factor` to zero if you only want `ndelta` to control convergence of the charge.

green_mu The chemical potential is set to the value of `green_mu` before the first iteration of the DMFT cycle or when the spectral functions are printed. This option should be used to get the correct fermi energy in band plots of non-interacting systems.

11.11 projection block

The projection block contains the most important physical approximation related to the LDA+U Hamiltonian, i.e. the shape of the correlated orbitals. In fact all the methods that are based on a given set of correlated orbitals depend arbitrarily on the basis chosen. However if this choice is made wisely, the results will present only weak differences, and this can be verified directly also by means of RSPt. In the present version of the code, two choices are implemented, but in the future other choices are going to be explored, e.g. the usage of the maximally localized Wannier functions.

The projection block looks like:

```
projection
1                              ! basis_type
```

It contains only one field

basis_type atomic-like basis used for the constructing the correlated orbitals from the LMTO basis functions

- **1** so-called "heads of LMTO" (MT), i.e. the radial part of the LMT orbitals inside the muffin-tin sphere and without linearization, i.e. basic solution of the radial Schrödinger equation for an energy corresponding to the center of mass of the band; such a choice possesses the proper angular character and is extremely localized, since the interstitial part is completely neglected; for these reasons it is important to keep in mind that a large muffin-tin radius has to be used, orientatively never less than 0.9 of the maximum radius allowed by the geometry of the problem; the worst problem of this basis is that the Hilbert space spanned by the correlated orbitals is not a subspace of the Hilbert space spanned by the LMT orbitals, and therefore the projection between the two spaces will include projection errors; a good measure of such errors is given by the `Local overlap matrix` printed in the **out** file, and already discussed in the LDA+U example above. In order to have a orthonormal set of orbitals after the projection, the MT orbitals are orthonormalized using Löwdin's orthogonalization procedure for each k-point separately.
- **2** LMT orbitals orthonormalized through the square root inverse of the overlap matrix (natural orbitals); notice that this choice strictly requires the number of tails for the correlated orbitals to be set to 1 in order to obtain an exact correspondence between the RSPt basis and the atomic-like basis; furthermore the angular character is not pure, because of the tails of the neighboring sites; the major advantage of such a basis is that the Hilbert space spanned is a subspace of the Hilbert space spanned by the full RSPt basis, and therefore no projection errors are introduced.

11.12 tensmom block

The tensor moment representation is an alternative description of occupation, instead of the more common m_l and m_s or m_j quantum numbers we transform this into a basis consisting of coupled multipole tensors $\mathbf{w}^{kpr}$ where k refers to the k'th multipole of charge, p refers to charge (0) or magnetization (1), which couple through r. Sometimes this alternative description reveals physics hard to penetrate with the standard representation. Most quantities can be decomposed into tensor moments, the decomposition of the density matrix results in the multipolar occupations, the decomposition of the self energy shows the symmetry of the self energies contribution to the potential etc. When the `tensmom` keyword is present the density matrix is decomposed by default, the dynamic quantities has to be specified within the block. In the current release only the decomposition of the density matrix is available, however, the input block has all features for debugging purposes. The weights specified

in the `tensmom` block are multiplied with the (optional) `tansmom_mag` parameter found in each cluster block. It is in this way possible to apply a symmetry breaking term or a double counting correction to only a selected cluster.

All Decompose density matrix into all allowed multipoles

Tz Outputs the Tz component in the occupations output.

Symbrk Initial symmetry break of the self-energy, useful to explore different solutions.

Active Sets the default weight to 1, used in symmetry break and Dc.

Dc Specify tensor components to remove in the double counting.

Modelham Change the local Hamiltonian to a model hamiltonian with an initial tensor moment configuration.

```
tensmom
Active Symbrk All Dc Modelham         ! tensmom_str
0 0 0                                 ! k p r [w]
0 1 1                                 ! k p r [w]
```

11.13 spectrum

The spectrum block allows us to calculate quantities related to the spectral properties of our system, such as the density of states or the spectral densities. While the main core of the DMFT code works for the imaginary Matsubara frequencies, the spectral properties are print on the real energy mesh, which is defined by the `energymesh` block.

Typically the spectrum block looks like:

```
spectrum
Band eV                ! keywords
 71 50 50 50           ! [gp_nkp]
X  G  L  W  X          ! [gp_labels]
```

The `gp_labels` corresponds here to an **spts** file with the Gamma, L, W, and X points moved to the very beginning of the file.

The keywords in the first line can be divided into two groups, depending on their function. The first group includes keywords specifying what tasks the user has in mind:

Dos print the total dos and the partial dos for the clusters in the following files:

dos.dat contains the spectral density for the full system;

pdos-ID.dat contains the spectral density projected onto each cluster.

The files contain a header describing what corresponds to each column. Notice that the meaning and construction of the ID tag has been described in the cluster section above.

Band print the total spectral densities; if the *gnuplot* format is enabled then the data will be stored in (small) binary files and an easy to use gnuplot plotting script (see below). Otherwise the data will be dumped to (huge) plain text files. The header describes what is contained on each file.

Pband print the partial spectral densities for each cluster type.

Hyb calculate the hybridization function; in case the `Band` or `Pband` keywords are absent, the local hybridization function is calculated; if one of them is present, instead, the hybridization function is given per k-point; note that the sum of the k-dependent hybridization function is not the same as the local hybridization function.

Surf print the spectral density projected onto a surface plane; its usage is described below.

Psurf print the partial spectral density projected onto a surface plane; its usage is described below.

Fermi print the Fermi surface for XCrySDen; at the moment Fermi liquid properties are enforced. Much memory is required to store the data before printing, and therefore a suitable computational environment should be specified, i.e. large memory per node and/or several nodes.

One important thing that the user should keep in mind is that the spectral functions are the generalization of the band structure plots for interacting systems, and therefore cannot be printed simultaneously with the density of states, since it requires a different mesh of k-points. Once one has decided what tasks to perform, then the second group of keywords (still in the first line!) can be used for more personalized output:

Eps make the *gnuplot* scripts save the intermediate eps-files instead of collecting them into pdf-files.

eV use eV instead of Ry as energy unit.

Cf project all quantities onto the irreducible representations of the orbitals in the clusters.

Obs calculate the spectrum only for the observer clusters.

Nospin prevent calculating spin up-down projections in the `Dos` files.

Sproj calculate dos, pdos, bands and pbands projected over Sx, Sy and Sz (spin axis reference frame); this option is activated by default for magnetic systems, unless `Nospin` is found.

Lproj calculate pdos and pbands projected over Lx, Ly and Lz (spin axis reference frame).

Jproj calculate pdos and pbands projected over Jx, Jy and Jz (spin axis reference frame).

Cartesian use the Cartesian reference frame instead of the spin axis reference frame for `Sproj`, `Lproj` or `Jproj`.

Bothaxes use both the spin axis reference frame and the Cartesian reference frame for `Sproj`, `Lproj` or `Jproj` in the `Dos` files; this option is not valid for the bands.

SymmX calculate pdos and pbands projected over the symmetry operator number X (See **symt.log** or **symcof** for the order of the symmetry operators).

Proj calculate pdos and pbands projected over the local orbitals of each cluster.

All the resolved data is appended as extra columns in the DOS and band files. Notice that the projections of the DOS files over the spin up and down directions are calculated automatically, unless `Nospin` is found. The corresponding projections for the band structures can be easily obtained in the *gnuplot* format, by uncommenting the corresponding lines at the end of the **gpi** files. They must instead be calculated by hand if one uses the plain text format.

Printing the band structures can be much simplified by using the *gnuplot* format. This can be activated by adding the last two lines of the spectrum block. They are optional but highly recommended. In these lines, the following fields must be specified:

gp_nkp numbers of k-points *in between* the high-symmetry k-points. If one has generated the **spts** through *kpath*, these values are equal to those used in the file **kpath.inp** and listed in **kpath.log**. If one has generated the **spts** through *lineg1*, instead, these values are equal to those used in **lineg1.inp**, but one point should be removed from each segment (Example: an input of "100 50 100" in **lineg1.inp** would correspond to "99 49 99" in **gp_nkp**).

gp_labels labels of the high-symmetry k-points enumerated in the **kpath.inp** or **lineg1.inp** file. Notice that the total number of labels for the high-symmetry points should be equal to the number of k-point segments (values in the line above) **plus one**.

These guidelines can be used to generate proper values for the fields **gp_nkp** and **gp_labels**. However, we stress that the program *kpath* automatically writes a properly formatted spectrum block in its output file **kpath.log**. Then, to avoid useless complications, **it is highly recommended** to use *kpath*, and copy this information directly from the **kpath.log** file into the **green.inp** file.

As stated above, the **gp_nkp** and **gp_nkp** lines change the output from a text-based raw data file (e.g. **band.dat**) to one binary raw data file (e.g. **band.data**) and one *gnuplot* instruction file (e.g. **band.gpi**). The latter can be directly modified to change the defaults layout and colors used in the band plot. Loading the instruction file (e.g. "*gnuplot* **band.gpi**") generates an encapsulated postscript file for each plot. If the keyword `Eps` is not given, these files are finally collected in several pdf files. The exact number and names of these files depend on the problem under consideration and on the options activated. As a final note, older versions of *gnuplot*, such as *gnuplot 4.2*, use an deprecated syntax for plotting binary data. In this case, one needs to replace the plot argument "array=(N,M)" with "array=NxM" in all the instructions files (e.g. **band.gpi**).

A bit more complex is the input for printing total and partial spectral density projected onto a surface plane. If the keywords `Surf` or `Psurf` are present, one has to add an additional line to the spectrum block:

```
spectrum
Surf Psurf eV           ! keywords
 1 1 0 0.0 50 50 20000  ! Miller indices, hv - E0 in units of bz, k-point bins
 0 0 0                  ! [gp_nkp]
G L W X                 ! [gp_labels]
```

The additional line contains the miller indices of the surface, the photon energy (hv) minus the work function (E0) (a negative value disables momentum conservation), and the number of k-point bins along the reciprocal surface vectors (See the example above). A good plot requires in principle a k-point mesh perfectly adapted to the k-point bins. Such k-point mesh, and the corresponding spectrum block input, is generated by the program *ksurf*, which functions much in the same way as *kpath* does. A surface plot calculation with the `Surf` keyword generates the following files:

surface.data Contains the raw data in binary format.

surface.gpi A *gnuplot* script to generate EPS-files from the raw data.

surface.dat Contains the unreduced raw data in text format. The first two columns are the k-point coordinates, and the remaining are the energy resolved spectral intensities. This file is only produced if the number of the k-point bins in some direction is set to a non-positive number.

In case of `Psurf`, instead, one obtains the following files:

psurface-ID.data Contains the raw data in binary format.

psurface-ID.gpi A *gnuplot* script to generate EPS-files from the raw data.

11.14 carriers

The carriers block activates the routines to obtain the relative carriers concentrations from the total and local densities of states. It requires the corresponding values in the spectrum block.

Typically the carriers blocks looks like:

```
carriers
-2                      ! doping
```

The value of the doping field specifies the number of holes (negative values) or electrons (positive values) which we determine the band edge from. This is done for those materials, such as dilute magnetic semiconductors, that have a metallic character, and therefore not a well defined band edge.

11.15 modelexchange

The `modelexchange` block creates a local exchange field on a given set of correlated electrons as due to another set of correlated electrons. The interaction is applied as an additional double counting correction, so only clusters with a solver will be affected. It requires that both sets are associated to corresponding cluster blocks. Typically, the `modelexchange` block looks like:

```
modelexchange
Sm3d Sm4f 0.100                ! label1 label2 Idf
Ni4p Ni3d 0.080                ! label1 label2 Idf
```

Here label1 and label2 are the labels of the two cluster blocks connected by the local exchange-field. To get convenient labels, use the Id keyword in each cluster block. The last value `Idf` determines the size of the Stoner I correction (in Ry). Several lines may be included in the model exchange block.

11.16 verbose

The purpose of the verbose block is to increase the verbosity of the information printed in **out** file. The currently available options are listed below, and all must start with a capital letter. Notice that several other verbose options are available depending on the solver chosen, and they will be described in the sections dedicated to the solver themselves.

Readin it writes out the raw input file as seen by RSPt in order to check for possible errors in the **green.inp** file

Interface it writes out the parameters collected from the DFT part of RSPt

Mesh it writes more information in relation to the meshes of the Matsubara frequencies and of the real energies

Projection it writes out the details of the second projection matrix and the creation of the correlated orbitals

Umatrix it writes all the elements of the U-matrix

Cycle it writes out a few additional data during the DMFT cycle

Chemical it writes out more detailed information in the routines where the chemical potential is calculated

Symmetries it writes out the Euler angles corresponding the group operations and the rotation matrices used during the symmetrization of the local Green's function

Sigma it writes out information about the reading and writing of the self-energy; moreover the self-energy is written on file at every solver iteration for both Matsubara frequencies (sig-ID.dat) and real energy axis (sig-realaxis-ID.dat); the columns contain respectively energy, total trace of the correlated orbitals, trace for majority spin, trace for minority spin, and then all correlated orbitals for majority spin and minority spin

Greens_functions it writes out the characteristics of the local Green's function at the beginning of the routine

Solver it writes out additional information while running the solver, and the verbosity of the output depends on the solver itself

Dc	it writes out detailed information on the calculation of the double counting
Mixing	it writes out several matrices related to the mixing of sigma between different iterations
Dump	it writes out the names of the files used when writing self-energies, Green's functions, etc, to disk
Pade	it writes out additional information on the analytical continuation when done through the Padé approximant method
Tails	it writes out several quantities calculated during the treatment of the long-decaying tails of the Green's function
Spectrum	it writes out several quantities used while calculating the spectral properties
Symbrk	it writes out the potential matrices used for breaking the symmetry through the tensor moment expansion

11.17 debug

The purpose of the debug block is to run RSPt with less tested options, usually for debugging purposes or for ad-hoc problems. As for the verbose block all keywords must start with a capital letter. Currently implemented keywords are:

Laurentspline	the interpolations are performed by means of the Laurent spline instead of the cubic spline
Chemical	a lot of information related to the finding of the correct chemical potential is printed at the end of the simulation
AFS	the self-energy file is saved only at the end of the simulation, and not at every iteration
Calcdmtx	the density matrix is calculated at every iteration, and not only once the self-consistence has been reached
Modelham	a special double counting is activated for removing contributions which possess certain symmetries
Notails	the fancy tails fitting of the self-energy and the Green's function at high frequencies is deactivated
Solver	it turns on the debugging mode for the solver, and its behavior depends on the solver under consideration
Masksigma	the self-energy is allowed to have finite elements only for non-zero elements of the initial LDA local density matrix (a mask of boolean values is constructed and stored in a file called mask_to_sigma.mat)

Noscreening the Slater parameters calculated by RSPt are by default rescaled to about 80% as done in the Hartree-Fock community. For d-orbitals F2 and F4 are multiplied with 0.82 and 0.88, respectively. For f-orbitals F2, F4, and F6 are multiplied with 0.92, 0.97, and 1.0, respectively. These values were obtained from comparing the bare U values of NiO to a constrained RPA calculation, and the spectral functions of $SmSn_3$ and $YbPd_2Sn$ to experimental photoemission data. If you for some strange reason want to use the bare values, you need to set the `Noscreening` keyword.

U2 only the density-density contributions are retained in the 4-index U-matrix

Simplifiedu a simplified 4-index U-matrix is used with the parametrization in U, U-2J and U-3J

11.18 energymesh block

The purpose of the `energymesh` block is to define the mesh of real energies that is used when calculating the spectral properties such as the density of states or the spectral functions. In fact while the Green's function involved in the DMFT cycle is calculated for the imaginary frequencies on the Matsubara axis, the proper physical observables are calculated for real energies. Given that the Green's function has poles along the real axis, the analytical continuation should be done at a finite distance `eim` from the real axis itself. The proper values of the observables are then obtained in the limit of `eim` going to zero. In practice, however, `eim` is usually fixed to be between 1 mRy and 10 mRy, depending on the solver employed.

The energymesh block looks like:

```
energymesh
1001  1.0  -1.0  0.010          ! nemesh emin emax eim
```

The meaning of the fields is the following:

nemesh number of points contained in the energy interval between `emin` and `emax`

emin minimum energy to calculate

emax maximum energy to calculate

eim smearing parameter which measures the distance from the real energy axis.

11.19 solvers

The following sections contain a description of all the solvers with their additional lines and their debug and verbose flags.

11.19.1 SPTF

The SPTF solver[16] is a perturbative solver that can be applied to systems with moderate correlations, in the metallic regime of the Mott-Hubbard transition. The basic guideline that must be followed when using SPTF is that the value of the applied Coulomb interaction should never be bigger than the value of the effective half-bandwidth.

The SPTF solver requires an additional line which specifies the running mode and other parameters. The additional line is such as:

```
5 100 1e-5 0.5d0                ! sptf_mode [sptf_niter sptf_conv sptf_mix]
```

The meaning of the fields is the following:

sptf_mode this is the most important parameter and specifies the running mode:

- **1** standard non-conserving SPTF where the self-energy is obtained as a perturbation in the bare G_0
- **2** non-conserving Hartree-Fock approximation where the self-energy is obtained as a perturbation in the bare G_0
- **3** conserving Hartree-Fock approximation where the self-energy is obtained as a perturbation in the renormalized G_{HF}
- **4** partially conserving SPTF where the self-energy is obtained as a perturbation in the renormalized G_{HF}
- **5** (hopefully) conserving SPTF where the self-energy is obtained as a perturbation in the fully renormalized G

sptf_niter maximum number of internal iterations of SPTF when converging the full Green's function

sptf_conv convergence required in the self-energy before quitting the cycle

sptf_mix linear mixing of the self-energy between two consecutive iterations.

The last three fields must be specified if `sptf_mode` has been set to 3, 4 or 5.

The SPTF double counting procedure requires one optional extra parameter

```
False     ! [spin_average]
```

spin_average Take the average of the two spin channels.

Further the SPTF solver allows the following debug options:

Sptf_badtail the tails of the Green's function are calculated by fitting instead than explicitly from the local Hamiltonian (less precise but exportable in codes with no direct access to the local Hamiltonian)

Sptf_single_model when calculating the Fourier transforms, only one model function is used instead of two (less precise Fourier transform but compatible with the previous version of SPTF)

Sptf_new the functional written in the most recent SPTF paper is used instead of the functional programmed by Pourovski in the former version of the solver; notice that this option has not been tested extensively

11.19.2 Exact diagonalization

The Exact Diagonalization (ED) solver is a non-perturbative solver and can therefore handle arbitrary correlation strengths. The restriction is instead that the impurity hybridization function must show distinct peaks in the energy resolved spectral density, as it is fitted using a small number of auxiliary bath orbitals.

The ED solver needs a few extra parameters

```
4 0 2 3.0 6 60.0  ! ed_nelec, ed_nextra, ed_nenvextra, ed_n, ed_nfit, ed_sweight
```

The meaning of the field is the following:

ed_nelec Number of electrons in the ground state (integer).

ed_nextra Extra electrons in the ground state (integer). If you have an intermediate valence ground state, set ed_nelec to the smallest integer number of impurity electrons, and extend the allowed ground state configurations by ed_nextra number of electrons. For example, SmB_6 has an intermediate valence ground state containing eigenstates with both 5 and 6 electrons. This can be described by setting ed_nelec = 5 and ed_nextra = 1.

ed_nenvextra Extra electrons and holes in the bath orbitals. If the solver produces an unphysical self-energy but the trace of the density operator is still very close to 1, then try to increase this number by one. If it is set too low then the hybridization to the bath is not properly described in the many-body problem.

ed_n Number of auxiliary bath orbitals per correlated orbital in the ED Hamiltonian.

ed_nfit Number of auxiliary bath orbitals per correlated orbital used in the fit of the hybridization function. Must be larger than `ed_n`.

ed_sweight Weight parameter used in the function which ranks the fitted bath states according to their physical relevance. A higher value puts more emphasis on the low energy states close to the Fermi energy.

The ED double counting procedure requires one extra parameter (and one optional)

```
0.2 0.0  ! ed_eshift, [ed_eextra]
```

ed_eshift Energy position of the first peak.

ed_occ The occupation of the impurity. If ed_eshift = 0, then the double counting potential will be set to make the occupation of the impurity ed_occ.

11.19.3 CT-QMC

The Continous Time Quantum Monte Carlo solver is formally exact in solving the impurity problem. It is able to handle the full range of bandwidth vs coulomb interaction strength. Currently only a version with the density-density approximation is implemented, meaning that for example spin-orbit coupling will not be treated exactly. Up to now, it has only been tested with a simplified U to activate with the debug keyword `Simplifiedu`. Moreover, the Hamiltonian matrix has to be diagonal, and this should be checked for any problem one intends to study. For CT-QMC, the additional solver line may look like:

```
2d8 10 128 2 .true. .true.    ! nsweeps nmcarlo tail_freq tail_type pm_symm orb_symm [symm_array]
```

where the variables stand for:

nsweeps Total number of Monte Carlo sweeps per rank.

nmcarlo Period of data sampling.

tail_freq Cut-off frequency for the QMC data; the rest of the data are constructed from the tail

tail_type Type of tail to attach, two options available:

1. Build high frequency self-energy function using moment expansion, and adjust impurity green's function by using Dyson's equation. Suitable for weak correlation region
2. Build atomic green's function and self-energy function using improved Hubbard-I approximation, then make interpolation for self-energy function between low frequency QMC data and high frequency Hubbard-I approximation data, the impurity green's function is then obtained by using Dyson's equation.

pm_symm Apply spin symmetry or not.

orb_symm Apply orbital symmetry or not. The symmetries are directly taken from the symmetries of the local Hamiltonian, unless the optional array is given.

symm_array Array of labels (integers) describing the symmetries of the final Green's function. Equal labels correspond to equivalent orbitals. Example for 6 spin orbitals: 1 1 2 3 3 4.

.

11.20 Spin-polarized simulations in Hubbard I

The Hubbard I solver is usually used without spin polarization, since it is not necessary to have a broken symmetry phase. However, it can also be used for spin polarized simulations, by means of different strategies and approximations.

Starting from a simulation **without** spin-polarization, one can use the fully localized limit double-counting (2 or 4; for the latter the optional parameters should be set to 0 and N respectively) and the debug flag `Spin_avg_dc`. The initial symmetry must be broken by means of a small magnetic field (the smallest, the better). The temperature of the simulation should be small enough to avoid superpositions of the different configurations, whose energy splittings are given by the Hund's exchange J. These simulations are reasonable but they require an unphysical magnetic field and do not lead to a spontaneous magnetization. The reason lays in the fact that in Hubbard I the correlated orbitals cannot communicate with the rest of the lattice, due to neglecting the hybridization. A different strategy, which does not present the previous inconveniences, consists in using the Hubbard I double-counting (-1 option in combination with Hubbard I solver) and the `modelexchange` block to model the effective exchange between the correlated electrons and the rest, as described in Phys. Rev. B **89**, 205109 (2014) for TbN. In this case one should make sure that a proper basis is used for both type of electrons involved in the model exchange. Here we refer to these electrons as d- and f-states respectively. Now the initial symmetry must be broken by means of the `Symbrk` option with the tensor moment index corresponding to the spin moment (0 1 1). This initial magnetic moment for the d-states creates a local exchange field acting on the f-states, and the simulation can then be relaxed to the equilibrium symmetry-broken solution. Notice that a solver should be applied also to the d-states, but with a very small U and a mixing of about 0.5, in order to slowly decrease the initial field driving the formation of the magnetic moment. Notice that this approach is extremely sensitive to the temperature, due to the very tiny self-consistent magnetic moment of the d-states.

On the other hand, starting from a simulation **with** spin-polarization, one can again use the fully localized limit double-counting (2 or 4; for the latter the optional parameters should be set to 0 and N respectively). This time no additional tricks are needed, but some care should be taken when choosing the value of J. In fact, if J is too big, the double counting correction associated to the spin polarization may be larger than the Hamiltonian spin splitting coming from LDA. If this is true than the system with start oscillating between magnetization with opposite signs, leading to non sense. Moreover a very low temperature is required to resolve the splittings between the configurations, as mentioned in the previous paragraph. If these guidelines are followed, the simulations should be straightforward. This strategy was tested for 1-shot DMFT simulations, but it should be applicable to CSC simulations too.

11.21 Calculation of the inter-site exchange parameters (J_{ij})

One can evaluate the inter-site exchange parameters using the Lichtenstein-Katsnelson-Antropov-Gubanov (LKAG) method [17]. Original method is formulated for the non-relativistic case, where the inter-site exchange interaction is isotropic. Later, a fully relativistic extension of the method has been done by Udvardi *et al.* [18], Ebert and Mankovsky [19] and Secchi *et al.* [20] by expanding up to 2nd order in the perturbation (i.e. small spin rotations). In this case, the interaction between any pairs of spins takes a tensorial form. Both formulations are implemented in RSPt and the details are given in these papers [21, 22].

11.21.1 Non-relativistic mode

In the non-relativistic case (`fullrel`=F in the **data**), one can map the magnetic system on an isotropic Heisenberg model:

$$\hat{H} = -\sum_{i \neq j} J_{ij} \vec{e}_i \cdot \vec{e}_j, \tag{45}$$

where $\vec{e}_i$ denotes the unit vector along the magnetic moment at the site i. Note that according to the current definition, each pair of spins is counted twice. In this case the exchange parameter between site i and j are defined in the following way:

$$J_{ij} = \frac{\mathrm{T}}{4} \sum_n \mathrm{Tr} \left[\hat{\Delta}_i(i\omega_n) \hat{G}^{\uparrow}_{ij}(i\omega_n) \hat{\Delta}_j(i\omega_n) \hat{G}^{\downarrow}_{ji}(i\omega_n) \right] \tag{46}$$

where T is the temperature, Δ is the on-site exchange potential, G_{ij} is an inter-site Green's function and $i\omega_n$ is the n-th fermonic Matsubara frequency. Note that all terms entering this expression are matrices in orbital and spin space. Eq. (46) is implemented in RSPt and gives a possibility to extract the J_{ij}'s from DFT and DFT+DMFT calculations.

11.21.2 Relativistic mode

In the relativistic case, the interactions become rank-2 tensors ($J^{\alpha\beta}_{ij}$) and the generalized Heisenberg Hamiltonian of the following form has to be considered:

$$\hat{H} = -\sum_{i \neq j} \sum_{\{\alpha,\beta\}=\{x,y,z\}} e^{\alpha}_i J^{\alpha\beta}_{ij} e^{\beta}_j, \tag{47}$$

where e^{α}_i is the α component of the unitary vector pointing along the direction of the spin located at the site i. For instance, one of the diagonal terms is calculated as:

$$J^{xx}_{ij} = \frac{\mathrm{T}}{4} \sum_n \mathrm{Tr}_{L,m} \big[\hat{\mathcal{H}}_i(i\omega_n), \hat{\sigma}^y \big] G_{ij}(i\omega_n) \big[\hat{\mathcal{H}}_j(i\omega_n), \hat{\sigma}^y \big] G_{ji}(i\omega_n), \tag{48}$$

where square brackets denote commutator, σ are the Pauli matrices, $\hat{\mathcal{H}}_i$ is the local potential which contrains both Hamiltonian and self-energy contribution. Note that in the 2nd order formulation, only transverse components of the tensor with respect to the spin direction can be extracted. Therefore, if the spin is located along Z axis, then only J^{xx}_{ij},J^{xy}_{ij},J^{yx}_{ij},J^{yy}_{ij} components can be accessed. (The rest terms are of higher order in spin rotations and can not be mapped.) This is discussed in detail in Refs. [18, 22]. It is thus adviced to run 3 separate calculations, where the magnetization is pointing along X, Y and Z directions in order to construct the full interaction tensor. From the elements of the tensor specified above, one can form anti-symmetric and symmetric parts as follows:

$$\vec{D}^z_{ij} = (J^{xy}_{ij} - J^{yx}_{ij})/2, \tag{49}$$

$$C^z_{ij} = (J^{xy}_{ij} + J^{yx}_{ij})/2, \tag{50}$$

where D_{ij}^{z} is the z-th component of the Dzyaloshinkskii-Moriya (DM) vector [23]. Note that the symmetric anisotropic exchange C_{ij} does not transform as vector.

The code computes these quantities in local coordinate system (where Z is along spin axis) as well as in global (Cartesian) one. If `fullrel` flag is set to "T" in the **data**, then the code will automatically compute the relativistic exchange tensor.

11.21.3 How to compute inter-site exchange in RSPt

In order to perform such calculations one has to:

1. Converge a self-consistent calculation for a certain k-point mesh.
2. Link RSPt to the same k-point mesh, but spanning the entire BZ. (This is described in the following section)
3. Prepare an input-file for the J_{ij}'s (a part of the **green.inp**).
4. Run one iteration of RSPt.

The **green.inp** file has to be augmented with few more lines and eventually will look like:

```
matsubara
1000 50 50 0

inputoutput
T F                                ! readsig csc

cluster                           ! DMFT cluster block
1 eV UJ
1 2 1 1 0 2.0 0.8
1 3 0.9

! Jij-related information:
isoexch
1                                 !  jij-proj
1                                 !  site-i
5                                 !  Nngh, [if negative, reads Rmax]
2 1 0 1                           !  site-j , R_ij
1 1 1 0                           !  site-j , R_ij
2 2 0 1
1 0 0 1
2 2 1 1

cluster                            ! jcluster
2 0
1 2 1 1 0
1 2 1 2 0
```

The meaning of the parameters following the keyword `isoexch` is the following:

jij-proj Projection scheme used for the construction of the localised basis set, centered at the sites i and j to evaluate J_{ij} between them. Possible choices:

1. Use the same projection scheme, as specified in `projection` block of **green.inp**.
2. Use the generalised multiple-κ formulation of Löwdin projection scheme (extension of BRIANNA's `projection`=2, which can be used only for the J_{ij}'s).

site-i Global index of the central site (i in the J_{ij}), according to `lda_basesite(i)%a` array. Set a verbose flag `Interface` to find out.

Nngh Number of neighbours to calculate the exchange interaction with.

site-j Global index of the neighbouring site (j in the J_{ij}), according to `lda_basesite(j)%a` array.

$\mathbf{R}_{ij}$ Translational vector of the site j from its initial position in the unit cell, specified in the units of the Bravais lattice vectors. This vector has three integer components.

jcluster A single standard "observer cluster" block (with `udef` set to 0), which contains the orbitals to be included in the calculation. The indices "**t**" and "**site**" should correspond to the sites i and j.

An example above was made for an hcp Co. There are two atoms in the unit cell, which have the same "**t**", but are characterised with different "**site**" indices. The exchange interaction mostly arises from the d-electrons, thus we have two entries in the `jcluster` block: Co_1-d and Co_2-d states. Note that in most cases, the global index of a site does not have to coincide with its "**t**" index, specified in the `cluster` block.

In the above-mentioned example we have chosen to read the self-energy, coming from the DMFT calculation, in order to be used for the J_{ij} evaluation. This choice was controlled by setting the `readsig` flag to `T` and by keeping the cluster block from the self-consistent DMFT calculation.

11.21.4 K-mesh generation

Current implementation of the Lichtenstein's formula requires to have a k-point mesh, spanning the entire Brillouin zone.[¶] For this purpose one has to generate a separate **symcof** file, called (for instance) **lowsym** where only identity operator is specified. Here is an example of such a file:

```
 (2i6)
    1     2 group order, spins
 (/ 3f18.14 / 3f18.14 / 3f18.14)
c Element  1
  1.00000000000000  .000000000000000  .000000000000000
  .000000000000000  1.00000000000000  .000000000000000
```

[¶]This is needed for the evaluation of the inter-site Green's function, which does not necessary obey all point-group symmetry operations.

```
  .000000000000000  .000000000000000  1.00000000000000
 (3i6)
<...>
```

Once this is done, one runs the *cub* program, using this file as an input and discarding the application of an inversion symmetry :

```
 Cub: produce brillouin zone points for Fourier quadrature

 input data file is cub.inp ...
 enter group file name [../symcof]: lowsym

 include time reversal by adding inverse? y/[n] n
<...>
```

The produced **cub.k.N** file should then be linked to the **spts**. Note that the newly created **lowsym** is used solely to generate a new **spts**. Thus, the original **symcof** file should still be linked to the working directory. After that, one can proceed to the evaluation of the exchange parameters.

Note that the potential is converged on a certain grid of k-points. Therefore, it is absolutely necessary to use the same k-point grid (i.e. divisions and offsets) in the self-consistent calculation and for the one-shot J_{ij} calculation.

One has also to check the **cub.inp** file and especially the "M" matrix used for the k-point generation. For certain symmetries it produced over-sampled and under-sampled regions in the BZ. If the symmetry of the lattice is lower than cubic, it is recommended to bring the "M" matrix to the identity form. Together with a proper choice of the divisions in the BZ, a uniform mesh of the k-points is guaranteed.

11.21.5 Automatic generation of the list of neighbours

There is a possibility of creating the list of neighbour automatically using RSPt. In this case, the `isoexch` block in the **green.inp** file should look as follows:

```
isoexch
1                                  !  jij-proj
1                                  !  site-i
-3.0                               !  Rmax
2                                  !  Ns
1 2                                !  Rmaxsites[1:Ns]
```

where:

Rmax the radius of the cluster around `site-i`, where the neighbours will be searched. It is specified as a negative number in the units of a **length_scale**.

Ns the number of magnetic sites to be included in the search.

Rmaxsites the global indices of the magnetic sites to be searched within `Rmax` radius.

In the above-mentioned example, all exchange interactions J_{1j}'s with the sites j belonging to the sublattices 1 and 2 within a radius of $3a_{lat}$ will be calculated. The program will generate the neighbours, group them by symmetry and run the calculation for a list of inequivalent neighbours. (In the fully relativistic mode, the symmetrization is skipped and the interaction tensors with every neighbour is calculated.) The multiplicity of each neighbours will be printed in the **out** file. It might be needed, for instance, for estimation of the T_c.

If the list of neighbours is generated automatically, then it is also possible to produce an input file for atomistic spin dynamics package "UppASD". In order to obtain it, a `verbose` flag called `UppASD` should be added to the **green.inp** file. Present option will generate a new file called **jfile-N.dat**, where N denotes the global index of a central atom. Note that if several magnetic sublattices are present in the system, a separate calculation for each sublattice has to be performed. Eventually, the complete **jfile** will have to be created by merging all **jfile-N.dat** files. Do not forget to include all magnetic sublattices in the search.

11.21.6 Few general remarks

Generally, different projection schemes will provide different results for the J_{ij}'s (see Ref. [21] for details). This is so, because there is no unique procedure to construct the localised basis. However, if one encounters large differences in the obtained results, it is possibly a signature of the delocalisation of the magnetisation density in the material and inapplicability of the Heisenberg model in this case.

It is preferable to use an extended LMTO basis set with multiple-κ, describing the correlated orbitals. Using the basis set containing only negative kinetic energy tails results in rather robust determination of the exchange parameters, weakly dependent on the choice of MT-spheres. Our experience suggests using Löwdin-projected (`jij-proj=2`) orbitals for metallic $3d$ systems. On the other hand, insulating oxides are usually better described with the "heads of LMTO" scheme (set both `projection` and `jij-proj` to 1).

The precision of the computed J_{ij}-parameters is controlled by the number of k-points and Matsubara frequencies, used in the calculation. Typically, a sufficient number of k-points is the one which guarantees the convergence of the magnetic moments up to $10^{-3}\ \mu_B$. Remember that long-ranged interactions with very distant neighbours might require much denser grids. We strongly recommend checking this before proceeding to any analysis of the results.

The situation is much simpler for the Matsubara mesh convergence. Since each Green's function at high frequencies behaves as $\sim 1/i\omega_n$, the quantity under summation in Eq. (46) decays at least as fast as $\sim 1/(\omega_n^2)$. Hence, usually the calculation of the J_{ij}'s requires less number of Matsubara frequencies than the calculation of the self-energy within DMFT. For room temperature (300K), the following number of frequencies:

```
matsubara
1000 50 50 0
```

provides sufficiently converged results for most systems. Note that when calculating the J_{ij}'s, the tail part of the mesh can be omitted, since the entire integral is taken numerically.

11.21.7 Orbital-resolved exchange parameters

It can be shown that Eq. (46) can be simplified if $[\hat{\Delta}_i]_{mm'}$ is diagonal in m. In this case one can identify the exchange integral between orbital m_1 on site i and orbital m_2 on site j and denote it as: $J_{ij}^{m_1 m_2}$. The latter is defined as follows:

$$J_{ij}^{m_1 m_2} = \frac{\mathrm{T}}{4} \sum_{n=-\infty}^{\infty} [\Delta_i(i\omega_n)]_{m_1} \cdot [G_{ij}^{\uparrow}(i\omega_n)]_{m_1 m_2} \cdot [\Delta_j(i\omega_n)]_{m_2} \cdot [G_{ji}^{\downarrow}(i\omega_n)]_{m_2 m_1} \quad (51)$$

Hence, the sum of all the orbital-resolved contributions gives the total inter-site exchange integral:

$$J_{ij} = \sum_{m_1, m_2} J_{ij}^{m_1 m_2}. \quad (52)$$

For instance, in bcc Fe the exchange splitting ($\hat{\Delta}_i$) has a diagonal (and purely real) form in the basis of cubic harmonics. In case of LSDA we obtain:

d_{z^2}	$d_{x^2-y^2}$	d_{yz}	d_{xz}	d_{xy}
0.15856	0	0	0	0
0	0.15856	0	0	0
0	0	0.12927	0	0
0	0	0	0.12927	0
0	0	0	0	0.12927

(53)

Thus, such a basis forms a natural set of physical orbitals, which interact with each other.

The situation is a little bit more complex, when the symmetry is not cubic. In this case one has to find some basis, where the exchange splitting becomes diagonal in orbital space. Hence, one has to perform the following transformation:

$$\hat{\tilde{\Delta}}_i = \hat{U}_i^{\dagger} \hat{\Delta} \hat{U}_i, \quad (54)$$

where $\hat{U}_i$ is a site-dependent unitary transformation matrix, which provides us with a $\tilde{\Delta}_i$, which is diagonal. Note that in the current implementation the transformation matrix $\hat{U}_i$ is assumed to be frequency-independent.

If one specifies **verbose** flag `Jij_matrix`, then the $J_{ij}^{m_1 m_2}$ matrix will be printed for each neighbour. If the symmetry is not far from cubic (e.g. in slightly distorted perovskites), then one can also extract the couplings between E_g-like and T_{2g}-like orbitals on different atoms. In order to activate the decomposition of the J_{ij}'s onto E_g-like and T_{2g}-like terms, one has simply to add a **verbose** flag `Jij_Eg_T2g`. It works only for the d orbitals in a single-l representation (i.e. `jij-proj=1` for multi-tail basis or both `jij-proj` for single-tail basis).

12 Other special calculations

12.1 Fixed spin moment calculation

It is possible to perform runs with spin moments constrained to be some particular number. There are several different choices for which moment to constrain, either the total moment, the interstitial moment or the moment inside

any given muffin-tin. In any case the program will attempt to enforce a given moment with a constant constraining field which is calculated in each electronic step using a PID (Proportional, Integral, Derivative) controller. The control signal (in our case a constraining field, V^{constr}) is a sum of three terms proportional to: the error, the integral of the error and the derivative of the error. In our case it turns out that it is more stable to have the integral term represented by the current control signal rather than the integral of past errors. The form of the constraining potential for SCF iteration step $n+1$ is given by the PID controller in terms of the deviation of the moment from the desired value, $e_n = \mu_n^{fix} - \mu_n^{actual}$, as

$$V_{n+1}^{constr} = \frac{1}{D(E_F)} \left[K_P e_n + K_D (e_n - e_{n-1}) \right] + K_I V_n^{constr}, \tag{55}$$

where $D(E_F)$ is the density of states at the Fermi level. The three tuning parameters in this expression, K_P, K_I and K_D, are known as the proportional gain, integral gain and derivative gain. The factor of $\frac{1}{D(E_F)}$ is a crude attempt to account for the expected response of the electrons to the external field with a model susceptibility. The rationale for this is simply the relation between the magnetization and the magnetic field, here in differential form

$$\Delta H = \frac{\Delta M}{\chi} \propto \frac{\Delta M}{D(E_F)} \tag{56}$$

where the last proportionality is true for a rectangular DOS around the Fermi level. In such a case the proportional term divided by the DOS at the Fermi level gives the correct field (with the remaining proportionality constant being contained in the proportional gain).

The default values for the gains typically work, but it is sometimes necessary to play around with the different gains to obtain a stable solution. This may seem to indicate that the results of such a scheme are not reliable, but this is not true. Results that have converged in both moment total energy are perfectly fine, it is just harder than usual to get convergence.

There is also an extra contribution to the total energy coming from the constraining field, given by

$$E^{constr} = -V^{constr} \mu^{SCF}. \tag{57}$$

Important. *Because of the additional energy from* (57) *the total energy in a constrained calculation is not correct until the moment has converged to its fixed value.*

As mentioned, the program will give default settings of the PID gains. These proved useful in a small set of test cases used in development. There is a plethora of texts that discuss PID controllers available on the internet, so if trouble arises, the reader is encouraged to have a look around. Quoting Wikipedia for simple rules for what the impact is of the different gains:

Proportional gain, K_P: Larger values typically mean faster response since the larger the error, the larger the proportional term compensation. An excessively large proportional gain will lead to process instability and oscillation.

Integral gain, K_I: Larger values imply steady state errors are eliminated more quickly. The trade-off is larger overshoot: any negative error integrated during transient response must be integrated away by positive error before reaching steady state.

Derivative gain, K_D: Larger values decrease overshoot, but slow down transient response and may lead to instability due to signal noise amplification in the differentiation of the error.

12.1.1 Comments and considerations

- The constrained calculation is often extra sensitive to the convergence of the k-point mesh. This is because we are typically pushing peaks in the DOS through the Fermi level with our additional field (recall the Stoner theory of ferromagnetism), and this can lead to unstable behavior if the peaks are not properly resolved.
- We always have $\frac{\partial E}{\partial m}\big|_{m=0} = 0$ (m is the spin moment), because the energy is a smooth and even function of the moment. If $m = 0$ is a maximum, it can be very difficult to get convergence when constraining to small moments. Remember that the point $m = 0$ is always available by not seeding with a spin polarized starting guess, and since the curve has a known shape in the region interpolation to zero is easy.

13 Utility programs

13.1 RPST

RPST (RSPT Python Setup Test) is a separate program designed to simplify the generation of the input files to RSPT. It is written in Python 2.x, and includes a graphical user interface (GUI) which utilizes Python's TkInter module. There are two main ways to run RPST, either in batch mode from the terminal, or in graphical mode using the GUI. The batch mode is preferable if a large number of different calculations should be set up from an already existing calculation or default setup. The graphical mode is the most efficient way of setting up a calculation from scratch or to change an old setup. Before we go into detail how these different modes work we should have a look at the different input file formats RPST can parse.

13.1.1 The RPST format

RPST can handle different input formats, like the default formats of **symt.inp** and the **dta** files, or the LMTO47 format of INIT. Apart from these external formats RPST has also an internal format which uses a set of keywords. The keywords should given on the form: keyword = value, and separated by a new line or semicolon (;). The keywords can also be added to the first commented lines in **symt.inp** file to provide extra information to RPST.

In symt.inp:

length The length scale

nharm Number of harmonics to expand the potential

kpoints k-point mesh. Should be given as a list [kx,ky,kz]

atommix Determines the mixing ratio of the atomic densities.

supercell Multiplies the lattics and basis vectors to make a SC. Should be given as a list [i1,i2,i3]

In symt.rpst:

bravais Bravais lattice: [[x1,y1,z1],[x2,y2,z2],[x3,y3,z3]]

spinaxis Spin axis: [[x,y,z],'a' or 'l']

datavectors Data vectors: [[x,y,z],Z,'a' or 'l',id.]

strainmatrix Strain matrix: [[x1,y1,z1],[x2,y2,z2],[x3,y3,z3]]

supercell See above.

In data files:

In element_

s Radius of MT sphere.

coord The unit of s.

core The core states. Given as a list of lists.

bases The basis. Given as a list of lists.

basisflags The basis flags. Also a list of lists.

In fouriermesh

ft fourier mesh: [ft1,ft2,ft3]

ft1 optional (see ft)

ft2 optional (see ft)

ft3 optional (see ft)

In length_scale

length length scale

In brillouin_zone_integration

method 2 for smearing, 1 for tetra

In header

zval Total Z

icorr Correlation method, 02 = LDA, 24 = GGA

Table 2: The RPST command line arguments.

-h	–help	Displays a long help message and exits
-i	–interactive	Open an interactive session (requires TkInter).
-s	–symfile	symt input file, either in .inp or .rpst format.
-d	–dtadir	the **dta/** input directory, containing dta input files.
-b	–brianna	Generate brianna files (Not fully implemented yet).
	–symt	Stop after symt is done.
	–data	Skip the symt step.
	–nokpoints	Do not generate a new k-point file

pmix Mixing

f-rel Spin-orbit coupling

sp-po Spin polarization

In paneldata

nsets Number of energy sets

The expert user may include any keyword on the form:

```
mainkey:subkey = val
```

where mainkey is the name of the print out class, subkey is a keyword, and val is its corresponding value. For example:

```
element_69:s = 2.95; element_69:coord = a
```

13.1.2 Batch mode

The batch mode is controlled by various command line arguments to RPST. These command line arguments are summarized in Table 2.

13.1.3 Graphical mode

The graphical mode consists of three tabs: files, dta, and symt. Each tab has several data entries where the user may change or add information. The information added in this way is only stored in an temporary database until the Save button is pressed. It is then stored in RPST's internal database. Any changes may therefore be revoked by pressing the Reload button, which tells RPST to load all values from it's internal database. The input files to RSPT are generated first when the Run button is applied. All data entries correspond to the keywords used in the batch mode.

files tab The files tab contains two data entries. The uppermost specifies the path to the symt.inp/INIT/symt.rpst file. The second entry contains the path to the default (**dta/**) directory.

dta tab This tab can only be accessed if there is a **dta/** directory available. It will then contain data entries corresponding to the dta files keywords.

symt tab The lay out of the symt tab mimics that of a symt.inp file. The rows labeled Extras should contain symbols or words that the user wishes to substitute with a value.

13.2 Other utility programs

13.2.1 Cellgen – supercell generation

cellgen2 is a supercell generator. It is run without arguments from the command line and reads an input file called **cellgen.inp** that looks like this

```
# CdTe, zincblende
# Lattice vectors
0.5 0.5 0.0
0.5 0.0 0.5
0.0 0.5 0.5
# number of atoms
2
# Basis vectors, atomic species, coordinates, label
0.00 0.00 0.00  48  1  a
0.25 0.25 0.25  52  1  b
# Supercell generation matrix
2 0 0
0 2 0
0 0 2
# reference vector
0.0 0.0 0.0
```

This particular example takes a primitive zincblende cell and multiplies the lattice by the supercell generation matrix (or map). In this case this results in a simple doubling of the cell along the x, y and z directions, making the cell 8 times bigger. It is however possible to do more sophisticated gradual expansion by shells of the cell. For instance the following map

```
 1  1 -1
 1 -1  1
-1  1  1
```

will in this case result in an expansion of the primitive fcc lattice to its corresponding conventional cubic unit cell, which is only 4 times bigger than the primitive cell. You may also translate all atoms in the cell by some constant vector by inputting the optional "reference vector" last in the file.

13.2.2 Programs for generating strain matrices

There are a number of programs for generating strain matrices from some distortion parameter.

al_to_strain Given an angle, α, output a rhombohedral strain matrix. The angle is given in units of pi, so an input value of 0.5 gives the unit strain.

d2hstrain Given the parameter δ, d2hstrain returns the following volume conserving strain matrix

$$\begin{pmatrix} 1 & \delta & 0 \\ \delta & 1 & 0 \\ 0 & 0 & \frac{1}{1-\delta^2} \end{pmatrix} \tag{58}$$

d4hstrain Given an input parameter δ the program d4hstrain outputs the volume conserving strain matrix

$$\begin{pmatrix} 1+\delta & 0 & 0 \\ 0 & 1+\delta & 0 \\ 0 & 0 & \frac{1}{(1+\delta)^2} \end{pmatrix} \tag{59}$$

13.2.3 *smooth, gsmooth*

It is possible to convolute the output plot data from DOS, COOP, optics, etc. with a smoothing function using *smooth* or *gsmooth*. The program outputs to standard out, so run like this

```
smooth width inputfile > outputfile
```

to get a smoothed curve to **outputfile**. The program *smooth* uses a Lorentzian broadening function and *gsmooth* uses a Gaussian. The width parameter is in units of whatever is the spacing of the input file, so specifying '4' will have a half width at full maximum value of roughly 4 grid points.

13.2.4 *quadmin0*

Make a quadratic fit to x-y data in **inputfile** and output x-value of the minimum.

```
quadmin0 inputfile
```

References

[1] O. K. Andersen, Phys. Rev. B **12**, 3060 (1988), and references therein.

[2] H. L. Skriver, *The LMTO method* (Springer-Verlag, Berlin, 1984).

[3] J. M. Wills (unpublished); J. M. Wills and B. Cooper, Phys. Rev. B **36**, 389 (1987).

[4] M. Springborg and O. K. Andersen, J. Chem. Phys. **87**, 7125, (1987).

[5] M. Methfessel, Phys. Rev. B **38**, 1537 (1988).

[6] K. H. Weyrich, Phys. Rev. B **37**, 10269 (1987).

[7] S. Savrasov and D. Savrasov, Phys. Rev. B **46**, 12181 (1992).

[8] J. D. Jackson, *Classical Electrodynamics*, John Wiley and Sons, (New York, 1978).

[9] J. Korringa, Physica **13**, 392 (1947); W. Kohn and N. Rostoker, Phys. Rev. **94**, 1111 (1954).

[10] J. M. Wills, O. Eriksson, and A. M. Boring, Phys. Rev. Lett. **67**, 2215 (1991); M. Alouani, R.C. Albers, J. M. Wills, and M. Springborg, Phys. Rev. Lett. **69**, 3104, (1992), M. Alouani, J. W. Wilkins, R.C. Albers, and J. M. Wills, *ibid.*, **71**, 1415 (1993); M. Alouani and J. M. Wills, Phys. Rev. B **54**, 2480 (1996); R. Ahuja et al., *ibid.* **55**, 4999 (1996).

[11] M. Weinert, J. Math. Phys. **22**, 2433 (1980).

[12] C. Lanczos, *Applied Analysis*, Dover Publications Inc., New York, 1988.

[13] D. D. Koelling and B. N. Harmon, Journal of Physics C, **10**, 37 (1977)

[14] J. P. Perdew and A. Zunger, Phys. Rev. B, **23**, 5048 (1981)

[15] A. Grechnev *et al*, Phys. Rev. B, **76**, 035107 (2007); I. Di Marco *et al*, Phys. Rev. B **79**, 115111 (2009); P. Thunström *et al*, Phys. Rev. B **79**, 165104 (2009); O. Grånäs *et al*, Comp. Mat. Sci. **55** (2012); P. Thunström *et al*, Phys. Rev. Lett. **109** (2012).

[16] I. Di Marco, Correlation effects in the electronic structure of transition metals and their compounds, PhD Thesis, ISBN 9789090245300

[17] M. I. Katsnelson and A. I. Lichtenstein, Phys. Rev. B **61**, 8906 (2000) ; A. I. Liechtenstein *et al*, Journal of Magnetism and Magnetic Materials **67**, 65 (1987).

[18] L. Udvardi, L. Szunyogh, K. Palot'as, and P. Weinberger, Phys. Rev. B **68**, 104436 (2003).

[19] H. Ebert and S. Mankovsky, Phys. Rev. B **79**, 045209 (2009).

[20] A. Secchi, A. Lichtenstein, and M. Katsnelson, Annals of Physics **360**, 61 (2015).

[21] Y. O. Kvashnin, O. Grånäs, I. Di Marco, M. I. Katsnelson, A. I. Lichtenstein, and O. Eriksson, Phys. Rev. B **91**, 125133 (2015).

[22] Y. O. Kvashnin, A. Bergman, A. I. Lichtenstein, and M. I. Katsnelson, Phys. Rev. B **102**, 115162 (2020).

[23] I. Dzyaloshinsky, Journal of Physics and Chemistry of Solids **4**, 241 (1958) ; T. Moriya, Phys. Rev. **120**, 91 (1960).

Main index

Index of programs

Index of files

Symbols

A

B

C

D

E

F

G

H

I

J

K

L