Optimization and Parallelization of Complex DFT Codes

Dissertation

zur Erlangung des akademischen Grades Doktor der montanistischen Wissenschaften eingereicht an der

Montanuniversität Leoben

Autor:
Christian
Meisenbichler

Betreuerin:
Univ.-Prof. Dr. Dr. h.c.
Claudia Draxl

March 12, 2012

Affidavit

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume.

Christian Meisenbichler
Leoben March 12, 2012

Contents

Preface

Density-functional theory (DFT) is one of the biggest fields in todays condensed-matter and materials science. It constitutes a precise and practical way to predict and interpret many material properties. DFT is a computational method to calculate the groundstate electron density of systems like molecules and crystals.

DFT is concerned with solving the Kohn-Sham equation. The DFT toolbox contains many codes, which differ mostly by the choice of the basis set used to expand the Kohn-Sham wave functions. One of those is the LAPW (*linearized augmented plane-wave*) basis set. LAPW codes are regarded to be the approach with the least approximations. Such they are versatile and precise but usually slower and more constrained concerning the maximum system size. The **exciting** code implements this LAPW method. **exciting** sets out to provide an accessible and powerful code, optimal for new developments.

Over the lifetime of the **exciting** project, many things have changed that, today, require substantial changes of the programming paradigms. There were changes in the technical evolution of computers, the evolution of concepts in software engineering, and user interface design. These three fields, all get addressed in this work. This thesis presents seven solutions to problems connected with the **exciting** code:

1. We show how to implement a basic MPI paralellization in the exciting code.

2. We analyze different iterative solvers for the generalized eigenvalue problem.

3. We look into how to improve the *mixing* to speed up the self-consistency cycle.

4. We redesign the input file format for **exciting** by using XML.

5. We introduce XML as an output data format for **exciting**.

6. We present **exciting**@web as an interactive user interface and results database.

7. We describe the development process of **exciting**.

After a general introduction into the **exciting** code (Chapter 1) the thesis is structured in three parts: Numerical Methods (I), User Interface (II), and Software Development(III). Each of the parts has two chapters. One chapter treats background and theory and the other one the results and the implementation in the **exciting** code.

This document describes a profound transformation of the scientific software project **exciting**. The goal is to make **exciting** better suited for future scientific code development and application.

Chapter 1

Introduction to DFT and the `exciting` Code

This chapter gives an introduction to the **`exciting`** code. We start with explaining the basics of DFT. Then we describe the specifics of the LAPW basis and how it is implemented in **`exciting`**. The last section gives an overview over the profile of a typical **`exciting`** simulation.

1.1 Density-Functional Theory

Density-functional theory is a tremendously successful method in material physics. It is one approach to solve the quantum-mechanical many-body problem, yet the most successful one. This is because of its theoretical elegance, and because it allows an enormous reduction in complexity by using the groundstate density instead of the many-body wave function to formulate the groundstate solution.

The many-body Schrödinger equation,

$$\mathbf{H}\Psi(\boldsymbol{r}_1 \dots \boldsymbol{r}_n) = E\Psi(\boldsymbol{r}_1 \dots \boldsymbol{r}_n), \tag{1.1}$$

as such is practically not solvable. It is impossible to store the solution for even medium sized systems. It was for P. Hohenberg and W. Kohn in 1964 [27] to show that there mus exist a functional that maps the groundstate density to the total energy of the many-body Schrödinger equation. By this, an equation for the groundstate density could be derived by Kohn and Sham [31]. This equation for the density would give the exact density of the system at the lowest possible energetic state, the groundstate.

1.2 The Hohenberg-Kohn Theorem

Kohn and Hohenberg [27] showed that the groundstate energy of an interacting many-particle system can be expressed as a functional of the density ρ.

$$E(\rho) = T[\rho(\boldsymbol{r})] + \int V_{eff}(\boldsymbol{r})\rho(\boldsymbol{r})d\boldsymbol{r}, \tag{1.2}$$

where T is the kinetic Energy and V_{eff} is an effective potential dependent on ρ.

The minimum of $E(\rho)$ (1.2) can be found by using the variational principle, including the constraint $\int \rho(\boldsymbol{r})d\boldsymbol{r} = N$ by means of a Lagrange multiplier. We define

$$L(\rho) = E(\rho) - \lambda \int \rho(\boldsymbol{r})d\boldsymbol{r}; \tag{1.3}$$

at the minimum it holds that

$$\frac{\delta L}{\delta \rho(\boldsymbol{r})} = \frac{\delta T}{\delta \rho(\boldsymbol{r})} + V_{eff} - \lambda = 0. \tag{1.4}$$

Therefore

$$\int \left[\frac{\delta T}{\delta \rho(\boldsymbol{r})} + V_{eff} - \lambda\right] \rho(\boldsymbol{r})d\boldsymbol{r} = 0 \tag{1.5}$$

which, from its structure, resembles the Schrödinger equation. For finding a solution it is problematic that the kinetic energy is expressed as a functional of the density. We do not know an exact functional of the density for the kinetic energy. Using the local-density approximation for the kinetic energy, as it was used in Thomas-Fermi DFT, does not give good results, because this approximation is to crude

1.3 Kohn-Sham Equation

Instead of searching for the groundstate density directly, one could try to set up a system of non interacting particles that have the groundstate density of (1.5). That this is possible was shown by Kohn and Sham [31]. We search the single-particle wave functions ψ_i that are connected with the groundstate density by

$$\sum_i^N \psi_i^*(\boldsymbol{r})\psi_i(\boldsymbol{r}) = \rho(\boldsymbol{r}). \tag{1.6}$$

Then the groundstate functional can be written in therms of the kinetic energy operator and potentials

$$E = \sum_i \int_{\boldsymbol{r}} \psi_i^*(\boldsymbol{r}) \left\{ -\frac{\nabla^2}{2} + V_{ext}(\boldsymbol{r}) + V_C(\rho(\boldsymbol{r})) + V_{xc}(\rho(\boldsymbol{r})) \right\} \psi_i(\boldsymbol{r}) d\boldsymbol{r} \quad (1.7)$$

where V_{ext} is the potential from the atom cores, V_C is the potential from the charge density $\rho(r)$ retrieved from the Poisson equation, and V_{xc} is the exchange correlation potential for which a variety of approximations exist [43, 39, 38, 12]. Written like this the functional resembles a Hamilton operator. The ψ_i are called Kohn-Sham orbitals and are identified as the lowest $N/2$ eigenfunctions of

$$\left\{ -\frac{\nabla^2}{2} + V_{ext}(\boldsymbol{r}) + V_C(\rho(\boldsymbol{r})) + V_{xc}(\rho(\boldsymbol{r})) \right\} \psi_i = \epsilon_i \psi_i, \quad (1.8)$$

the Kohn-Sham equation.

1.4 Discretization

In order to search for the groundstate on a computer one has to find a finite, discrete representation of the wave function. For numerical treatment we require a discrete vector space that can be truncated in some systematic way to a finite discrete vector space. The traditional approach in physics is: Take eigen-solutions of a related system (ϕ_n) and expand the operators and wave function in this basis functions. This is known as the Ritz variational method. Therefore, ψ is expressed as linear combination of those basis functions

$$\psi = \sum_n c_n \phi_n. \quad (1.9)$$

We want to find the ground state, *i.e.*, the wave functions with the smallest expectation values:

$$\epsilon_0 = min\left(\langle\psi|\,\mathbf{H}\,|\psi\rangle\right) \quad (1.10)$$

To find the minimum we set up the Lagrangian with the constraints for $\langle\psi\,|\psi\rangle = 1$

$$\mathcal{L} = \langle\psi|\,\mathbf{H}\,|\psi\rangle - \lambda\left(1 - \langle\psi\,|\psi\rangle\right). \quad (1.11)$$

By setting the gradient of the Lagrangian to zero, we end up with the eigenvalue equation

$$\frac{\partial\mathcal{L}}{\partial c_k} = \sum_{j=1}^{N} c_j (H_{kj} - \lambda S_{kj}) = 0. \quad (1.12)$$

In general, the basis functions need not be orthogonal, therefore we have $S_{kj} = \langle \phi_k | \phi_j \rangle$ which is the *overlap matrix*. $H_{kj} = \langle \phi_k | \mathbf{H} | \phi_j \rangle$ is the *Hamiltonian matrix*. We can identify the lowest N eigenvectors of this algebraic problem as the contributions to the groundstate density. Alternatively it can be written as matrix equation:

$$\mathbf{H}\boldsymbol{x} = \lambda \mathbf{S}\boldsymbol{x} \tag{1.13}$$

with $\boldsymbol{x} = (c_1, \ldots, c_n)$, the vector with the coefficients of the basis functions. It is important to note that the Hamilton operator depends on the density. It will have to be linearized and solved iteratively (see Section 2.7).

One example for basis functions are plane waves. Plane waves are solutions for the free particle, they are the most common basis for pseudopotential calculations. Also, atomic wave functions used in the LMTO (*linear muffin-tin orbital*) or LCAO (*linear combination of atomic orbitals*) method may serve as an example. Gaussian functions are solutions to the harmonic oscillator and form the basis of well established quantum-chemistry codes.

Plane waves as basis for DFT are very popular and widely used in a variety of codes. Plane waves have some features that make them very attractive for condensed matter physics. For one, it is an orthonormal basis, and the basis functions are eigenfunctions of the kinetic energy operator, which is the quantitatively biggest part in the Hamilton operator. It implicitly supports periodic boundary conditions as a result of the discrete Fourier transformation. And the existence of the fast Fourier transformation makes the transformation from Fourier space to real space a cheap operation. Still, there is a disadvantage: The core potentials in condensed matter physics are singular at the atomic positions. They require an impractical amount of plane waves to reach a sufficient precision. Thus plane waves are used in pseudopotential codes that replace the Coulomb potentials with a smoother approximation. This approximation is in many cases not acceptable.

Some discretization schemes do not use basis functions that are derived from known quantum-mechanical solutions and embrace the tools from numerical mathematics. Wavelet multi-scale analysis using Daubechy wavelets for example, has some very striking features for DFT [19]. The wavelet basis functions are not solutions to a quantum mechanical system, rather they are suited to approximate the wave function in real *and* Fourier space and therefore allow interesting applications which are very promising but also complex to implement. Also, there are DFT codes that use real space grids [13] and finite elements [34].

1.5 APW Methods

Augmented plane wave (APW) methods merge the benefits of plane waves and atomic wave functions into a dual basis. Reference [1] provides a comprehensive introduction to the APW methods. The advantage of plane waves for periodic structures is combined with the ability of atomic wave functions to represent the wave function close to the atomic cores. This is reached by *augmenting* the plane waves, by continuing them, with a linear combination of atomic wave functions, inside a spherical domain around the cores. This domains are called *muffin-tin* (MT) spheres in analogy to the pastry forms. The atomic wave functions are spherical harmonics for the angular component, and numerical solutions to the Schrödinger equation in the full self-potential for the radial part (u_l^α). The core states having very low energies, are considered to be localized, and not interacting with the other atoms. The core states are, however, solved in the self-consistent potential, and contribute to the charge density ρ of the system. The valence states are treated in the APW basis, where the radial solutions at valence state energies are used to construct the basis in the MT. APW basis functions are not orthogonal thus lead to a generalized eigenvalue problem.

Table 1.1 gives an overview over the APW-like basis functions as they will be discussed in the sections below.

1.5.1 The APW Basis Functions

In the original APW method the basis functions are written as

$$\begin{aligned} \phi_{\boldsymbol{k}+\boldsymbol{G}}(\boldsymbol{r}) &= \frac{1}{\sqrt{\Omega}} e^{i(\boldsymbol{k}+\boldsymbol{G})\boldsymbol{r}}, & \boldsymbol{r} \in I \\ \phi_{\boldsymbol{k}+\boldsymbol{G}}(\boldsymbol{r}) &= \sum_{l,m} A_{lm}(\mathbf{k}+\mathbf{G}) u_l^\alpha(E,r) Y_{lm}(\hat{\boldsymbol{r}}), & \boldsymbol{r} \in MT. \end{aligned} \tag{1.14}$$

The A_{lm} are determined by the requirement that the values of the basis function match at the muffin tin boundaries. The radial solutions are not defined until energy level or boundary conditions are chosen. In APW the energy level for the u_l^α has to be set to the Kohn-Sham energy which makes the basis functions themselves dependent on the solution. This poses severe problems for calculating the eigenvalues in this basis.

Name	N	**Radial functions used for matching**	**Local orbitals**
APW [47]	1	$A_{lm}(\mathbf{k}+\mathbf{G})u_l(\epsilon, r)$	–
LAPW [3]	2	$A_{lm}(\mathbf{k}+\mathbf{G})u_l(E_l, r)$ + $B_{lm}(\mathbf{k}+\mathbf{G})\dot{u}_l(E_l, r)$	–
SLAPW-3 [44]	3	$A_{lm}(\mathbf{k}+\mathbf{G})u_l(E_{l1}, r)$ + $B_{lm}(\mathbf{k}+\mathbf{G})\dot{u}_l(E_{l1}, r)$ + $C_{lm}(\mathbf{k}+\mathbf{G})u_l(E_{l2}, r)$	–
SLAPW-4 [45]	4	$A_{lm}(\mathbf{k}+\mathbf{G})u_l(E_{l1}, r)$ + $B_{lm}(\mathbf{k}+\mathbf{G})\dot{u}_l(E_{l1}, r)$ + $C_{lm}(\mathbf{k}+\mathbf{G})u_l(E_{l2}, r)$ + $D_{lm}(\mathbf{k}+\mathbf{G})\dot{u}_l(E_{l2}, r)$	–
LAPW+lo [44]	2	$A_{lm}(\mathbf{k}+\mathbf{G})u_l(E_l, r)$ + $B_{lm}(\mathbf{k}+\mathbf{G})\dot{u}_l(E_l, r)$	$\tilde{A}_{lm}u_l(E_l, r)$ + $\tilde{B}_{lm}(\mathbf{k}+\mathbf{G})\dot{u}_l(E_l, r)$ + $\tilde{C}_{lm}u_l(E_{lo}, r)$
APW+lo [46]	1	$A_{lm}(\mathbf{k}+\mathbf{G})u_l(E_l, r)$	$\tilde{A}_{lm}u_l(E_l, r)$ + $\tilde{B}_{lm}(\mathbf{k}+\mathbf{G})\dot{u}_l(E_l, r)$

Table 1.1: Table of different APW-derived basis schemes. N stands for the number of matching coefficients.

1.5.2 The LAPW Basis Functions

The LAPW (*linearized augmented plane-wave*) basis is defined as:

$$\begin{aligned} \phi_{\boldsymbol{k}+\boldsymbol{G}}(\boldsymbol{r}) &= \tfrac{1}{\sqrt{\Omega}} e^{i(\boldsymbol{k}+\boldsymbol{G})\boldsymbol{r}}, && \boldsymbol{r} \in I \\ \phi_{\boldsymbol{k}+\boldsymbol{G}}(\boldsymbol{r}) &= \sum_{l,m} \left[A_{lm}(\mathbf{k}+\mathbf{G})u_l(E_l, r) + B_{lm}(\mathbf{k}+\mathbf{G})\dot{u}_l(E_l, r)\right] Y_{lm}(\boldsymbol{r}), && \boldsymbol{r} \in MT. \end{aligned} \tag{1.15}$$

Now, instead of fixing the energy of the $u_l(\boldsymbol{r})$ we kind of expand them around a linearizion energy (E_l) by adding $B_{lm}(\mathbf{k}+\mathbf{G})\dot{u}_l(E_l, r)$, where

$$\dot{u} = \frac{\partial u}{\partial E}. \tag{1.16}$$

Now the A_{lm} and B_{lm} are determined by matching the value and the first derivative at the MT sphere boundary. In LAPW, the basis functions are not any more dependent on the Kohn-Sham energy, thus the system of equation is linearized with respect to the basis functions and can be solved as a generalized eigenvalue problem.

This linearizion has one drawback. The LAPW basis is suited to describe only one state per principal quantum number. So materials with low-lying

valence states (semicore states) are not treated optimally. As an example may serve the copper 3p semicore and 4p valence states. Further extensions of the LAPW basis, like SLAPW-3 and SLAPW-4 (Table 1.1), use two linearizion energies in order to be able to treat semicore states. The additional coefficients are determined by matching the basis function to second or third order derivatives. It turns out that although they better describe semicore states, these SLAPW-3 and SLAPW-4 basis functions need a higher number of plane waves for a converged result. The radial wave functions approximate the wave function the best when their energy is close to the actual band energy. The more the radial wave function is constrained by the matching conditions, the more it deviates from the ideal form, thus one needs more basis functions to approximate the wave functions in the muffin tin spheres.

1.5.3 Local Orbitals

Local orbitals where introduced by Singh [44] to better deal with semicore states in LAPW. Local orbitals are only defined in the MT and are zero anywhere else:

$$\begin{array}{lcl} \phi^{lo}_{lm}(\boldsymbol{r}) & = & \left[\tilde{A}_{lm}u_l(E_l,\boldsymbol{r}) + \tilde{B}_{lm}\dot{u}_l(E_l,\boldsymbol{r}) + \tilde{C}_{lm}u_l(E_{lo},r)\right]Y_{lm}(\hat{\boldsymbol{r}}) \\ \phi^{lo}_{lm}(\boldsymbol{r}_a = R_\alpha) & = & 0 \\ \frac{d}{dr}\phi^{lo}_{lm}(\boldsymbol{r}_a = R_\alpha) & = & 0 \end{array} \tag{1.17}$$

The matching coefficients (A_{lm} and B_{lm}) are determined by the boundary conditions, the C_{lm} are chosen to normalize the basis function. These local orbitals are used to describe the semicore states without compromising the LAPW basis.

1.5.4 APW + Local Orbitals

As the local orbitals worked well for describing the semicore states, Sjöstedt et al. [46] pursued the idea of using local orbitals to solve the linearization of the basis functions with local orbitals too. This APW+lo basis has two distinct basis functions, the APWs

$$\begin{array}{lcll} \phi_{\boldsymbol{k}+\boldsymbol{G}}(\boldsymbol{r}) & = & \frac{1}{\sqrt{\Omega}}e^{i(\boldsymbol{k}+\boldsymbol{G})\boldsymbol{r}}, & \boldsymbol{r} \in I \\ \phi_{\boldsymbol{k}+\boldsymbol{G}}(\boldsymbol{r}) & = & \sum_{lm} A^{\alpha}_{lm}u^{\alpha}(r,E_l)Y_{lm}(\hat{\boldsymbol{r}}), & \boldsymbol{r} \in MT \end{array} \tag{1.18}$$

and the local orbitals

$$\begin{array}{lcll} \phi^{lo}_{lm}(\boldsymbol{r}) & = & \left[\tilde{A}_{lm}u_l(E_l,r) + \tilde{B}_{lm}\dot{u}_l(E_l,r)\right]Y_{lm}(\hat{\boldsymbol{r}}), & \boldsymbol{r} \in MT \\ \phi^{lo}_{lm}(\boldsymbol{r} = R_{MT}) & = & 0. & \end{array} \tag{1.19}$$

The APW basis functions use radial wave functions with fixed energy E_l. The matching at the MT boundary is only in the value of the wave function. This basis turned out to need the least number of basis functions among all APW-derived methods to reach sufficient accuracy, and it can describe semicore states. Therefore it is the default choice today.

1.6 The `exciting` Code

The **`exciting`** code [2] is a density-functional theory and excited-states package based on the linearized augmented plane-wave method. It can be applied to all kinds of materials, irrespective of the atomic species involved, and also allows for the investigation of the core region. **`exciting`** is an open source code. The code particularly focuses on excited-state properties, within the framework of *time-dependent DFT* (TDDFT) as well as within *many-body perturbation theory* (MBPT). **`exciting`** is written in FORTRAN and is distributed as source code. Users must compile the code in order to use it on their platform. The **`exciting`** code can be obtained from http://exciting-code.org and a public source-code repository.

The package consists of the **`exciting`** program, the `spacegroup` tool, a set of example systems, documentation, and a set of templates for data processing. **`exciting`** is a command-line tool which reads one input file that contains the information about the structure to be calculated as well as the properties which are desired. The atomic species used in the calculation are described by species-files that contain the information necessary to define the basis functions in the muffin-thin sphere. Also, information like mass and charge is stored, and it is specified which of the electrons are treaded as core electrons. It is the place where one configures whether LAPW or APW+lo, or any other basis set is used. The **`exciting`** code comes with species files for all elements, and provides 3 versions of each. There is APW+lo, the default, one LAPW and one LAPW+lo species file. If needed, any other scheme as described in Table 1.1 can be configured manually.

The code was developed in the framework of the EU Research and Training Network **`exciting`** [17], with the aim to provide a highly precise, and at the same time, developer-friendly, DFT package.

1.7 Profiling

The first step in the optimization of a computer code is to measure computation times. One way is to use the timing capabilities of the operating

system to put out timing information manually. This is done with calls like `cpu_time(timer)`. In order to discover hot-spots, that are parts of the code where most time is spent, a more fine-grained information is needed. The procedure to record timing information during a computation is called profiling. The tools that are used to collect the data are profilers. There are various different methods to get such timing information. It is possible to instrument the code at compile time with instructions that write a log whenever a subroutine is called. The next possibility is to sample the code at a certain frequency to look which function is on top on the stack, and therefore is executed in that moment. These two approaches make different compromises. Sampling can leave small but very frequently called functions underrepresented, and the log has a time overhead leading to great deviations of the timing information. Only recent developments in operating systems brought forward a new approach which allows very accurate time measurements at almost no overhead, even including timing of library calls. This is done by a dedicated operating-system service which allows to instrument the code at runtime and record timing with almost no overhead. Examples are the *trace facility* by IBM (tprof) or *DTrace* by SUN.

The data presented here was gained with `gprof` on an IBM Power 5 cluster with xlf90 compilers and ESSL numerical libraries. Tables 1.2 and 1.3 assert the state as it was before any changes. The material used for the benchmarks was *polyacetylene* (PA).

1.8 Profiling of the `exciting` Code

The first profile (Table: 1.2) was made without using compiler optimizations. The table shows the time spent inside various procedures, sorted by the most consuming procedures at the top. So the top-most procedure is first to be analyzed. The times spent in the **`exciting`** code, and in the mathematical library functions are listed separately. The table tells that 77.36% of computation is spent in the **`exciting`** binary which is very irritating. The expectation would rather be that the most expensive computation is to deal with the large matrices of the eigensystem and that this would mostly happen in the solver routine from LAPACK. Instead we find the `zmatinp.f90` routine to lead with 70.32%. This routine is used in the matrix setup and is responsible for rank 2 vector updates to the Hamiltonian and the overlap matrix. This is very unexpected. Fortunately, it is not so serious, because the measurements where done without compiler optimizations, such that `zmatinp.f90` is quite inefficient and does not use the advanced processor features. Repeating the profile with optimization changes the impression quite a bit.

Program part	Ticks	%	File name
exciting binary	863271	77.36%	
.zmatinp	784828	70.32%	zmatinp.f90
.hmlalo	10560	0.96%	hmlalo.f90
.match	8477	0.76%	match.f90
.spline	7088	0.64%	spline.f90
Library (essl)	247411	22.16%	
.zeumvb	113466	10.16%	sl/src/work/zeumvb.f
.zhur2b	73572	6.6%	sl/src/work/zhur2b.f
.zuctsb	33531	3%	sl/src/work/zuctsb.f
.zacbp4	16885	1.52%	sl/src/work/zacbp4.f

Table 1.2: Profile of original **exciting** 0.9.151.

Program part	Ticks	%	File name
exciting	105543	31.92%	
zmatinp	88731	26.84%	zmatinp.f90
.match	2009	0.6%	match.f90
.gradzfmt	1340	0.4%	gradzfmt.f90
.spline	1227	0.36%	spline.f90
.zpotcoul	1036	0.32%	zpotcoul.f90
Library (essl)	224435	67.88%	
.zeumvb	102936	31.12%	sl/src/work/zeumvb.f
.zhur2b	66912	20.24%	sl/src/work/zhur2b.f
.zuctsb	30467	9.2%	sl/src/work/zuctsb.f
.zacbp4	15232	4.6%	sl/src/work/zacbp4.f

Table 1.3: Profile of original **exciting** 0.9.151 compiled with -O3 for the same example as Table 1.2

Program part	Ticks	%	File name
exciting binary	92626	19.72%	
.hmlalon	9642	2.04%	./../src/hmlalon.f90
.match	8678	1.84%	../../src/match.f90
Hermiteanmatrix _indexedupdate	7830	1.68%	/src/modfvsystem.f90
.cmf4kb	7611	1.6%	cfftnd.f90
.spline	7061	1.52%	../../src/spline.f90
.zpotcoul	6787	1.44%	/../src/zpotcoul.f90
.gradzfmt	5949	1.28%	/../src/gradzfmt.f90
.hmlistln	5157	1.08%	/../src/hmlistln.f90
.cmf3kb	4534	0.96%	cfftnd.f90
Library (essl)	369746	78.64%	
.zhur2b	185269	39.4%	sl/src/work/zhur2b.f
.zeumvb	112991	24.04%	sl/src/work/zeumvb.f
.zuctsb	63890	13.6%	sl/src/work/zuctsb.f

Table 1.4: Profile of modified **exciting** 0.9.151. The `zhur2b.f` is the library function in ESSL that does the computation for the `zher2` call.

Table 1.3 shows the same code with optimization flags turned on. The fraction of zmatinp.f90 immediately drops to 26%. We note that the matrix setup is very important but now other features show up. The time spent in the numerical library is raised to 67%, the dominant part of which now is the eigen-solver. The functions listed are not familiar names because they are proprietary implementations of LAPACK functionality by IBM.

The `zmatinp.f90` function is almost the same as the LAPACK procedure `ZHER2` for hermitian matrices except the sign of the input vectors. So it is easily replaced. The result is shown in Table 1.4 and 1.5. Remarkably, the rank-two update procedure (`zhur2b.f`) is now dominant in the profile. Not all of the calls come from the matrix update but it is clear that this kind of operation is an outstanding hot-spot which must be addressed. The problem with this operation is that it has to update every entry in the matrix for every call which uses enormous memory bandwidth. This dominance could not be explained with floating point operations. On the other side, a parallel implementation of this update, is straight forward and can be expected to scale optimally. We leave this for discussion at a later point.

The conclusion from this example is that the eigen-solver needs indeed most of the computing resources, here about 50%, but the matrix setup is also a significant part. We learn that any optimization strategy must address the setup with the same priority as the diagonalization. The matrix setup scales

exciting	32618	8.88%	
.cmf4kb	6738	1.84%	cfftnd.f90
.cmf3kb	4028	1.08%	cfftnd.f90
iteanmatrix_indexedupdate	2532	0.68%	/src/modfvsystem.f90
.match	1801	0.48%	../../src/match.f90
.spline	1296	0.36%	../../src/spline.f90
Library (essl)	334242	90.96%	
.zhur2b	166785	45.4%	sl/src/work/zhur2b.f
.zeumvb	102055	27.76%	sl/src/work/zeumvb.f
.zuctsb	58553	15.92%	sl/src/work/zuctsb.f
.zmv6b	1139	0.32%	ssl/src/work/zmv6b.f
.zuntsb	1109	0.32%	sl/src/work/zuntsb.f

Table 1.5: Profile of modified **`exciting`** 0.9.151 compiled with -O3.

with the number of basis functions squared times the number of atoms. The diagonalization takes typically longer than the setup and scales with n^3 with the number of basis functions, but is much more difficult to do in parallel.

Part I

Numerical Methods

Chapter 2

Algorithms

In the first chapter, we identified two main factors that contribute to the computation time. One is the setup, and the other one the diagonalization of the eigenvalue problem. An additional factor is the mixing algorithm that has a large influence on how many iterations are needed to reach a converged result. This chapter gives a review of some algorithms that are suited to improve the performance of the **exciting** code.

2.1 Scaling of Algorithms

The scaling is the relation between the number of operations and the size of the problem to solve (n). Analysis of the scaling behavior is sometimes also called complexity analysis. The computational complexity, or scaling, describes the behavior of algorithms when the problem size changes. Examples are logarithmic ($\log n$), linear (n), quasi linear ($n \log n$), quadratical (n^2), or even exponential (a^n) scaling. Different parts of a code may have different scaling behaviors when the system size changes. This results in previously small parts of the computation becoming dominant when they scale worse.

2.2 Eigensolvers

The solution of the Kohn-Sham equation involves the eigenvalue problem to find the eigenvectors for the n lowest eigenvalues, n being the number of occupied states in the system. The LAPW or APW+lo basis sets are not orthogonal, which makes the problem a generalized eigenvector problem. To solve such a system, there are highly accurate and very efficient algorithms available (LAPACK[4]), yet they have n^3 scaling, and are difficult to perform

on parallel architectures. There is a library to perform large scale-linear algebra computations on massively parallel systems, which is ScaLAPACK[8].

Unfortunately, replacing the LAPACK eigen-solver with its ScaLAPACK equivalent, doesn't quite solve the problem. The challenge is rather, to exploit specific features of the LAPW calculation, such as the fact that only the lowest 10-15% of the eigenvalues are needed, and that the self-consistency procedure involves the repeated solution of very similar problems.

A closer look at the scaling problem gives a clear scenario. The technological progress makes memory and computation speed grow roughly with the same exponential rate (Moores law). The n^3 scaling of the direct diagonalization algorithm and the n^2 scaling in memory consumption, result in the tendency that systems that fit into the available memory take longer and longer to run. This is the practical experience in today's research. If the scaling cannot be reduced from being of order three, algorithms that may be inferior in speed today but scale better, will overtake in the near future.

Any modern computer has multiple processing units, and the way to reach the edge of high-performance computing today, is to connect multiple systems in a low-latency network to a cluster. Whoever develops high-performance computing code must think in these terms. It is a requirement to have better scaling and better parallelization, even if it has a penalty at small and medium system sizes.

Iterative solvers can get the computational complexity, at least in parts, down to n^2 if only a small number of eigenvectors is required. Strictly this is only true if the number of required eigenvalues is constant and the number of iterations needed for the algorithm to converge is also constant because one iteration has already order n^2 operations

Most iterative algorithms find a range of eigenvectors for themselves, others are designed to refine a set of test-vectors with each iteration. Solving the Kohn-Sham equation involves repeatedly solving the eigensystem in order to reach self-consistency. If the algorithm requires a set of test eigenvectors, and the sequential systems are similar enough, one could use the solutions of the previous iteration as the new trial vectors and reduce the number of iterations. Wood and Zunger [55] proposed such a refinement method for the SCF loop.

2.3 Iterative Solvers

Iterative methods are all methods that search the eigenvectors by iteratively refining approximate eigenvectors until a convergence goal is met. In contrast, direct solvers need a defined number of operations to come up with the

solution. Iterative solvers converge towards the solution and take as many steps as necessary. If the convergence can be optimized by choosing the right algorithm, iterative solvers can be faster than direct solvers, especially on parallel systems. This is because the required linear algebra operations for iterative solvers perform better on parallel systems.

The basic principle is best illustrated by the *power method*. When a matrix is successively multiplied to a vector, and the vector is normalized in each step, the vector converges to the eigenvector with the largest eigenvalue. This is because the largest eigenvalue gets dominant rapidly when exponentiated. This can be easily seen if one writes down the operator multiplication with the vector, where the vector is written as linear combination of eigenvectors, $\boldsymbol{v} = \sum_i c_i \boldsymbol{e}_i$,

$$\mathbf{A} \dots (\mathbf{A}(\mathbf{A}\boldsymbol{v}) = (\lambda_1)^n c_1 \boldsymbol{e}_1 + (\lambda_2)^n c_2 \boldsymbol{e}_2 + [\dots] \,. \tag{2.1}$$

With higher n the the highest eigenvalue becomes so dominant that the iterated vector converges to the corresponding eigenvector. The Lanczos algorithm uses the sequence of iterated vectors and orthogonalizes them to span the Krylov subspace. In this basis, the matrix $\mathbf{A}$ is approximated as a tridiagonal matrix that can be expected to have eigenvalues that are a good approximation tho the eigenvalues of $\mathbf{A}$. This way, one can efficiently find reasonable approximations for a few eigenvalues close to the maximum eigenvalue. Extensive theory about iterative solvers can be found in Reference [42].

2.3.1 Shift-Invert Lanczos

The Lanczos algorithm is well suited if few maximal eigenvalues are searched. In LAPW, we search the groundstate which is on the other end of the spectrum. The Lanczos algorithm can find the minimal eigenvalues too but it converges more slowly.

An elegant solution is to perform the iteration with the shift-inverted matrix. The shift-invert iteration finds the eigenvalues of the operator

$$(\mathbf{A} - \sigma\mathbf{I})^{-1}. \tag{2.2}$$

This iteration converges very fast to the eigenvalues around the inversion center σ, given that the equation

$$(\mathbf{A} - \sigma\mathbf{I})x = b \tag{2.3}$$

has a solution and can be computed. For hermitian matrices this is of course the case.

For the generalized eigenvalue problem there are two options when we want to use a Lanczos-like algorithm. If the overlap matrix $\mathbf{B}$ is positive definite (which it is in LAPW), we can apply a Cholesky decomposition [21] of $\mathbf{B}$ to transform the generalized system into a standard one, and then apply the standard algorithm. The other option is doing the shift-invert procedure with the operator

$$(\mathbf{A} - \sigma\mathbf{B})^{-1}. \tag{2.4}$$

This second option allows to find also the eigenvalues closest to σ. The shift-invert operator is implemented by doing a LU-decomposition beforehand, and applying it by back substitution in each iteration. It sounds inefficient to solve a linear system in each iteration, but, in fact, it is always the same system with different right sides. Back-substitution is both fast, and can rely on parallel implementations. It scales with n^2 just as a matrix multiplication would.

In LAPW, we find the situation that the shift-invert procedure is advantageous if the fraction of required eigenvalues is small. Then it can be much faster than the direct solver with the additional advantage of using operations that can use parallel libraries. This fraction, however, depends on the material and whether one needs unoccupied states. Above a certain fraction of required eigenvectors, the efficiency is worse than with the direct solver, although the advantage of parallel efficiency remains. The ARPACK library [33] provides production-ready implementation of the Arnoldy iteration [5] which is an extension of the Lanczos process for normal or generalized eigenvalue problems including the shift-invert procedure.

2.4 Refinement Methods

There are a number of methods that try to refine an approximate eigenvector. The basic idea is to approximate a *Newton update* or *correction equation* (see [55] or [40])

$$|\delta\boldsymbol{a}\rangle = -(\mathbf{H} - \lambda^{ap}\mathbf{S})^{-1} \left|\boldsymbol{r}(|\boldsymbol{a}^{ap}\rangle, \lambda^{ap})\right\rangle. \tag{2.5}$$

The correction equation uses the residual $\boldsymbol{r}$ to get an update $\delta\boldsymbol{a}$ for the approximate eigenvector $\boldsymbol{a}^{ap}$. The residual is given by

$$\left|\boldsymbol{r}\left(|\boldsymbol{a}^{ap}\rangle, \lambda^{ap}\right)\right\rangle = (\mathbf{H} - \lambda^{ap}\mathbf{S}) \left|\boldsymbol{a}^{ap}\right\rangle. \tag{2.6}$$

The approximate eigenvalue λ^{ap} is calculated by

$$\lambda^{ap} = \frac{\langle\boldsymbol{a}^{ap}| \mathbf{H} |\boldsymbol{a}^{ap}\rangle}{\langle\boldsymbol{a}^{ap}| \mathbf{S} |\boldsymbol{a}^{ap}\rangle}. \tag{2.7}$$

As stated above, an approximation of the correction equation must be found because solving it as a system of linear equations is computationally too expensive.

2.4.1 Diagonal Approximation of the Correction Equation

In planewave-based DFT methods the Hamiltonian matrix is diagonally dominant. This can be explained from the fact that the kinetic energy operator in a plane wave basis is a diagonal matrix and it is quantitatively the largest part. The correction equation (2.5) may be approximated by using only the inverse of the diagonal elements as it was originally proposed by Davidson [16] for the Block-Davidson algorithm. In planewave codes, this approach was sufficient and fast, but the more complex APW-like basis sets don't allow for this simplification.

2.4.2 Preconditioned Diagonal Approximation

An extension to the idea of the diagonal approximation of the correction equation is to find a preconditioning matrix to make the Hamiltonian more diagonal dominant, get a new update in the diagonal approximation, and transform the results back. The eigenvectors $\boldsymbol{x}_i$ of the Hamiltonian would make $\mathbf{H} - \lambda^{ap}\mathbf{S}$ diagonal:

$$|\delta \boldsymbol{a}\rangle = -\sum_i (\langle \boldsymbol{x}_i | \mathbf{H} - \lambda^{ap}\mathbf{S} | \boldsymbol{x}_i \rangle)^{-1} \langle \boldsymbol{x}_i | \boldsymbol{r} \rangle | \boldsymbol{x}_i \rangle \tag{2.8}$$

$$= -\sum_i (\frac{1}{\lambda_i - \lambda^{ap}}) \langle \boldsymbol{x}_i | \boldsymbol{r} \rangle | \boldsymbol{x}_i \rangle . \tag{2.9}$$

In DFT this can actually lead to a useful algorithm [40]. In the SCF loop, the successive eigensystems can be assumed to be similar enough such that the eigenvectors from a previous system can serve as a preconditioner for a couple of SCF iterations. Such updates from the preconditioned correction equation do not converge well enough for themselves. In order to get a better convergence one needs convergence enhancing algorithms such as RMS-DIIS.

2.4.3 RMS-DIIS Method

The *residual minimization by direct inversion in iterative subspace* (RMS-DIIS) is a method to search for the solution with the minimum error in the

subspace spanned by successive iterations of approximate eigenvectors. RMS-DIIS tries to minimize the error by minimizing the residual vector. The RMS-DIIS method assumes that one can find a linear combination of approximate solutions which has a smaller error, and thus is a refined solution. Each of our trial solutions $\boldsymbol{a}_{ap}$ can be written as the exact solution plus an error term

$$\boldsymbol{a}_{ap} = \boldsymbol{a} + \boldsymbol{e}. \tag{2.10}$$

Now we expect that there is a linear combination of a set of approximate solutions that has a smaller error than each of them.

$$\boldsymbol{a}_{ap} \quad = \quad \sum_{i=1}^{m} c_i \boldsymbol{a} + \sum_{i=1}^{m} c_i \boldsymbol{e}_i \tag{2.11}$$

We wish the second term to be minimal, with the requirement for the coefficients c_i

$$\sum_{i=1}^{m} c_i = 1. \tag{2.12}$$

Of course we don't know the error but we have a related indicator, the residual vector (2.6) which is required to be zero for any exact eigenvector. The assumption is that the linear combination of previous residuals with the smallest norm delivers the coefficients for a better approximation of the eigenvector. So we search for the minimum of the residual norm

$$min(\langle \boldsymbol{r} \,|\boldsymbol{r}\rangle) = min(\sum_{ij}^{m} c_i^* c_j \,\langle \boldsymbol{r}_i \,|\boldsymbol{r}_j\rangle) \tag{2.13}$$

The Lagrangian to minimize the residual norm under the constraints (2.12) is

$$\mathcal{L} = \langle \boldsymbol{r} \,|\boldsymbol{r}\rangle - \lambda \left(1 - \sum_{i=1}^{m} c_i\right). \tag{2.14}$$

$$\frac{\partial \mathcal{L}}{\partial c_k} = 0. \tag{2.15}$$

The partial derivations of the Lagrangian set up a linear system for $\boldsymbol{c}$

$$\begin{pmatrix} P_{11} & P_{12} & P_{13} & \dots & P_{1n} & -1 \\ P_{21} & P_{22} & P_{23} & \dots & P_{2n} & -1 \\ P_{31} & P_{32} & P_{33} & \dots & P_{3n} & -1 \\ \dots & \dots & \dots & \dots & \dots & -1 \\ P_{n1} & P_{n2} & P_{n3} & \dots & P_{nn} & -1 \\ -1 & -1 & -1 & \dots & -1 & 0 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \dots \\ c_n \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \dots \\ 0 \\ 1 \end{pmatrix} \tag{2.16}$$

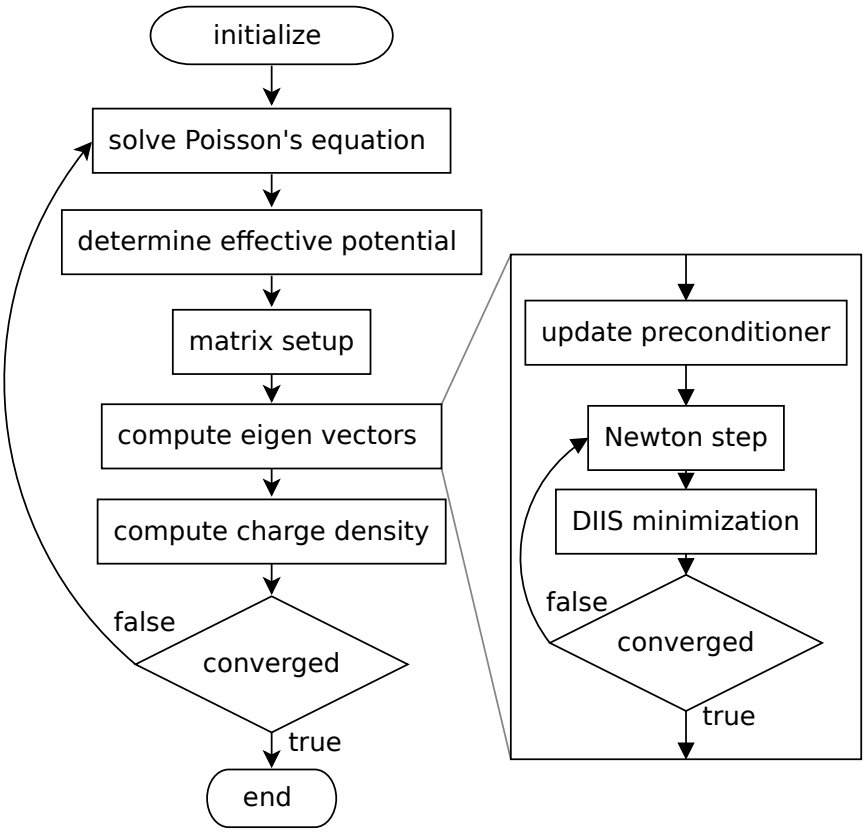

Figure 2.1: SCF loop (left) and iterative diagonalization (right).

$\mathbf{P}$ is defined as

$$P_{ij} = \langle \boldsymbol{r}_i \, | \boldsymbol{r}_j \rangle \, . \tag{2.17}$$

The new vector is given by

$$\boldsymbol{a}_{m+1} = \sum_{i=1}^{m} c_i \boldsymbol{a}_i . \tag{2.18}$$

This DIIS variant is also known as Pulay mixing. It performs the minimization as an extremal value problem with the constraints 2.12. This is not the way it is used in DFT like by Wood and Zunger [55] or Rayson [40]. They would rather solve for the minimal residual by finding the lowest eigenvalue in the iterative subspace. With $\mathbf{P}$ from (2.17) and $Q_{ij} = \langle \boldsymbol{a}_i \, | \boldsymbol{a}_j \rangle$:

$$\mathbf{P} \, | \boldsymbol{c} \rangle = \rho^2 \mathbf{Q} \, | \boldsymbol{c} \rangle \, . \tag{2.19}$$

In this case the constraint for $\boldsymbol{c}$ is

$$\sum_{ij}^{m} c_i^* c_j \langle \boldsymbol{a}_i | \boldsymbol{a}_j \rangle = 1. \tag{2.20}$$

This is used in some DFT codes today. As an example for DIIS implemented for a dense matrix eigensystem may serve the solver from the Gaussian code [40]. Figure 2.1 shows how the RMS-DIIS iteration is a part of the SCF iteration. The preconditioned Davidson update from (2.9) is used to get new approximate eigenvectors. The iterative subspace of the vectors yielded by the Davidson iteration is then used in the RMS-DIIS until the residual norm meets the convergence criterion.

2.5 How to Make Parallel Code

Today's developments in computer technology demand high-performance software to be parallel in one way or the other. Higher integration density on the chips no longer leads to much improvement in scalar performance, but allows the integration of multiple execution units on one chip. This has to be considered at every stage of developing algorithms. Here follows a review of the possible levels of concurrency and their challenges.

2.5.1 Task Parallelism

A calculation may be split up into independent parts on a high level, for example, repeated calls to a subroutine that calculates partial results to be summed up in the end. This task parallelism is often rather easy to realize because the coupling of the independent parts is so small. This kind of parallel execution is well suited to be distributed over nodes in a cluster. Each node may perform the same algorithm on different data in parallel, instead of computing one data set at a time. The main aspect that influences the efficiency of that approach is load balancing. If the processes do not have the same amount of work, all the processes must wait until the processes with the highest work load has finished. As each of the processes does the same job as one process did sequentially before, the amount of used memory multiplies with the number of processes. If the memory consumption is the limiting factor, this approach is not applicable.

2.5.2 Data Parallelism

Data parallelism refers to computation with distributed data structures, like doing linear algebra on distributed arrays. There are two ways to reach that: *symmetric multi processing* (SMP), and implementing array operations with message passing. The latter is provided by the BLACS, PBLAS, and ScaLAPACK libraries.

2.5.3 Symmetric Multi Processing

Symmetric Multi Processing (SMP) is about programming with threads. Threads are the parallel execution paths one multi-threaded process can have. Threads have a shared memory space but can also allocate their own memory. The ability for using threads has to be provided by the operating system. The programmer can use the POSIX thread API (*aplication programmer interface*) and care for all the locks and synchronization issues manually, or use OpenMP (*open multi-processing*) directives. OpenMP directives are useful for parallelizing loops, but it is difficult to get efficient code, except for very high level parallelism. There are, however, multi-threaded BLAS and LAPACK libraries on many platforms. In these libraries, not every subroutine exists in a multi-threaded version. This is because the parallel execution gives no benefit if the memory access pattern is badly suited for parallel execution. So the key to benefit from these multi-threaded libraries is to only those library subroutines that actually are efficient with multiple threads. The great advantage of using multi-threaded libraries is, that using them requires no changes in the source code. The limitation of this approach is, that the system has to fit into the main memory.

2.6 Linear Scaling

The most desirable scaling behavior is linear or almost linear scaling. This, however, implies that it is possible to store the necessary operators and data in structures that themselves scale linearly or almost linearly with problem size. The term "almost linear" scaling here means linear times a logarithmic factor. Examples for such structures are sparse matrices that originate from finite-element discretization. Unfortunately, the LAPW discretization generates no sparsity at all.

A second approach to linearly scaling storage is the concept of *hierarchical matrices* (H-matrixes) [22], which exploits a feature referred to as: "data-sparsity". The method is primarily applied in integral equations with a strongly decaying kernel function. If one assumes a localized basis set, where

the magnitude of the matrix elements depends primarily on the distance of the basis functions, then it is possible to cluster (reorder) the basis-functions based on the geometry information. This is done in such a way, that one looks for blocks in the matrix that can be approximated with low-rank matrix products, *e.g.* , the matrix product of two vectors. Unfortunately, we could not find any strategy to apply the concept of H-matrices in LAPW codes.

2.7 Fixed-Point Iteration

The computation of the density within the framework of DFT requires an iterative convergence procedure to reach self consistency. This is necessary, because the Kohn-Sham equation is a nonlinear equation. The potential depends on the density via the Poisson equation. The nonlinear equation is linearized by defining the potential as constant and ignoring the dependence on the wave function for a while. After the Kohn-Sham equation is solved for such a trial potential one has a new charge density which gives a new potential. If the input density was the exact solution, the new density should be the same as the input density. So, when the subsequent densities differ less then a certain ϵ, the iteration has reached self consistency.

The number of steps required for this convergence multiplies the total computation time. Finding algorithms that optimize this convergence to be fast and stable is therefore important. The problem is an optimization with many variables, where the error must be minimized with the fewest steps possible. In DFT this procedure is called *mixing* because the simplest algorithm that reaches a stable convergence is to use a linear combination of the new and the old density, thus a mix of the two, for the new input density. Just using the new density as the input for the next iteration is not stable and may lead to uncontrolled oscillations.

2.8 Newton Methods

Let us define $\boldsymbol{f}(\boldsymbol{\rho}_k)$ as the function that computes the new density from the old density.

$$\boldsymbol{\rho}_{k+1} = \boldsymbol{f}(\boldsymbol{\rho}_k), \tag{2.21}$$

The perfectly converged solution satisfies

$$\boldsymbol{\rho}_k = \boldsymbol{f}(\boldsymbol{\rho}_k), \tag{2.22}$$

therefore we search the vector $\boldsymbol{\rho}$ where the residual (R) becomes zero.

$$\boldsymbol{r}(\boldsymbol{\rho}_k) = \boldsymbol{f}(\boldsymbol{\rho}_k) - \boldsymbol{\rho}_k = 0 \tag{2.23}$$

For such problems the Newton method is known to converge very fast. Newtons method denotes as

$$\boldsymbol{\rho}_{k+1} = \boldsymbol{\rho}_k - \left[\mathcal{J}\{\boldsymbol{r}(\boldsymbol{\rho}_k)\}\right]^{-1} \boldsymbol{r}(\boldsymbol{\rho}_k). \tag{2.24}$$

$\mathcal{J}$ is the Jacobian matrix of $\boldsymbol{r}(\boldsymbol{\rho})$, which leaves us with the problem of calculating the Jacobian and solving the linear system. In the case of the SCF cycle, we do not have a practical way to calculate the partial derivatives for the $\boldsymbol{\rho}$-vector coefficients. So the challenge is to find reasonable approximations to the mapping of the Jacobian.

2.9 Broyden Methods

The Broyden methods [11] are a class of methods that use the secant equation to approximate the Jacobian. It is in a way a generalization of the secant method for finding roots in one dimension where the secant is the line trough two function values. We want to find a matrix $\mathbf{B}_n \approx \mathcal{J}\{\boldsymbol{r}(\boldsymbol{\rho}_k)\}$ that satisfies the secant equation

$$\mathbf{B}_n \boldsymbol{r}(\boldsymbol{\rho}_n) = \boldsymbol{r}(\boldsymbol{\rho}_n) - \boldsymbol{r}(\boldsymbol{\rho}_{n-1}). \tag{2.25}$$

From this alone $\mathbf{B}_n$ is underdefined and it is subject to the different Broyden methods to construct this matrix $\mathbf{B}_n$, or rather its inverse, because a new update for $\boldsymbol{\rho}$ denotes as

$$\boldsymbol{\rho}_{n+1} = \boldsymbol{\rho}_n - \mathbf{B}_n^{-1} \boldsymbol{r}(\boldsymbol{\rho}_{n-1}). \tag{2.26}$$

In Ref. [11] Broyden gives 3 methods how to construct updates to the matrix $\mathbf{H_n} = \mathbf{B}_n^{-1}$ from previous $\boldsymbol{r}(\boldsymbol{\rho})$.

2.10 Multi-Secant Broyden Method

The *multi-secant Broyden* (MSEC Broyden) [35] method doesn't just use two sequential iterates to approximate the Newton step, but uses more of the history. In MSEC Broyden, $\mathbf{H}_n$ should satisfy all secant equations from a number of previous iterates k. Marks and Luke [35] developed the algorithm for use in an LAPW code. Particularly, they added regularization and preconditioning. The MSEC Broyden algorithms according to Marks and Luke decompose as

$$\boldsymbol{\rho}_{n+1} = \boldsymbol{\rho}_n - \mathbf{H}_0 \left(\boldsymbol{r}_n - \mathbf{Y}_{n-1}\mathbf{A}_n\boldsymbol{r}_n\right) - \mathbf{S}_{n-1}\mathbf{A}_n\boldsymbol{r}_n. \tag{2.27}$$

Here, $\mathbf{A}_n$ is a matrix dependent on the method, $\mathbf{H}_0$ is an initial estimate for the Jacobian, $\mathbf{Y}_{n-1}$ is a matrix with the last k residual changes, and $\mathbf{S}_{n-1}$ is a matrix with the last k step directions. As there is no better approximation for the initial inverse Jacobian, one uses $\mathbf{H}_0 = \sigma_n \mathbf{I}$ where σ_n is a dynamic step length. $\mathbf{A}_n$ is the approximated Jacobian derived from the secant equations.

For LAPW, Marks and Luke found in numerical experiments that convergence can be improved by rescaling the coefficients for the interstitial relative to those for the muffin tin region. This rescaling is applied by $\mathbf{\Omega}_n$, a diagonal matrix containing the scaling factors. For multi secant Broyden Type 1 (MSB1) $\mathbf{A}_n$ denotes as

$$\mathbf{A}_n = \mathbf{\Psi}_n \left(\mathbf{\Psi}_n \hat{\mathbf{S}}_{n-1}^T \hat{\mathbf{Y}}_{n-1} \mathbf{\Psi}_n + \alpha \mathbf{I} \right)^{-1} \mathbf{\Psi}_n \hat{\mathbf{S}}_{n-1}^T \mathbf{\Omega}_n, \tag{2.28}$$

with $\hat{\mathbf{Y}}_n = \mathbf{\Omega}_n \mathbf{Y}_n$ and $\hat{\mathbf{S}}_n = \mathbf{\Omega}_n \mathbf{S}_n$. $\mathbf{\Psi}_n$ is a diagonal matrix that renormalizes $\mathbf{Y}_n$, and α is a parameter for the *regularization* according to Ref. [26].

Chapter 3

Performance Optimizations

The profiling of the **exciting** program in section 1.7 revealed the time consuming parts of the code. There are two parts that consume most of the computation time: The matrix setup and the diagonalization. To improve the performance we identify 3 strategies. First, implementing k-point paralellization; second, a better paralellizable eigen-solver; and third, an improved mixing algorithm to reduce the number of necessary iterations.

3.1 Implementation of k-point Parallelism

The Bloch wave functions have to be calculated on a grid in k-space. These are independent calculations until the partial densities have to be summed to get the total density. For each SCF cycle, the parallel processes can independently compute the wave functions and need only to exchange the density at the end of each iteration. This can be implemented with the Message Passing Interface (MPI) standard. MPI allows communication between separate processes. These processes can be distributed among a set of computers, a cluster of compute nodes.

In order to illustrate how the program flow in **exciting** with k-point parallelization works we want to distinguish 4 different modes of the program flow:

A *Parallel execution on same data.* This has no benefit performance-wise but sometimes it is simpler to have the same thing executed on all processes than to exchange the data over the network.

B *Parallel execution on different data.* This is where the actual parallel calculation takes place.

C *Execution on one node, others waiting.* Sometimes there are parts that cannot be easily parallelized and that are better executed on one node because multiple nodes could, for example, interfere with each other during file access.

D *Communication.* This is when the processes exchange messages via MPI.

The enumeration below enlists the steps that are relevant in the k-point parallel execution. At the end of each point, we denote the execution mode as described above.

1. All processes initialize on the same data. (**A**)
2. The eigensystem is solved for different k-point ranges in parallel. (**B**)
3. The eigenvalues have to be exchanged in order to compute the occupation numbers. This is done in `mpisyncevalsvspnchr.F90`. (**D**)
4. The occupation numbers are computed on one process and then are distributed to the others. (**C**)
5. The partial charge densities are computed parallel on different k-ranges. (**B**)
6. The partial densities are summed in procedure `mpisumrhoandmag`. (**D**)
7. Potential calculation on all nodes. (**A**)
8. If not converged restart from 2.
9. After leaving the SCF loop, the program continues in mode (**C**), meaning that only the master process computes and the others are waiting.

Figure 3.1 shows the diagram of the dependencies of the different parts in the groundstate calculation.

When the part of the algorithm that can be calculated in parallel, is the most time consuming one, the MPI parallelization is a big benefit. This is true for a certain class of materials. The range of materials for which it is a useful parallelization strategy, starts from materials that have enough atoms, such that the time spent for diagonalization is dominant over the rest of the computation, and it ends with the materials that have so many atoms that one does not require more than one k-point. Figure 3.2 shows the scaling behavior for polyacetylene, an organic crystal that falls well between the boundaries mentioned above. The figure illustrates that the speedup is very good up to 16 processes.

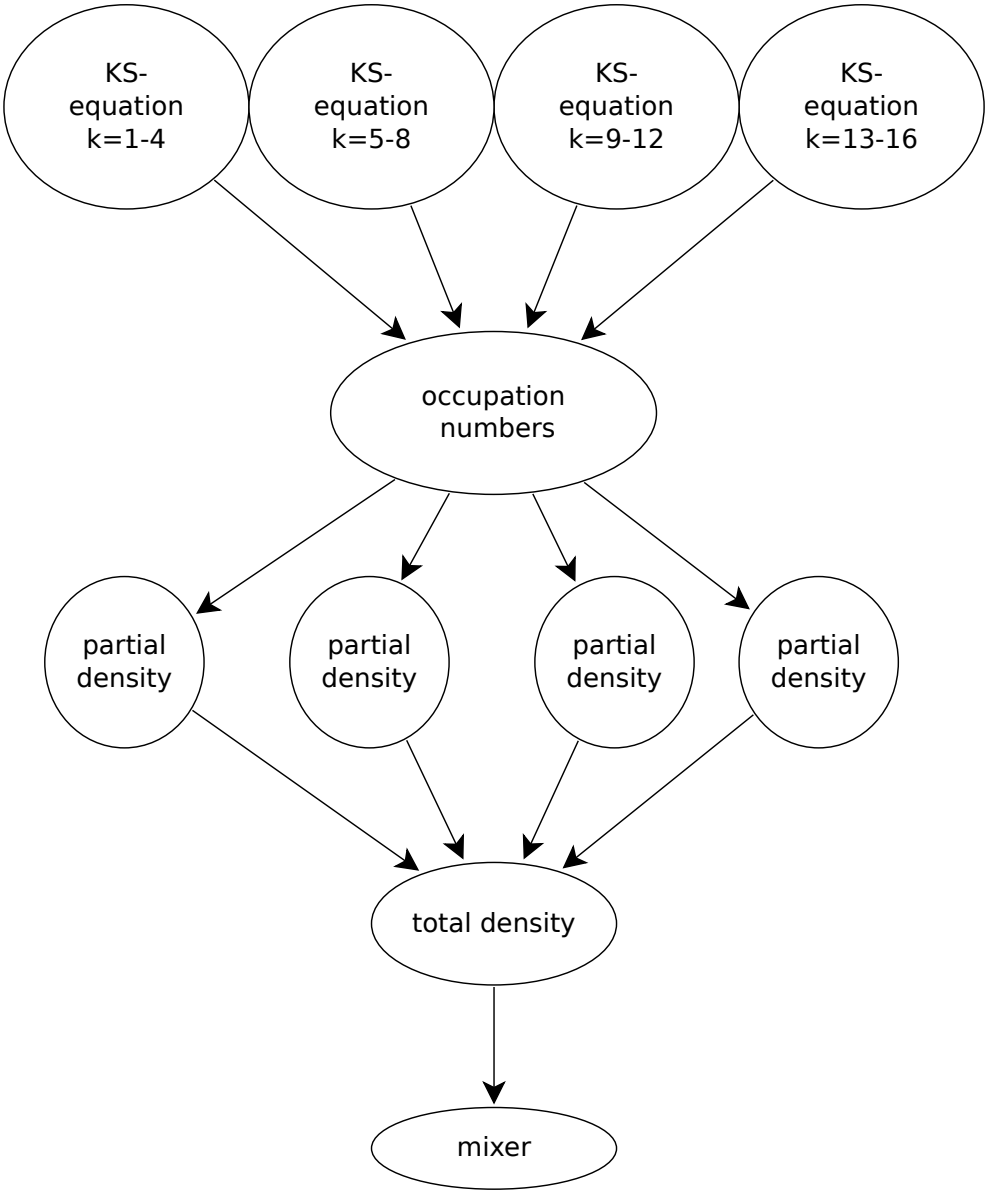

Figure 3.1: This diagram shows the dependency graph for solving the Kohn-Sham equation in the SCF loop. The 16 k-points are evenly distributed over the four processes.

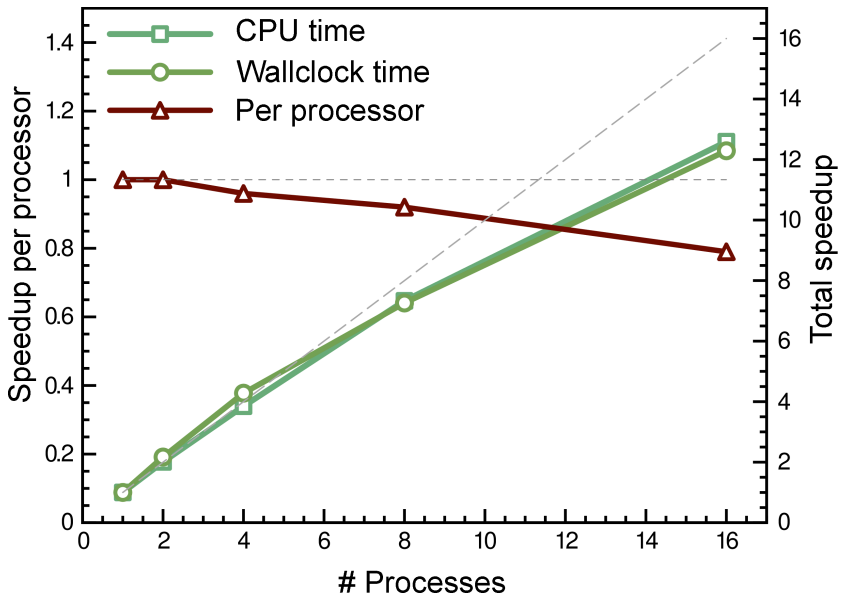

Figure 3.2: MPI scaling behavior. Green lines are speedup factors and the red line is speedup per processor. The dashed grey lines are the ideal case.

3.2 New Solvers in the `exciting` Code

Existing DFT codes as Gaussian [40] or WIEN2k [9] use refinement methods to converge approximate eigenvectors. We implemented such a method in **`exciting`** too. We implemented a version of a RMS-DIIS solver for **`exciting`**. Unfortunately, the implementation did only work for extremely simple structures and could not be made stable enough to be a good replacement for the direct solver.

In the **`exciting`** code, best results were reached with the shift-invert algorithm. The implementation of the algorithm in the ARPACK library outperforms the direct solver whenever the fraction of eigenvectors searched is particularly small. When used with parallel numerical libraries it is faster for all systems of significant size.

Figure 3.3 illustrates the parallel efficiency with multi-threaded libraries. The reference for the speedup is the time of the single-threaded LAPACK solver. Table 3.1 gives the numbers from Figure 3.3. The benchmarks where performed on 4-core power 5 nodes. The two materials are a *polyacetylene* (PA) crystal and a *naphthalene molecule* (2A).

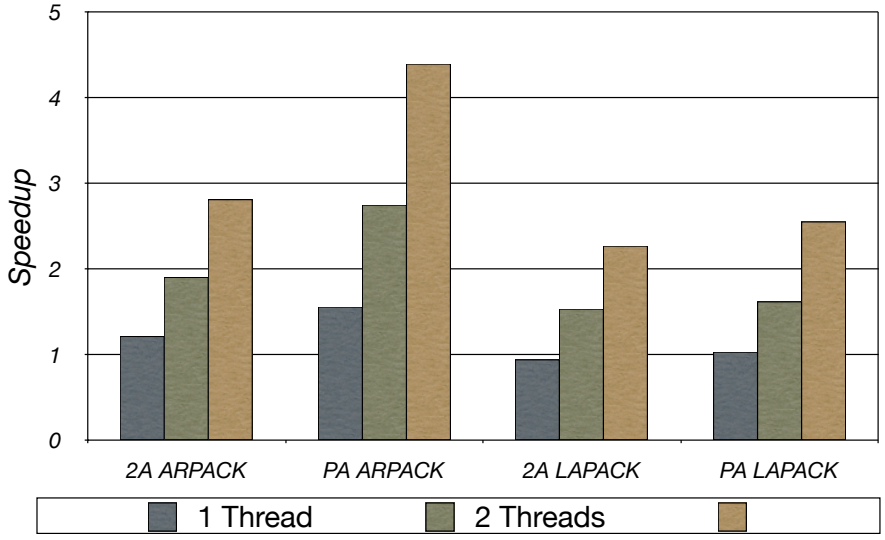

Figure 3.3: Multi threaded efficiency of ARPACK and LAPACK solver. Speedup to the single-threaded LAPACK solver for PA and 2A.

	Base time	1 Thread speedup	2 Thread speedup	4 Thread speedup
ARPACK 2A	4:50:18	1.21	1.90	2.81
ARPACK PA	0:57:16	1.56	2.75	4.39
LAPACK 2A	4:50:18	0.94	1.53	2.27
LAPACK PA	0:57:16	0.97	1.62	2.55

Table 3.1: Speedup for multi threaded libraries for PA and 2A.

3.3 SMP Optimization

Profiling the **exciting** code shows where most of the time is spent. These hot spots are the solution of the eigensystem, but also the matrix setup. Maybe surprisingly, the most time-consuming part of the matrix setup is not calculating the integrals, it is the rank-2 updates of the type:

$$\mathbf{A} \Leftarrow \mathbf{A} + \alpha \boldsymbol{x}\boldsymbol{y}^* + \alpha \boldsymbol{y}\boldsymbol{x}^*. \tag{3.1}$$

Multi-threaded math libraries have this procedure available in a multi-threaded fashion, but only for matrices stored in the regular column-wise pattern. **exciting** used to store the matrices in packed form which allows to save half the memory for hermitian matrices. There is, unfortunately, a penalty to that. It complicates dividing the processing to multiple threads. The packed-form procedure lacks a multi-threaded implementation. In the ESSL library no LAPACK procedures for packed matrices have a multi-threaded implementation. This leads to the conclusion that regularly stored matrices must be seen as a requirement for multi-threaded processing.

By replacing the hand-coded version of the rank-2 update with a multi-threaded version an almost optimal parallel speedup in the matrix setup could be achieved.

The direct eigen-solver (`zhpgvx`) unfortunately doesn't perform that well. In order to leverage SMP for the eigen-solver, a different algorithm has to be selected. It turned out that the inverse iteration fulfills this requirement. The LU-decomposition and back-substitution operations needed for the inverse iteration perform well with multiple threads. In addition, there is a established library, ARPACK [33], which performs this algorithm. [h!] Figures 3.4 and 3.5 show the parallel efficiency of the **exciting** code with the ARPACK solver. The speedup reaches its maximum at about 9, *i.e.*, when the parallelized calulation is about 9 times faster than the single-threaded run. The test system was a two-socket 12-core Opteron system, that is a total of 24 cores. The calculation used about 42 GB of Memory. The test material was a naphthalene molecule. The benchmark was run with `rgkmax` (A.11.41)= 5 and a unit cell size of $24.55 \times 28.34 \times 15.11$ *Bohr*, which leads to a matrix size of 36483.

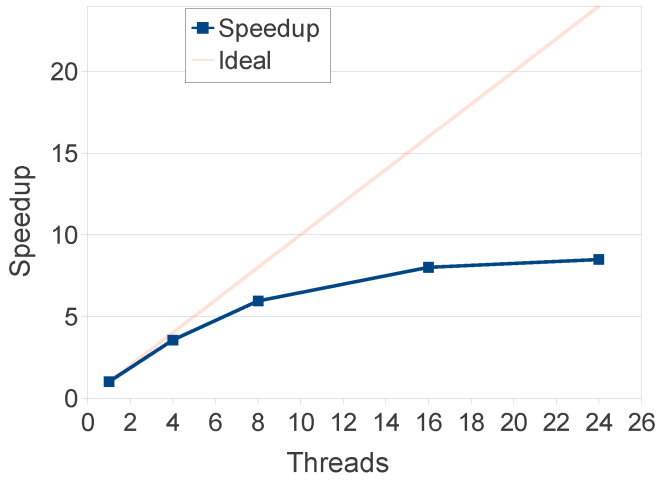

Figure 3.4: Speedup for a naphthalene molecule on a 2×12 core Opteron system. The red line indicates the graph for ideal efficiency.

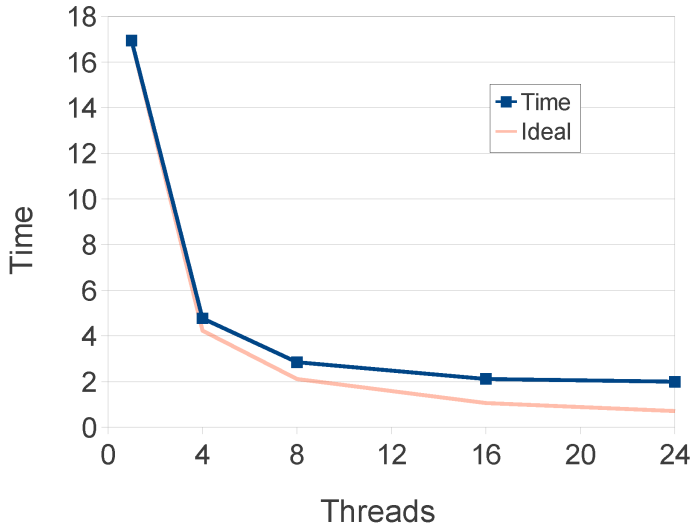

Figure 3.5: Total time for a naphthalene molecule on a 2×12 core Opteron system. The red line indicates the graph for ideal efficiency.

3.4 Abstraction of the Matrix Data Structure

As mentioned above, the data structure of the matrices used in the linear-algebra operations matter for the performance. An analysis of the code reveals that there are only nine different operations performed on the Hamilton and overlap matrices. This makes it practical to hide the actual matrix format behind an *eigensystem API*. Table 3.2 lists the API for the eigensystem and hermitian matrices.

As a future step, this API could be generalized to work also for distributed matrices as they are required in the ScaLAPACK library.

3.5 New Mixing in the `exciting` Code

We have ported the code written by Marks [35] to **`exciting`**. The mixer was primarily written to work with WIEN2k but the functionality is suitable to be ported to other DFT codes as well. The current port does not implement the rescaling preconditioning as described in Section 2.9. Figure 3.6 compares the MSB1 with the Pulay mixer and the linear mixer. All the tested examples show a better convergence for the multi-secant Broyden algorithm.

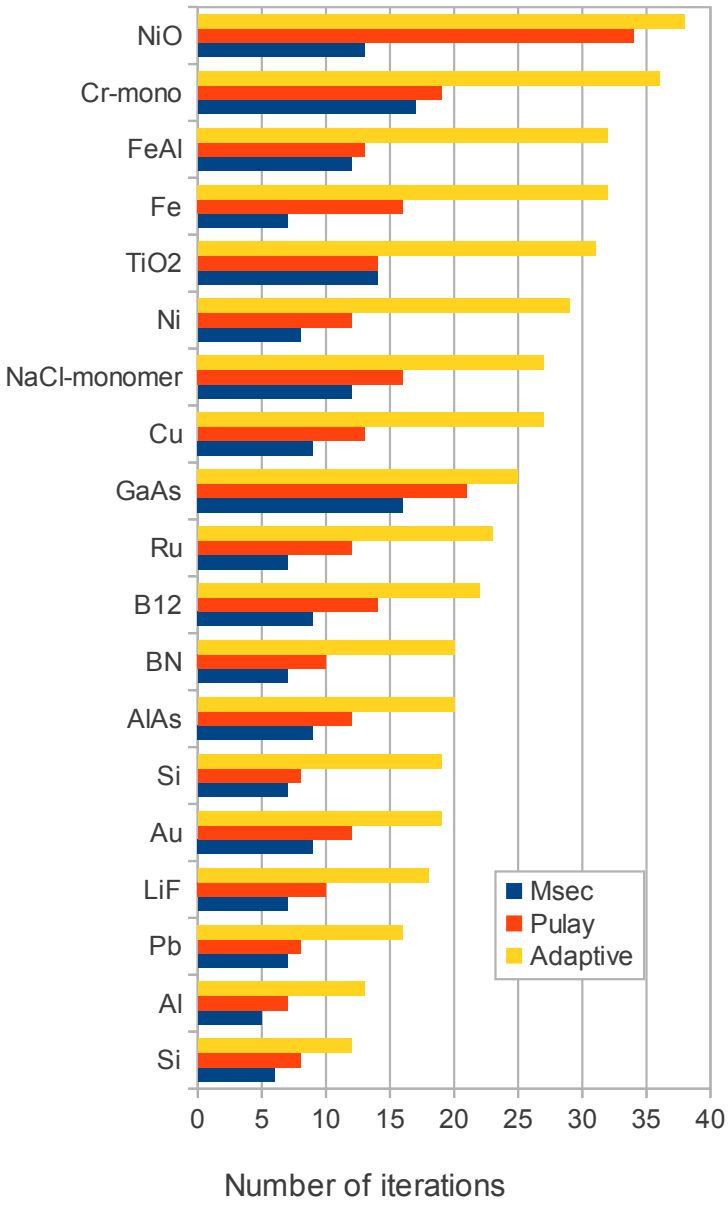

Figure 3.6: Comparison of the multi-secant Broyden mixing scheme (blue) with Pulay (red) and adaptive linear mixing (yellow) in the **exciting** code for a variety of materials.

`newsystem`(self, packed, rank)
`deleteystem`(self)
`Hermiteanmatrix_rank2update`(self, n, alpha, x, y)
`Hermiteanmatrix_indexedupdate`(self, i, j, z)
`Hermiteanmatrixvector`(self, alpha, vin, beta, vout)
`HermiteanmatrixLU`(self)
`Hermiteanmatrixlinsolve`(self, b)
`HermiteanMatrixAXPY`(alpha, x, y)
`HermiteanMatrixcopy`(x, y)
`HermiteanMatrixTruncate`(self, threshold)
`HermiteanMatrixdiagonal`(self, d)

Table 3.2: API to access and manipulate the eigensystem matrices.

Part II

User Inteface

Chapter 4

XML Technologies in Science Applications

4.1 XML

XML is a powerful data-description language. XML stands for *extensible markup language*. Its goal is to be both human and machine readable. The XML specification was developed by the W3C (*world wide web consortium*). It is a language designed to develop domain-specific languages.

XML technologies are already used in scientific computing. The work of White et al. [54] showed how the use of XML standards in computational materials science helps in portability, visualization, and analysis. Qbox [24], a plane wave DFT code, also makes extensive use of XML technologies, even for high-throughput file access. We followed pretty much their arguments, but extended the application of XML for the configuration files too, which opened a few new opportunities concerning the design of a user interface.

The basic unit of XML is the XML document. The XML document is a text file, usually encoded with UTF-8 unicode. It contains one *root element* as the root of the document tree. This hierarchical tree is built up by populating the root element with other elements, which may themselves contain elements too. Elements are denoted with the angle-bracket tags as known from HTML.

```
<element></element>
```

These elements can have attributes. Attributes are key-value pairs written into the opening tag.

```
<element attribute="value"></element>
```

Attributes as well as elements may have any name, except, they cannot start with a number or contain a space or other characters that have a special

meaning in XML like `">"`,`"<"` and `"&"`. Apart from attributes, elements can contain also text content.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <input>
    <titel>foobar</titel>
    <structure>
      <crystal>
        <basevect><basevect>
      </crystal>
    </structure>
</input>
```

This listing shows a well-formed document. The first line contains the XML declaration which declares the file to be an XML document, and specifies the encoding. UTF-8 is a superset to ASCII, which means that ASCII is also valid UTF-8, but UTF-8 allows to use all the special characters in unicode. The well-formedness of a document means, that it satisfies the basic rules of XML syntax. These rules are:

- A document must contain only one root element.
- For every open element tag there must be a closing tag.
- Elements must not interleave.
- Elements can have only one attribute with the same name.

XML itself gives no other limitations than these rules. Elements and attributes of any name can, in pricipal, appear in any order. In order to design a file format for a specific application, one can compile rules about what elements can appear where, and in what order, also which attributes they may or must have. Such definitions are called *schema*. There are a couple of such schema languages. The most basic one is the DTD, the Document Type Definition. When such a schema exists, the validity of any document can be checked by computer programs. Each of the schema languages has validation tools, which are, for example, built into XML editors, so they can check for validity while editing. Another, more powerful, schema language is "XML Schema". We have chosen to use XML Schema to define the **`exciting`** input-file format.

4.2 XML Schema

XML Schema is a standard by the W3C [48]. It is a language to express the grammar or *schema* of an XML file format. In XML Schema, one specifies

in what order and where elements may occur, and what attributes they may have, but also the type of data elements or attributes may contain. The data can be of a simple type as integer, float or string, or more complicated, as a selection of defined strings, or as general as a regular expression. The values may also have a default value, and the syntax allows to put documentation right next to the definitions. Listing 4.1 shows a simple example of an XML Schema definition. XML Schema is defined in XML syntax. There are tags to describe the elements and attributes and their type. Modern XML editors have special Schema editor functions that simplify building complex schemas significantly. The most common use for the XML Schema is to validate XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="foo">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ding" maxOccurs="3" minOccurs="0">
                    <xs:simpleType>
                        <xs:restriction base="xs:token">
                            <xs:enumeration value="a"/>
                            <xs:enumeration value="b"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="bar" type="xs:integer"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Listing 4.1: Example for a simple XML Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<foo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="schemafoobar.xsd">
    <ding>b</ding>
    <ding>a</ding>
    <ding>b</ding>
</foo>
```

Listing 4.2: This XML file is valid according to the schema definition in Listing 4.1.

files. A file that validates against a schema can be expected to conform with

the definition and cause no error in programs that expect inputs according to this schema.

A simple way to validate an XML file against a schema is the command line tool `xmllint`. The command line to validate the file from Listing 4.2 against the schema from Listing 4.1 is:

```
# xmllint --schema schemafoobar.xsd foobar.xml
```

In case the XML document is not valid according to the schema, the command returns an error report.

Another important application is to generate parsers from the XML Schema. The Schema has information about the legal inputs and about their data type. This is exactly the information that the program reading the input must have in order to convert the text into primitive data types in the programming language.

4.3 XML Namespaces

XML tags and attributes can have a namespace prefix. In Listing 4.1 every element and attribute begins with `xs:`, this is the namespace prefix. The namespace is defined by the `xmlns:xs` tag in the root element. The value of this tag is the namespace identifier, it is often an URL but the requirement is only that it is a unique name. In the example in Listing 4.1 the namespace definition defines the `xs:` prefix to be in the XML Schema namespace, thus the element and attribute names have the meaning according to their definition in XML Schema.

Applications that can read XML Schema use the namespace definition to recognize that they should interpret the tags with that prefix as XML Schema. Tags with an other prefix would be ignored by the application that reads the XML Schema. Like this, it is possible to embed multiple different languages within each other, because each application that understands only one namespace can ignore the others. This feature is in fact, what makes XML extensible.

4.4 XML Parser

The general definition of a parser is: *A parser is a tool that analyzes a text file and recognizes its structure. A parsers transforms the text file into a format better suited for further processing.*

XML parsers are an interesting species, because there are parsers that have a standardized model and API. There is the concept of the event-driven

parser which is called SAX for *simple API for XML*. It is not formally standardized but available in so many implementations that it may be regarded as a standard. The *document object model* (DOM), however, is standardized [23]. A DOM parser translates the XML file into a tree like data structure of nodes that contain the data. The DOM has a defined API to navigate and manipulate the DOM tree. The DOM representation of the XML document allows for much easier manipulation of the content than in the text representation.

Due to the popularity of XML, parsers for XML are available in most programming languages, which makes XML documents easier to manipulate and the text manipulation with regular expressions can mostly be avoided.

4.5 FoX: an XML Parser Library

There is a parser for XML written in FORTRAN. It is called FoX [54] [53]. FoX provides libraries to parse and write XML files. The FoX parser will read a valid XML document and construct a DOM tree in FORTRAN datastructures. It is a tree of which the nodes are elements, attributes, or contentnodes. FoX provides an API for navigating this tree and accessing its content.

For writing XML, a DOM can be serialized into an XML file or one can use the wxml API to write XML files. Additionally, FoX provides an API to write CML (*chemical markup language*), a standard for chemical structure information.

4.6 XSLT

The main goal of XML is to store structured data. One can implement very general data formats that represent the meaning of the data in all possible aspects. This is in itself very useful, but one might imagine that it is a very frequent task to convert the XML data into other formats. This is what XSLT (*extensible style sheet language transformations*) is for.

XSLT is a language to create so called *templates* which transform the input XML data into some output file or files. Some input data, *e.g.* an `exciting` output file, is transformed by the XSLT processor to an output, according to a template. The input data must be XML, the output can be XML or any other text format.

XSLT is also written in XML syntax. The basic building units are templates. Templates may also be called by name and with parameters and therefore they implement the concept of a function. From within XSLT, it is

very easy to address data in the source document by using XPath expressions.

XPath allows to select node sets with a UNIX-directory-like path string. *Node sets* are sets of XML nodes (elements, attributes and text) that satisfy the XPath query. XPath also provides syntax to express conditions, so-called predicates. XPath also includes a set of functions and operations to evaluate expressions with nodes, numbers, and text. XSLT has variables, but they are strictly immutable. XSLT is a very reduced and specialized language, which makes it very efficient when used within its domain. It is efficient, as it is fast and easy to develop templates, as well as regarding the speed of the transformations.

The tool that performs the XSLT transformation is called *XSLT processor*. There exist a number of implementations of XSLT processors. The most important ones are SAXON [30], XALAN [18], and libxslt [50]. Libxslt is today one of the fastest and most complete implementations available. To use it, there is the `xsltproc` commandline tool. This tool is simply called with the source document and the template, and returns the result of the transformation.

```
# xsltproc template.xsl sourcedoc.xml > result
```

As this tool is usually already installed on all recent UNIX-like operating systems it allows for implementing portable and powerful XML work flows.

4.7 XForms

XForms [51] is an embedded language to express form logic. The data which is edited trough this form is represented in XML. XForms are written in XML themselves, which allows for embedding them into other XML languages such as XHTML. Unfortunately, today's browsers do not support XForms directly. But there are multiple ways to translate XForms markup into HTML and javascript, one of which is XSLTForms [15]. The resulting web page can be rendered in all common web browsers and delivers a rich interactive user interface.

4.8 XML Databases and Data Mining

The subject of databases and data mining in computational science mainly comes up in two scenarios: High throughput-calculations and general knowledge-bases. High-throughput calculations are about calculating a property of thousands or maybe tens of thousands of compounds. The data mining then consists of searching for particular properties and correlations. For this

purpose, the data sets are rather well defined and not expected to change during the research campaign. Searching correlations requires the database to be very fast.

Knowledge databases have different requirements. For a knowledge database for atomistic calculations, it may not be that easy to know the structure of the data in advance. The data is rather a collection of documents, inputs and outputs, maybe with annotations and meta-data. The main purpose of the database is to make queries for searching particular datasets rather simple. It may be more important for a knowledge database, that it is easy to build and develop, than it being able to perform many queries per seconds.

4.8.1 Databases

Databases or DataBase Management Systems (DBMS) usual solve two problems: store data efficiently on hard-disks and provide a concept to perform queries on the data.

The classical database is the relational database. It is able to store tables with many, many entries, and perform queries with SQL (*simple query language*). SQL is the language to select data from a relational database. The database stores the data on the disk, and automatically creates indexes to make, search, and sort operations very fast. In order to store the data in tables, one has to know the structure of the data very well in advance. It is not impossible to alter the data structure later on, but this can be complicated, especially when the data is complexly structured.

In the 21st century, a variety of other database paradigms where invented that are known as NoSQL databases. Most of them address the problems, that come with high-traffic web applications. They try to trade some of the features of a relational database for better scaling properties, when the data gets too big to fit on one computer. Mainly, they are classified as key-value stores or document-oriented databases. The scenarios in web applications are so different from the requirements of scientific data-mining that most of them will probably never be considered for that purpose.

4.8.2 XML Databases

Although one could argue that XML databases are document-oriented NoSQL databases, the rationale behind them is different from the one of horizontally-scaling NoSQL databases for web applications. XML databases focus on XML documents to provide a database for special applications, where XML is the data format of choice. An XML database can store collections of XML documents in efficient data structures. The smallest unit is the node, *e.g.* ,

an XML attribute, element, or text node. The data structure is efficient in the sense that it can perform queries on the level of any node efficiently. It is very fast to answer questions like: Give me the values of a specific attribute in all documents and sort it numerically. That works with any node stored in the system, without the necessity to design a data model in advance. Building an XML database, demands less decisions while collecting the data. It can be used as an ignorant data store, with the security that queries on a reasonably big dataset will be fast enough in the end. Simple queries will work just fine and demand less abstraction from the user. The user will only have to understand the file formats, not much more. If queries have to make more complex comparisons or search for numeric values, XML databases can build indexes for the fields in question. In order for indexes to work, the database must have knowledge of the data type, which has to be provided to optimize this kind of queries.

The real advantages of XML databases are the advantages of XML itself. XML is unique in the way that it has the features and tools to develop domain-specific data-description languages. A native XML database supports all these features which, in turn, allows to create interchangeable file formats and the associated work flows.

4.8.3 XML and Other Data Serialization Languages

Today there are 2 different popular formats to encode structured data in text files: JSON (*Javascript object notation*) and XML. Each of them has its specific use cases where it works best. JSON is widely used in web applications to exchange structured data between the components of the application. It is simple, can be parsed by almost any language, and does the job of describing hierarchical data, but not much more. XML has these features as well and, moreover, it is extensible, has name-spaces, and has a lot of tools built around it.

For scientific data, XML is still a very good choice. Only if the encoding in text files is a problem in itself, binary file formats like NetCDF [41] are sometimes a necessary choice. In practice, text-based formats are so much easier to work with that binary formats are only used when absolutely necessary.

4.8.4 Combination of XML Databases and Relational Databases

Finally, I want to propose the possibility of combining XML databases with relational databases for data mining. There is no reason that data-mining

must be performed on the system that continuously gets updated with new data. XML databases are well suited to build up a data-store, validate, and organize the data, and archive it for further use. Relational databases, in turn, are faster and often can be directly accessed by data-mining software. Thus, one can combine the benefits of both databases by querying the XML database for the data that is significant for the data mining, and putting it in a relational database. This way, the advantages of the XML database – flexible, easier to build – are combined with the advantage of relational databases, which is their speed.

4.8.5 eXistdb, an Open Source XML Database

eXistdb [36] is an open-source XML database. eXistdb is a java program, that can efficiently store and query XML data, but also provides an HTTP interface. It allows to store programs written in XQuery inside the database. Such an XQuery script can be accessed via HTTP, thus it is possible to create entire web-applications that process XML data. XQuery is an XML query and transformation language much like XSLT, but with a more condensed syntax, and primarily designed to operate on whole collections of documents.

This environment is perfect for storing XML data, and providing a web interface at the same time.

Chapter 5

Optimizing the User Interface

5.1 The Complexity Wall

Scientific codes, like all software projects, get more complex with every new feature added. The more complex the code gets, the more important it becomes that it has a good user interface. If the user interface is not improved, every new feature makes the code harder to use, and it takes longer to become productive.

As many scientists who develop code, rarely have experienced a formal education in software engineering or user-interface design, the knowledge thereof propagates only slowly among scientists. To conquer the increasing complexity in software, one needs tools and concepts from software engineering to find abstractions in the software that decompose it into parts which are understandable in reasonable time.

In praxis, a young researcher entering the field has two to three years to learn the code, to setup the systems to be studied, and to publish results. This timescale is likely to remain the same in the foreseeable future. If the time for learning the tools and processes ever increases, good results get ever less likely. So the challenge is to regularly reduce complexity, in order to allow progress to persist.

Scientific computing often requires a complex tool chain for setting up calculations and analyzing results. This tool chain must be reviewed and modernized regularly. All technology is transient and will eventually be replaced by something more abstract, simpler, and more powerful. The UK Computing Research Committee enlisted the grand challenges in computing research [29]. They list the integration of new tools and languages as necessary to overcome current and foreseeable challenges in computing. There is, however, another way to accelerate progress, and this concerns the data.

The open-gene data bases [37] in life sciences were very successful, and revolutionized the way science is done in the field. This suggests, that openness and data reuse really can change things.

Data may be chemical structures, crystal structures, input parameters but also result data, as calculated physical properties. The current method to share and publish data is in journals as print-optimized documents. This is not optimal for several reasons: It obscures reproducibility, complicates data comparison, and it complicates building on old results.

If results were available in a variety of machine-readable formats, searchable on the Internet, and quick to assert for usefulness, it would take a lot of friction out of the scientific process, and potentially lead to an explosion of knowledge. Additionally, it could re-introduce reproducibility to computational science on a much more practical level. These are some of the observations, that did influence the **exciting**@web project, as described in the last section of this chapter (5.7).

In the following, we want to examine what should be expected from the user interface of the **exciting** code and how it can be implemented. In the end, we will discuss the concept of a *graphical user interface* (GUI) as it is part of **exciting**@web.

5.2 The exciting User Interface

The interaction with scientific software as **exciting** is rather complex. Setting up the structure and optimizing the parameters is necessary and cumbersome. Usually, the user has to use some kind of scripting to automatize the tasks.

The interface that allows this kind of automation and can accommodate any level of complexity is the configuration file. But what makes a good configuration file format? What problems must a good file format solve? I will discuss the main observations about file formats, that eventually led to designing the **exciting** XML input file format.

Although the main purpose of a configuration file is to be read by a computer program, it is quite important that it helps other humans to understand what is expressed in the commands and parameters in the file. It is not easy to say, what an optimal format is, but it is easy to say what helps and what does not help. In my view, there are 4 important points to consider:

- All parameters should have expressive names. Abbreviations should only be used when they are very common, and number codes are problematic in any case. It should be possible to get a basic idea what is

happening, without the need to constantly look up cryptic names or features that are encoded by numbers in the documentation.

- Parameters that are logically associated with each other or with a particular feature should be grouped together in the input. This requires the file format to provide some sort of hierarchical organization or grouping features. The advantage is, of course, that any knowledge of the logic of the program helps to navigate in the input, and a parameter may reveal more of its meaning, by being clearly associated to a feature or group of parameters.
- It is important that the user can get meaningful feedback from the program if the configuration file contains errors. This requires careful processing of the input by the program which, at best, should be automatized to ensure consistency.
- The file format should have a systematic structure that can be manipulated easily and securely by other tools and scripting languages.

The old **exciting** file format did fall short in each of these points. It wildly used number codes to switch on certain features, it had very cryptic abbreviations, it did not group parameters, and for any list of items, one had to write the number of items to follow in the first line, or the file could not be read correctly. Also, the code that was responsible for reading the inputs, was one of the hardest-to-maintain parts of the program. Particularly the documentation and the actual behavior of the program used to diverge because the documentation was not updated with the necessary care. Sill, the code could be handled as long as it only treated ground-state properties. With the introduction of the excited-state features, however, the difficulties became even more apparent, because the number of parameters almost doubled at once. Switching to XML syntax for the input file allowed for systematically addressing all the above issues and for some additional benefits that come with the elaborate tool set available for XML.

5.3 The New XML Input for the exciting Code

XML allows to organize data hierarchically, which makes it a suitable format for structuring the input parameters into meaningful modules. A hierarchical structure is necessary to make the number of parameters manageable and to make complex inputs better readable. Related configuration parameters can be clearly associated in the textual representation as well.

```
<?xml version="1.0" encoding="UTF-8"?>
<input xsi:noNamespaceSchemaLocation="http://xml.exciting-code.org/
    excitinginput.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <title>Lithium Fluoride</title>
  <structure speciespath="../../species">
    <crystal>
      <basevect>3.80402 3.80402 0.00000</basevect>
      <basevect>3.80402 0.00000 3.80402</basevect>
      <basevect>0.00000 3.80402 3.80402</basevect>
    </crystal>
    <species speciesfile="Li.xml">
      <atom coord="0.0000 0.0000 0.0000"/>
    </species>
    <species speciesfile="F.xml">
      <atom coord="0.5000 0.5000 0.5000"/>
    </species>
  </structure>
  <groundstate lmaxvr="8" vkloff="0.05 0.15 0.25"
  lradstep="2" lmaxapw="10" swidth="0.0001" ngridk="4 4 4">
  </groundstate>
</input>
```

Listing 5.1: Example for an **exciting** XML input file

5.3.1 The Input-File Concept

As described before, elements and attributes allow for a hierarchical structure of the input file. An example input file is provided in Listing 5.1. In the following, the green attribute names and blue element names are followed by the section number in the appendix where more detailed information can be found.

The input (A.2) element is the root element of the **exciting** input file. It must contain at least the three elements title (A.3), structure (A.5), and groundstate (A.11), where each of them must be present only one time. The structure (A.5) element contains all structural information, such as unit-cell parameters as well as type and position of each atom. The groundstate (A.11) element is required for any calculation. Its attributes concern the methods and parameters used to calculate the groundstate density. These are the essential inputs. For properties and producing the corresponding output data, the properties (A.17) element allows for specification of the requested calculations and outputs. The excited-states features are grouped in the xs (A.48) element. Two more elements are defined for the structure-optimization (structureoptimization (A.16)) features and the phonon cal-

culation (phonons (A.41)). Details can be found in Appendix A.

5.3.2 File Format Definition with XML Schema

As described in Section 4.2, one can use XML Schema to formally describe an XML file format. Listing 5.2 shows the definition of the atom (A.9) element with its attributes. Such a formal description, when available, is very useful for a number of things. It allows for an automatic generation of the documentation with the correct data types and default values. And it allows to generate the code that reads the input automatically. The functionality to translate the text of the input file into a form that can be referenced by the program is called *parsing*.

5.3.3 `exciting`'s XML Parser

Listing 5.2 shows a section of the **`exciting`** input file schema. The schema, written in XML Schema (Section 4.2), describes the grammar of the input file. Each parameter can have properties such as:

Type: Each input can have a simple data type such as string, float or integer.

Occurrence: In the case of elements it states how often they may or must occur. For attributes it says if they are required or not.

Restrictions: Values may be restricted to be chosen out of a selection.

Default: Attributes may have a default value, which is used if they are not explicitly set.

This is the information needed to evaluate the DOM of the XML input and to create a data structure that contains the parameters as native FORTRAN data types, and to assign default values to optional parameters. **`exciting`** uses FoX (Section 4.5) to transform the XML input into a DOM. At this point, the content is strings of text, stored in the DOM nodes. The parameters for **`exciting`** are used as strings, integers, floating-point numbers, or arrays thereof. To make the data available to the FORTRAN code, the strings in the DOM must be converted into FORTRAN native types. We do this, by generating FORTRAN code from the schema that can do this translation. XML Schema is written in XML. Therefore, we need a tool suitable to process XML and produce FORTRAN code.

There exists a special language to transform XML into other formats. It is called XSLT (Section 4.6). In this thesis, I have developed an XSLT template to generate the FORTRAN code from the XML Schema. It defines a derived

```
<xs:element ex:importance="essential" ex:unit="" name="atom" minOccurs="1"
     maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>Defines the position and other attributes of one
        atom in the unit cell.</xs:documentation>
    <xs:appinfo>
      <oldname>noname</oldname>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute ex:importance="essential" ex:unit="lattice coordinates"
        name="coord" type="vect3d" use="required">
      <xs:annotation>
        <xs:documentation>Position in lattice coordinates.</
            xs:documentation>
        <xs:appinfo>
          <oldname>atposl</oldname>
        </xs:appinfo>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute ex:importance="expert" ex:unit="" name="bfcmt" type="
        vect3d" default="0.0d0 0.0d0 0.0d0" use="optional">
      <xs:annotation>
        <xs:documentation>Muffin-tin external magnetic field in Cartesian
            coordinates.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute ex:importance="expert" ex:unit="" name="mommtfix" type="
        vect3d" default="0.0d0 0.0d0 0.0d0" use="optional">
      <xs:annotation>
        <xs:documentation>The desired muffin-tin moment for a Fixed Spin
            Moment (FSM) calculation.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

Listing 5.2: Excerpt of the schema defining the input file format.

data type for every element that contains the configuration parameters. Then it generates a subroutine for each element type. This procedure is called with the corresponding node in the DOM tree an an argument. It checks if the required parameters are given, assigns the default values, and checks if the right number of child elements is there. If the requirements are not met or an unknown parameter is given, the user is provided with an error message. The result is a FORTRAN data structure in which all inputs can be addressed. The code snippet in Listing 5.3 reads the input file and returns the FORTRAN data structure:

```
1 program main
2 use inputdom ! module that cares about the FoX Calls
3 use modinput ! module that contains the derived type definitions and
      parser functions
4 call loadinputDOM() ! loads xml to DOM and assigns inputnp from inputdom
      to document node
5 input=getstructinput(inputnp) ! builds the tree structure populated with
      default values and configured values
```

Listing 5.3: FORTRAN call to parse XML input.

From now on all the data from the input file is available as native FORTRAN data-type in the `input` structure. The values can be used in expressions directly without any further call to a function.

```
array(1:3)=input%groundstate%ngridk
```

One example: the expression `input%groundstate%ngridk` gives the integer array of the number of k-mesh points in each direction. All the other attributes are accessed in the same way. In case of multiple occurrences of an element the syntax to access one of the items in the list is the following:

```
coords(:)=input%structure%speciesarray(is)%species%atomarray(ia)%atom%
    coord(:)
```

The peculiar notation `atomarray(index)%atom[...]` stems from the fact that FORTRAN does not allow the construction of an array of pointers. Rather one can only define an array of derived types that contain a pointer.

There are a few general rules for the logic of the input file, partly because of the syntax of XML, but there are also conventions in the implementation that must be followed. Attributes are *required* if there is no meaningful default value possible such as in the `coord` (A.9.2) attribute. If an element occurs in the input file, it is initialized with all its available attributes. If a required attribute is missing, the program will abort with an error message. Optional attributes are initialized with their default values. Some elements can be repeated inside the parent element. Repeated elements are, for example, the `atom` (A.9) elements inside the `species` (A.8) element. Optional

elements act as a switch that activates the associated feature. Parts of the code, that process one optional element, must not rely on parameters defined in any other optional element. This is enforced by the fact that optional elements which are not present in the input file, are not initialized, and will cause a segmentation fault when referenced in the code. Therefore the existence of an optional element must be checked with the `associated()` inquiry function before its data may be referenced.

5.3.4 Input Documentation

XSLT has proven to be a comfortable way to transform the input schema into other formats. The other important information to come from the schema is the input file-format documentation. The XML Schema syntax allows to add documentation about the elements and attributes. We made use of this by writing the descriptions of the parameters right into the schema. The advantage of this is, that the documentation is right next to the type definition and the definition of the default values, which are at the same time used by the program. The documentation can be generated whenever the schema changes, such the documentation is synchronous with the code all the time. The documentation is available in two different formats. One is a PDF, compiled from Latex, the other one is a wiki syntax which provides the on-line help on the **exciting** website. This is done with the templtes: `schemetolatex.xsl` and the `schematowikidot.xsl`.

5.3.5 Species Files

The chemical elements used in the calculation are described in separate files. The **exciting** code comes with a set of files for all elements. These files describe the charge and predefined basis functions. As the proper usage of APW-derived basis functions is a pretty complex issue, it benefits a lot from the descriptive and hierarchical properties of XML. These files are generated by the `species` tool and don't need to be touched by the user. If it is necessary to change something, the descriptive file format lowers the barrier to do so effectively. Listing 5.4. shows the species definition for aluminum. The species files are parsed in the same way as the input file. In fact, the same XSLT template is applied to the species schema as to the input schema. Detailed documentation about the species file format is provided in Appendix B.

```
<?xml version="1.0" encoding="UTF-8"?>
<spdb xsi:noNamespaceSchemaLocation="../../xml/species.xsd" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance">
  <sp chemicalSymbol="Al" name="aluminum" z="-13.0000" mass="49184.33493">
    <muffinTin rmin="0.554700E-06" radius="2.0000" rinf="45.7440"
        radialmeshPoints="400"/>
    <atomicState n="1" l="0" kappa="1" occ="2.00000" core="true"/>
    <atomicState n="2" l="0" kappa="1" occ="2.00000" core="true"/>
    <atomicState n="2" l="1" kappa="1" occ="2.00000" core="false"/>
    <atomicState n="2" l="1" kappa="2" occ="4.00000" core="false"/>
    <atomicState n="3" l="0" kappa="1" occ="2.00000" core="false"/>
    <atomicState n="3" l="1" kappa="1" occ="1.00000" core="false"/>
    <basis order="1">
      <wf matchingOrder="0" trialEnergy="0.1500" searchE="false"/>
      <exception l="0">
        <wf matchingOrder="0" trialEnergy="0.1500" searchE="true"/>
      </exception>
      <exception l="1">
        <wf matchingOrder="0" trialEnergy="0.1500" searchE="true"/>
      </exception>
      <exception l="2">
        <wf matchingOrder="0" trialEnergy="0.1500" searchE="true"/>
      </exception>
    </basis>
    <lorb l="0">
      <wf matchingOrder="0" trialEnergy="0.1500" searchE="true"/>
      <wf matchingOrder="1" trialEnergy="0.1500" searchE="true"/>
    </lorb>
    <lorb l="1">
      <wf matchingOrder="0" trialEnergy="0.1500" searchE="true"/>
      <wf matchingOrder="1" trialEnergy="0.1500" searchE="true"/>
    </lorb>
    <lorb l="2">
      <wf matchingOrder="0" trialEnergy="0.1500" searchE="true"/>
      <wf matchingOrder="1" trialEnergy="0.1500" searchE="true"/>
    </lorb>
    <lorb l="1">
      <wf matchingOrder="0" trialEnergy="0.1500" searchE="false"/>
      <wf matchingOrder="1" trialEnergy="0.1500" searchE="false"/>
      <wf matchingOrder="0" trialEnergy="-2.4974" searchE="true"/>
    </lorb>
  </sp>
</spdb>
```

Listing 5.4: Species file of aluminum. The species file defines the properties of the basis functions that are used to expand the wave function within the muffin-tin sphere.

```
<?xml version="1.0" encoding="UTF-8" ?>
<symmetries HermannMauguinSymbol="Bmab">
<title>LaCuO </title>
<lattice a="10.0605232" b="10.0605232" c="24.972729" ab="90"
      ac="90" bc="90" ncell="1 1 1"/>
      <WyckoffPositions>
        <wspecies speciesfile="La.xml">
          <wpos coord="0.0000  0.0000   0.3608 " />
        </wspecies>
        <wspecies speciesfile="Cu.xml">
          <wpos coord=" 0.0000  0.0000   0.0000" />
        </wspecies>
        <wspecies speciesfile="O.xml">
          <wpos coord="0.2500  0.2500   0.0000" />
          <wpos coord=" 0.0000  0.0000   0.1820" />
        </wspecies>
      </WyckoffPositions>
    </symmetries>
```

Listing 5.5: Example input for the **spacegroup** tool.

5.3.6 Spacegroup Input

The program **spacegroup** is part of the **exciting** code. In the **spacegroup** input, one defines the space group by providing the Hermann Mauguin symbol and the Wyckoff positions. The Wyckoff position implicitly defines the positions of a group of equivalent atoms: After specifying the position of one representative atom, the positions of the other equivalent atoms are derived from the symmetry operations. This way, **spacegroup** determines all the atom positions in the unit cell. If required it can also create a supercell. The input file of **spacegroup** has its own file-format documentation (Appendix C). Listing 5.5 shows an example input file for the **spacegroup** tool.

5.3.7 Assisting Editors

XML has been in use for a long time and has many applications. Accordingly, the tools to process and manipulate XML are very mature. Modern XML editors can understand XML Schema and assist the user at editing files. After setting the location of the schema in the root element, the editor can validate the input for conformance with the schema. It can provide the user with suggestions of possible inputs in the current context (Fig. 5.1 and 5.2). The XML editor also has access to the documentation in the XML Schema. It can display the description of the element or attribute (Fig. 5.3). A modern XML editor can give feedback on the validity of the input while editing and

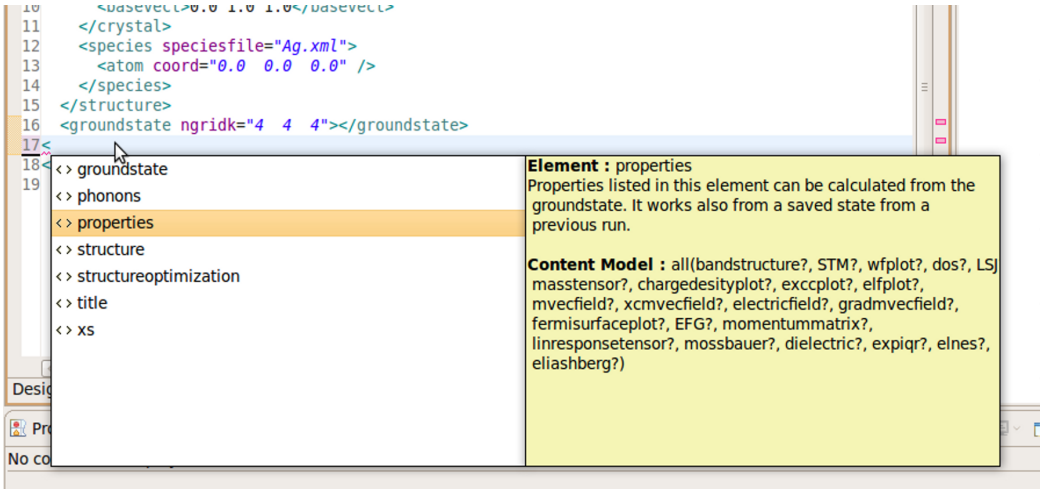

Figure 5.1: The XML editor in *eclipse* can use the XML Schema to propose possible elements in the current context.

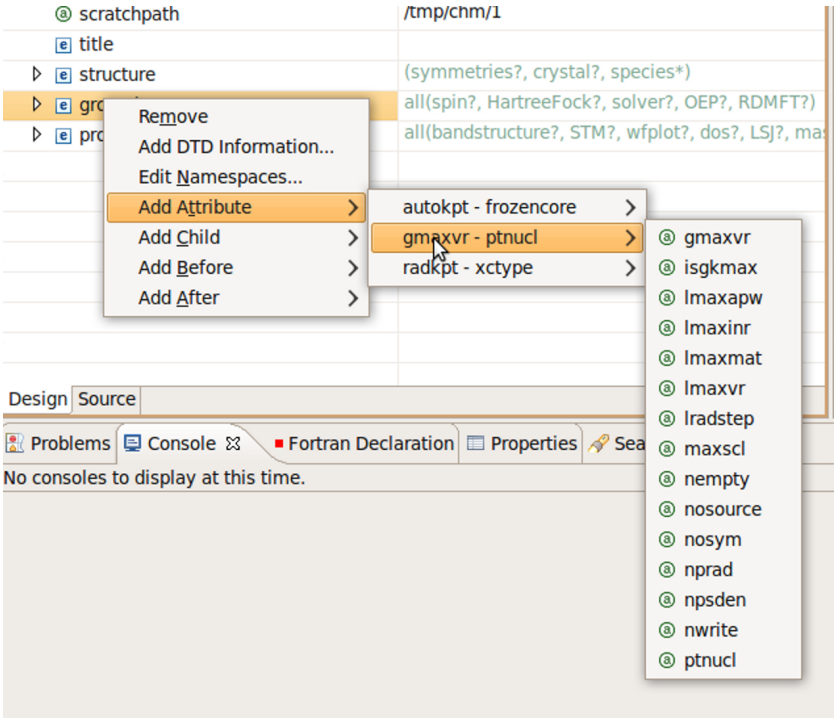

Figure 5.2: The XML editor provides the list of attributes, defined in the XML Schema, for the current element.

Figure 5.3: The XML editor shows the documentation from the XML Schema when the mouse hovers over the attribute.

assist the user to edit complex input files quickly.

5.4 XML Output for the `exciting` Code

Results of the calculations are a number of different properties and data sets. These data sets may be used for further analysis or visualization. Traditionally, the file formats were designed to be readable by various visualization tools. The downside of this is, that it pretty much forbids to have useful descriptions, meta data, units or labels in the output. Having all this descriptions in the output is desired because it makes interpretation of the data easier, especially when they stem from an older calculation. XML also supports the idea of creating data archives, which are potentially very valuable as the complexity of research moves on.

One of the goals of XML is the separation of data and presentation. XML is designed to store structured data, not to display it. Once data is structured and well organized, it is, however, very straight forward to transform it to multiple display or visualization formats.

```
<bandstructure character="true">
    <title>MgO</title>
    <species name="magnesium" chemicalSymbol="Mg">
        <atom coord="0.000000000       0.000000000       0.000000000">
            <band>
                <point distance="0.000000000" eval="-2.812873404"
                    sum="0.9974405788">
                    <bc l="0" character="0.9974404573"/>
                    <bc l="1" character="0.1015679110E-16"/>
                    <bc l="2" character="0.2407216115E-07"/>
                    <bc l="3" character="0.9734178263E-07"/>
                </point>
                <point distance="0.2790755614E-01"
                    eval="-2.812873832" sum="0.9974398065">
                    <bc l="0" character="0.9974396825"/>
                    <bc l="1" character="0.8973555232E-09"/>
                    <bc l="2" character="0.2459204751E-07"/>
                    <bc l="3" character="0.9850811722E-07"/>
                </point>
                [...]
```

```
# xsldproc xmlband2agr.xsl bandstructure.xml > graph.arg
```

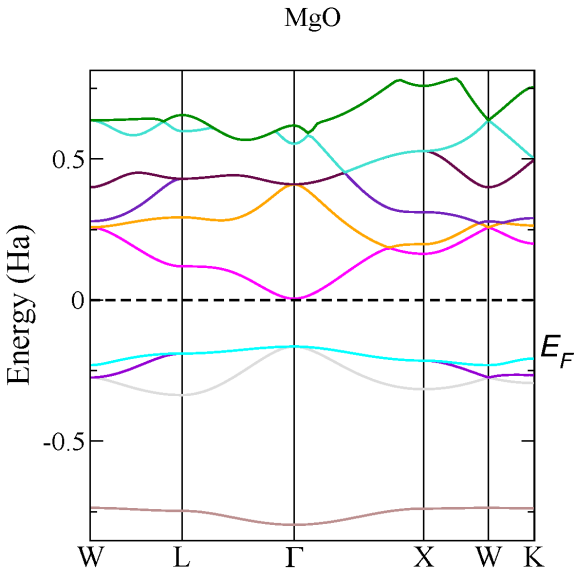

Figure 5.4: This band structure graph was generated by an XSL template from the `bandstructure.xml` file. The template generates a file to be visualized in `xmgrace`.

plot3d2opendxscalar.xsl	Template to transform any 3d plot data into OpenDX file format.
plot3d2xsf.xsl	Template to transform any 3d plot data into XSF file format. XSF can be read by XCrysDen and Vesta.
plot3d2xyz.xsl	This templates creates output in the form of "x y z" data from the plot3d output.
xml2ascii.xsl	Extracts all numbers from XML into a plane text file.
xmlband2agr.xsl	Template to generate an xmgrace plot from bandstructure.xml
xmldos2grace.xsl	This xsl style-sheet prepares a file for xmgrace to plot density-of-state (DOS) graphs.
xmlfermis2bxsf.xsl	This generates the input file to visualize the Fermi surface for XCrysDen.
xmlinput2xsf Xsl	Conversion template to generate an XCrysDen structure file (`.xsf` file) out of the **exciting** `input.xml` file.

Table 5.1: Visualization templates available for **exciting**.

exciting2sgroup.xsl	This template converts **exciting** input into the file format of the sgroup tool.
exciting2wienstruct.xsl	Template to create a WIEN2k `.struct` file from **exciting** `input.xml`.
xmltinputoblock.xsl	Converts `input.xml` to the block format used by Elk.
xmlinput2xsf.xsl	Conversion template to generate an XCrysDen structure file (`.xsf` file) from **exciting** `input.xml` file.

Table 5.2: Inputfile conversion templates available for **exciting**.

One way to transform the data into a visualization for XML is using *Templates*. Our approach is the following: **exciting** packs all the information available about a physical property into an XML file, regardless what the analysis tools can understand. Then a template is applied to this data which transforms it into various file formats readable by the tools. The templating language used is XSLT (Section 4.6). **exciting** comes with a wide selection of these templates. We have templates for 1d, 2d, 3d plots and more specific ones that create fully annotated graphs with title, axes labels, and legend (Tables 5.1 and 5.2). As an example Figure 5.4 shows the procedure for creating a band structure plot.

5.5 Work-Flow Concepts in exciting

If one develops a concept for a user interface it makes sense to distinguish the common work flows that need fundamentally different approaches in terms of configuration and program execution. The calculations with **exciting** follow one of three patterns: *single-SCF properties*, *optimization procedures* and *parameter sweeps*. Any problem amounts to one of them.

Many properties are derived from the groundstate density or Kohn-Sham-orbitals (Fig. 5.5 top left). Examples are: Band structure, total energy, Fermi energy, density of states, Fermi surface. Because these properties are derived from one fixed input geometry we shall refer to them as *single-SCF properties*.

Other questions in DFT come as optimization task. To find the optimal parameters one needs to do a full groundstate calculation for each step. The new step is calculated from the previous results after some algorithm that uses the history (Fig. 5.5 top right). The most important example oft that pattern is structure optimization. This kind of work flow we call *optimization procedure*.

The third pattern involves independent groundstate calculations for a set of points in some parameter space. The generation of such a parameter set is often referred to as DEO (*design of experiments*). This yields a function over this parameters from which properties can be derived. The points are independent from each other and can thus be calculated in any order or in parallel (Fig. 5.5 bottom). Examples are volume optimization or convergence tests. This type of calculation is called *parameter sweep*.

The preferred way to tackle this third type of calculations is through input templates (Table 5.3). Templates are used to produce a set of input files from a prototype by varying one or few parameters (Fig. 5.6). **exciting** comes with tools to generate parameter files. We provide also input templates to

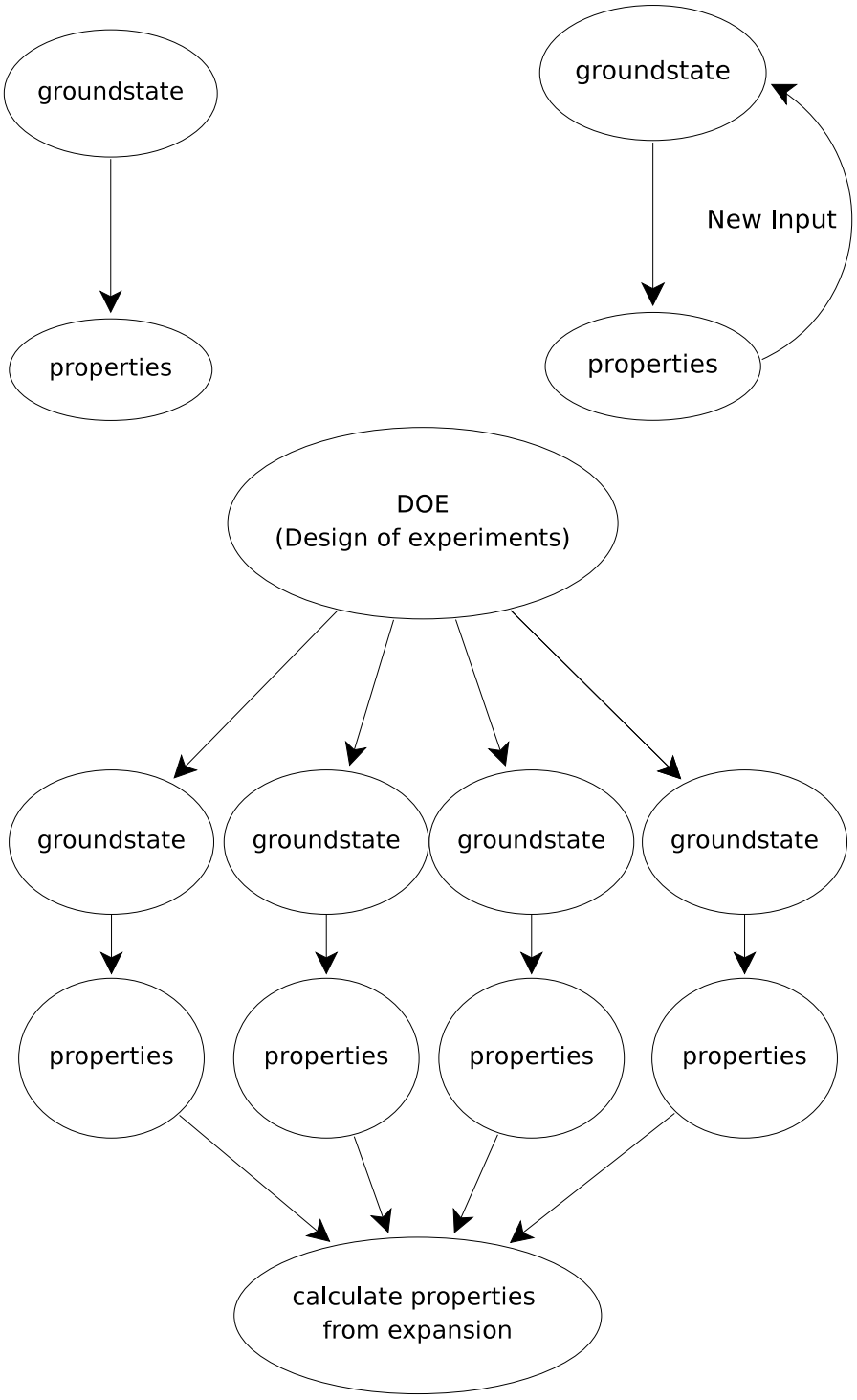

Figure 5.5: The three work-flow types are *single-SCF properties* (top left), *recursive optimization* (top right), and *parameter sweeps* (bottom).

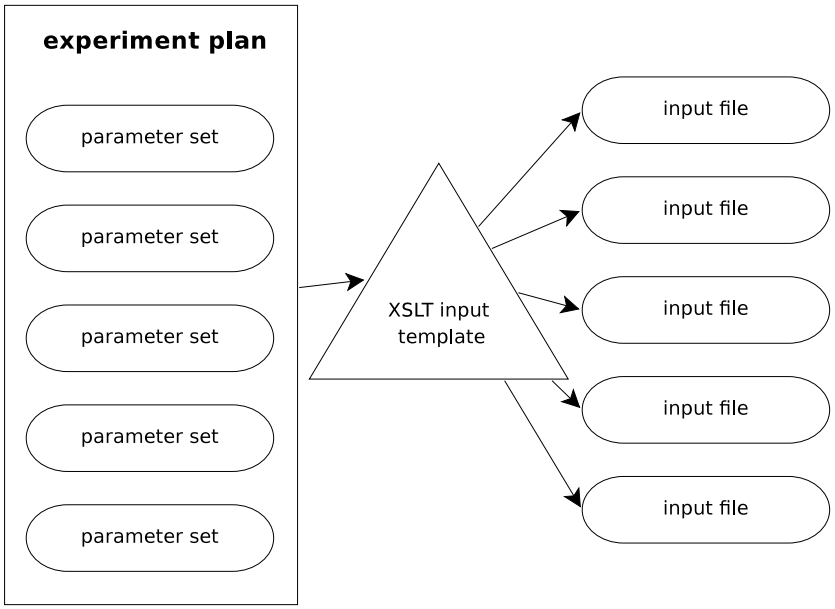

Figure 5.6: Input templates can be used to generate a set of input files from a list of parameters (experiment plan).

generate the input files for all the calculations on the **exciting** website [2]. For the use on HPC clusters, there are example templates to generate the scripts to submit the jobs to the cluster. XSLT templates can also be used to extract data from the output files and collect them in one file.

5.6 The exciting Code and ASE

As another new option to work with the **exciting** code we have implemented an interface to the **exciting** code for the *atomic simulation environment* (ASE). ASE [6] is a framework to manipulate, visualize, and optimize structures. ASE is an object-oriented framework written in python which provides an `Atoms` *class* to hold atomic structure information. An `Atoms` *object* can be created by importing structure information from a variety of file formats or by using ASE functions to create unit cells for bulk systems, slabs, and so on. An `Atoms` *object* can be connected with a *calculator object*, which can interface to a list of DFT codes. These codes give back total energies or forces, which then are used by ASE functions to perform optimization or analyzes.

The ASE-**exciting** interface written in this work, allows to call the **ex-**

expandset.xsl	Expands all parameter permutations. It is used to create a parameter list.
loadl.xsl	The template generates a loadleveler job script from the parameter list.
expandsetup.xsl	Create parameter series for use with `expandset.xsl`.
set2shellcommand.xsl	Creates shell-script to execute all calculations in parameter list.
setaddpath.xsl	Adds a path attribute to parameter sets in parameter list.
example_input.xsl	Example template for creating input files from parameter list.

Table 5.3: Input file templates available for **exciting**.

citing code from within ASE, and read from and write to the **exciting** XML input file format. The calculator object has methods to return total energy and forces in the units used by ASE.

5.6.1 ASE Interface to the exciting Code

An **exciting** *calculator object* is defined by the `Exciting` *class* within ASE. Calling the `Exciting` *constructor* returns a *calculator object*. The interface to create an **exciting** *calculator object* is as follows:

```
class ase.calculators.exciting.Exciting(dir='.', template=None,
     speciespath=None, bin='excitingser', kpts=(1, 1, 1), **kwargs)
```

It can be used in two different ways. One is by giving parameters of the groundstate as keyword arguments in the constructor interface:

```
class exciting.Exciting(bin='excitingser', kpts=(4, 4, 4), xctype='
    GGArevPBE')
```

The other way is to use an XSLT template:

```
class exciting.Exciting(template='template.xsl', bin='excitingser')
```

An example template is given by Listing 5.6. The use of the template has the advantage that it allows access to all possible configuration options of the **exciting** code, not only to those which are attributes to the `groundstate` (A.11) element.

The *calculator object* must be associated with the structure for which it should perform the calculations. This is done by calling the `set_calculator()` method of the `Atoms` *class*. For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" />
  <xsl:template match="/">
    <xsl:comment>
      created from template
    </xsl:comment>
<!-- ############# -->
    <input>
      <title></title>
      <structure speciespath="./">
        <xsl:copy-of select="/input/structure/crystal" />
        <xsl:copy-of select="/input/structure/species" />
      </structure>
      <groundstate ngridk="4 4 4" vkloff="0.5 0.5 0.5" tforce="true" />
    </input>
<!-- ############# -->
  </xsl:template>
</xsl:stylesheet>
```

Listing 5.6: Example for an ASE calculator template. One can change anything manually in the template except the part that is handled by ASE *e.g.* the unit cell and the atom positions.

```
structure = fcc111('Cu', size=(4,4,2), vacuum=10.0)
structure.set_calculator(Exciting(bin='excitingser', kpts=(4, 4, 4) )
```

5.6.2 ASE exciting Input/Output

ASE can read the geometry data from an **exciting** input file and create an `Atoms` *object* from it, and can write the structure stored in an `Atoms` object into an **exciting** input file. This means that the `read` and `write` functions of ASE have the option to read and write **exciting** files. This is best explained by an example:

```
atoms=read('input.xml', format='exi')
write("outfile.xml", atoms, format="exi")
```

This example code snippet reads the structure from the `input.xml` file and stores the atomic positions, the unit cell, and the species in the *atoms object*. The *format* keyword specifies the format `"exi"` which is short for **exciting** input. The next line writes the structure from the atoms object as **exciting** input file into the file with the name `outfile.xml`.

Current limitations

The calculator supports only total energy and forces; stress and strain is not yet implemented in **exciting**. However, its implementation is in progress.

The keyword arguments in the `Exciting` *constructor* are converted into attributes of the `groundstate` (A.11) element. No check for validity is done there, and not all **exciting** options are accessible in that way.

5.7 exciting@web

The new input and output file formats in the **exciting** code allowed to rethink the concept for a graphical user interface and storage of results. Powerful XML technologies as the XML database eXist-db (Section 4.8.5) and Xforms (Section 4.7), enabled us to develop a very powerful system in very short time.

exciting@web is a user interface to **exciting**, realized as a web service. It consists of an interactive input file editor, an execution environment, and a results database.

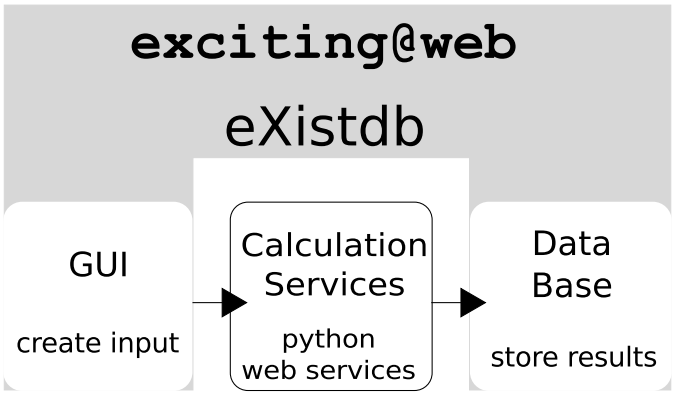

Figure 5.7: **exciting**@web integrates a GUI with the storage and presentation of the data. Calculations can be started and monitored, but the actual number crunching is performed on a separate system which is connected via a web service.

5.7.1 Interactive Input File Editor

An interactive graphical user interface can lower the entrance barrier for new users. A GUI should lay out a clear path to relevant options, eliminate typos as error source, and help to setup the calculation quickly. Therefore we developed an interactive input file editor for the **exciting** input file.

As described in Section 5.3.2, the grammar of the **exciting** input file is defined by an XML Schema, which also contains the documentation of the input parameters. This structure is used for generating the GUI automatically. The immediate advantage is, that the GUI can be kept in sync with the code development and the documentation with little effort. This is realized by a web application that displays an interactive form which reflects the input file structure along with the documentation. Figure 5.9 shows a screen shot of the interactive input file editor. The technology behind involves XSLT and XForms (Section 4.7). Figure 5.8 depicts the diagram for the generation of the user interface. We have created a XSLT template that translates the **exciting**-input schema to XForms code. This XForms code defines the UI (*user interface*) elements to add attributes and elements, display documentation, check the input for validity, and enforce required attributes and elements. The result is an XHTML page with embedded XForms code, which is then translated to HTML and Javascript by the XSLTForms (Section 4.7) template.

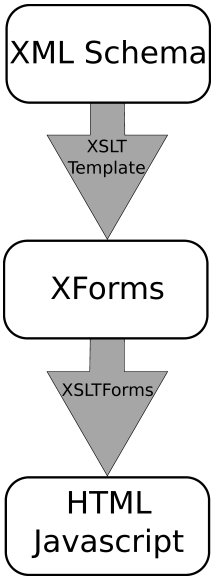

Figure 5.8: The user interface is generated from the XML Schema of the **exciting** input file. The schema is used to generate XForms markup with a custom XSLT style sheet. This XForms is transformed to HTML by the XSLTForms (Section 4.7) style sheet.

Figure 5.9: Interactive input file editor.

5.7.2 Calculation Services

The **exciting**@web server can be connected with a calculation service in order to perform **exciting** calculations on a cluster or remote workstation. This calculation service is a simple HTTP server that takes requests for calculations, initiates them on the HPC system, reports the status, and delivers the results back to the **exciting**@web database.

Remote calculations are started through HTTP requests. The **exciting**@web server opens an HTTP connection to the calculation service if requested by the user. An HTTP POST request with the input file as post data, starts the calculation. The started calculation now has a unique url under which it can deliver information about the convergence and the status of the calculation. The whole API is summarized in Table 5.4.

command	HTTP request
start calculation	POST input.xml content to http://$host:$port/$sha1hash(input.xml)/ **returns:** POST OK
get info.xml	GET http://$host:$port/$sha1hash(input.xml)/ **returns:** content of info.xml
get status	GET http://$host:$port/$sha1hash(input.xml)/status **returns:** XML with job status
get sdtout stderr	GET http://$host:$port/$sha1hash(input.xml)/stdout **returns:** XML files list with stdout and sdterr
get results	GET http://$host:$port/$sha1hash(input.xml)/import **returns:** XML files list with all xml data
delete	GET http://$host:$port/$sha1hash(input.xml)/delete **returns:** error code

Table 5.4: HTTP API for an **exciting** calculation service.

5.7.3 Data Archiving and Presentation

Using XML as data format for results, opens new possibilities for archiving and presenting data. XML is designed to structure any kind of data, and store it in a text file. As it is structured data, it can also be stored in an XML database to allow performing complex queries over large datasets. The XML database can also apply XSLT style sheets on XML documents, which enables us to reuse the output templates (Table 5.1). The data can also be displayed directly in the browser. That way any calculated result has a

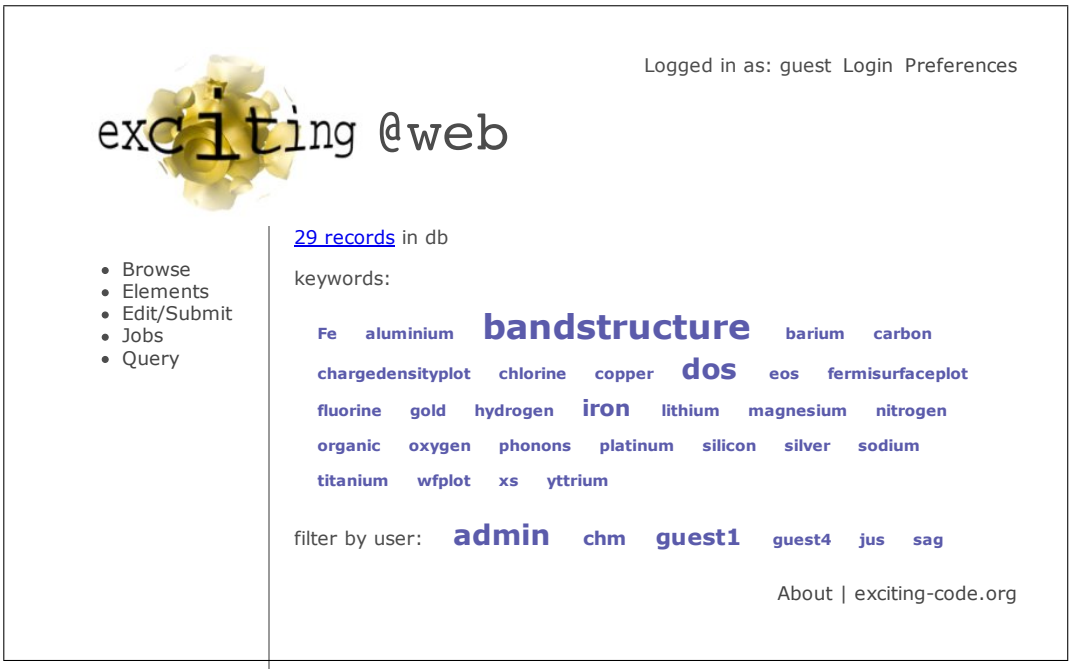

Figure 5.10: Browsing the **exciting** database by keywords.

web page that can be linked. These pages have links for downloading all file formats for which an XSLT transformation is available.

Apart from being of use for training and data management, our database provides a possibility of publishing DFT data on the web, making it searchable with search machines. The latter is a big potential for this kind of computational science, complex calculations and their results can be referenced by a link. Anyone following the link can quickly asses if the results are useful, and can reproduce the result, as all the necessary data is right there.

The database can be used to organize the data in many possible ways. One is the keyword browser (Fig. 5.10). The keyword browser allows to select only the entries that match a set of keywords. The keywords are partly generated from the input file and custom keywords.

When one entry is selected, the input file with a 3d model of the structure is displayed (Fig. 5.11). The 3d model is interactive and can be zoomed and rotated by the user. This hopefully helps to decide quickly if the structure in question is useful and interesting enough for exploring it further. A couple of properties and graphs do already have a defined view to visualize the data: the density of states (Fig. 5.12) and the band structure (Fig. 5.13). The view for the *equation of state* (EOS) is a special case. The EOS is obtained from the energy-vs.-volume curve and can be used to determine the groundstate

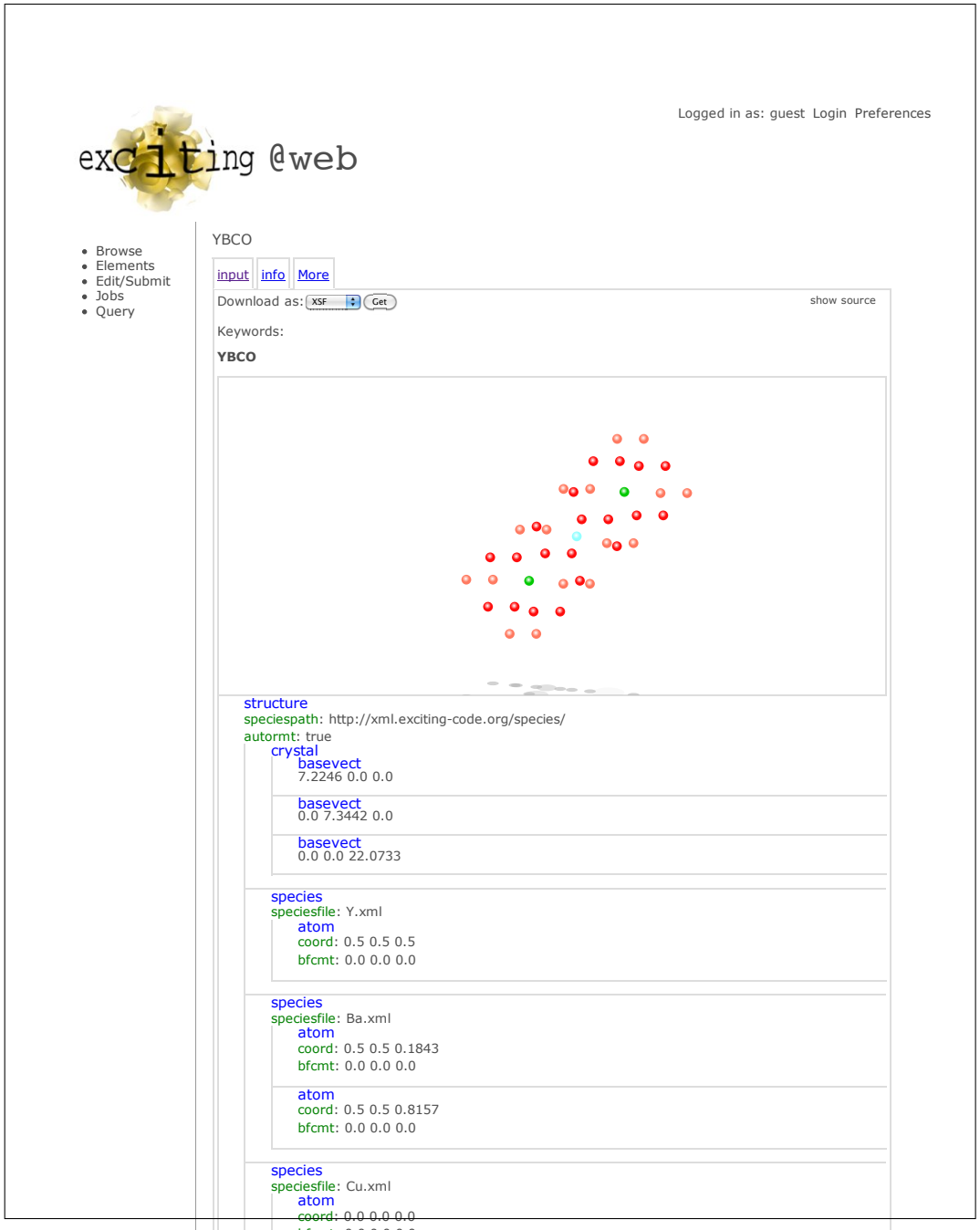

Figure 5.11: Display screen for the input file with 3d visualization of the structure.

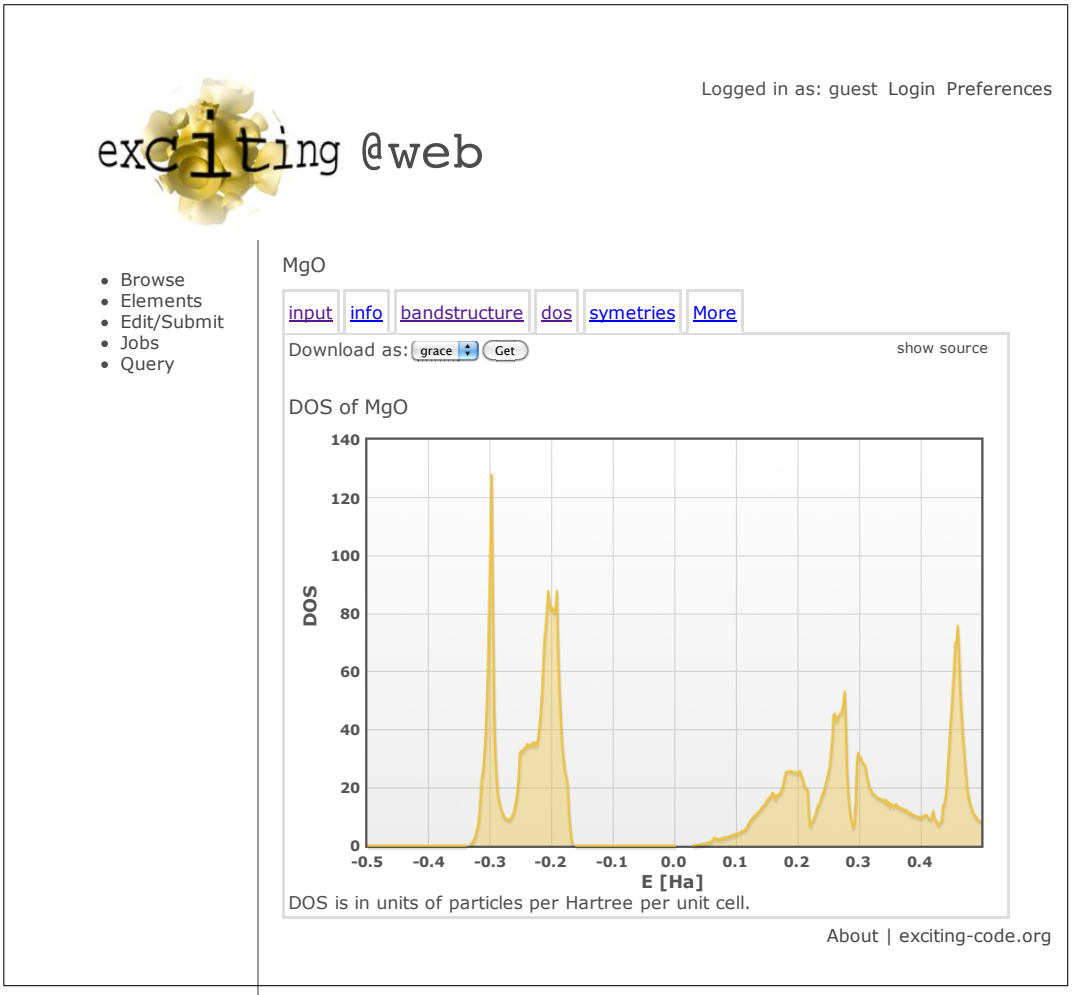

Figure 5.12: DOS-visualization screen.

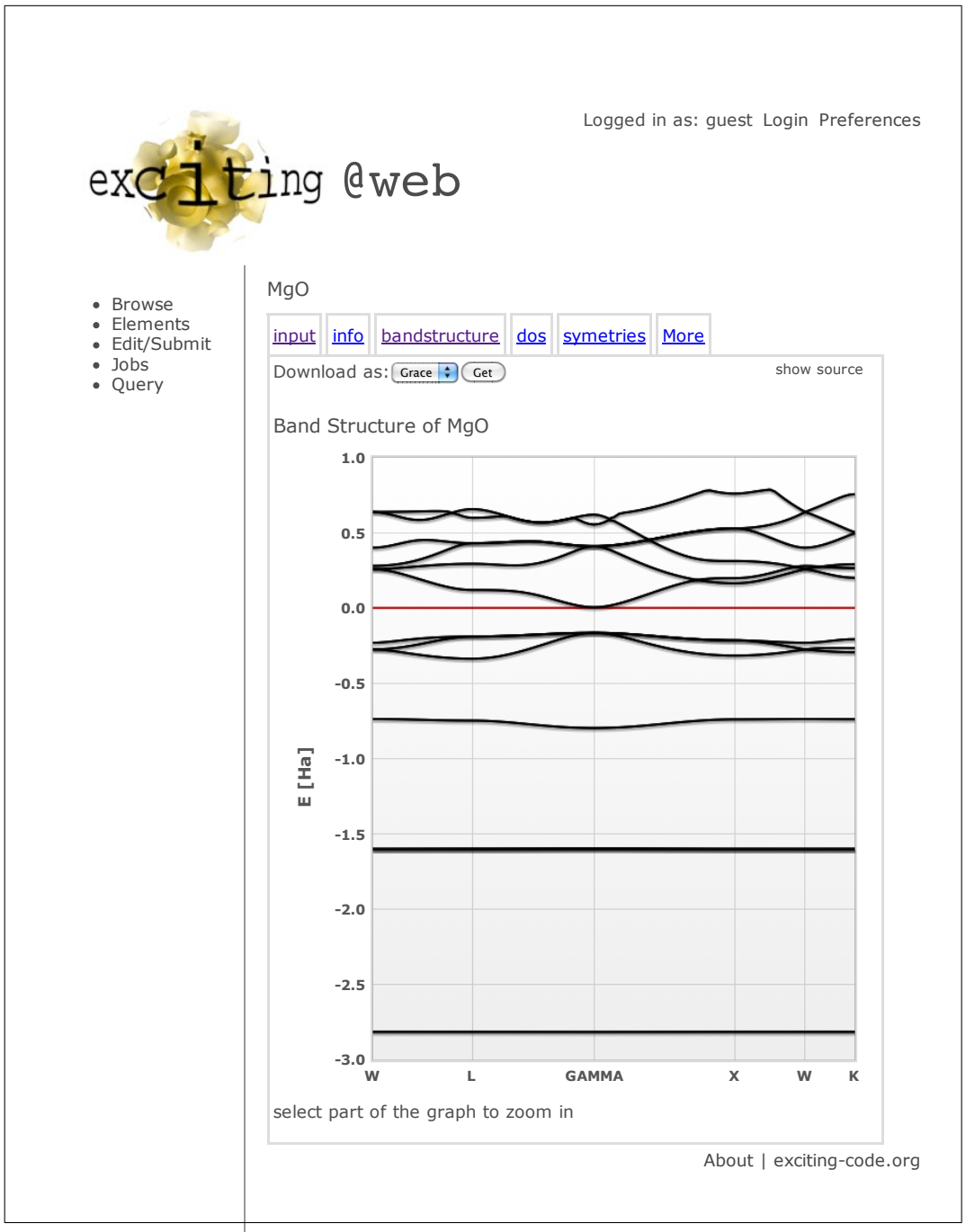

Figure 5.13: Interactive band-structure visualization. The graph can be interactively zoomed in to show details.

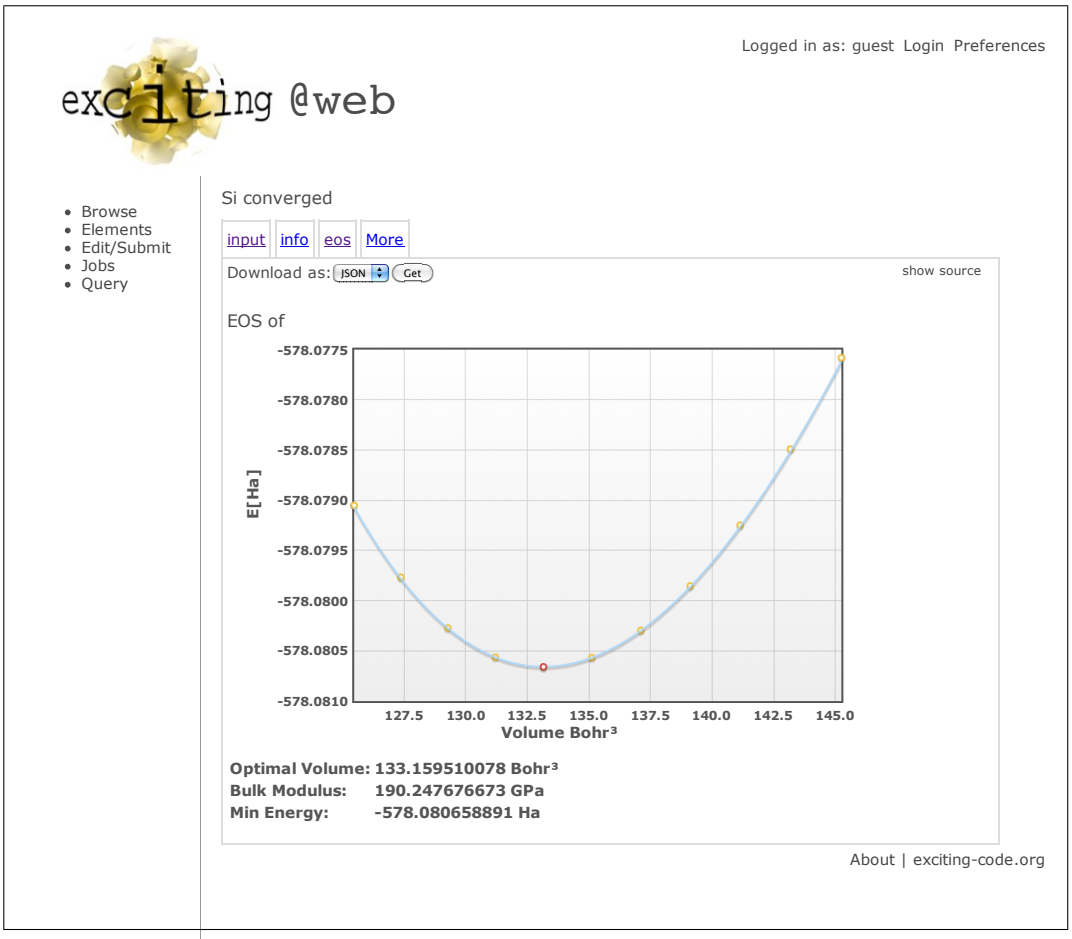

Figure 5.14: Screen shot for equation-of-state data.

volume. To get this curve, a whole set of calculations for different volumes is required. We have defined a data format for EOS data, that can be stored with the results of the groundstate volume. This data format also has its own visualization (Fig. 5.14).

If there are templates available to convert the stored files into other file formats, the user can select the desired format and download it do the client computer for visualization or further processing.

Part III

Software Development

Chapter 6

Scientific Software Development

Developing complex software systems is a difficult problem. Today it is an engineering discipline on its own with many abstract concepts and tools. When the first programmable computers where available, scientists and technicians did not have these concepts. The software was also relatively simple and maybe written by one person only. With time these simple programs where extended and were worked on in teams until one very critical observation was made by Fred Brooks [10]: "adding manpower to a late software project makes it later". Software development did not scale to teams and sometimes productivity deteriorated with the group size. This is described famously in the book: "The Mythical Man-Month" by Fred Brooks [10].

The software crisis was a situation in the 1960's where many software projects stagnated or failed due to organizational difficulties. Since then, concepts to prevent this crisis are the driving force in software engineering.

Science got more and more dependent on computational methods, largely without adopting the necessary structures and procedures to maintain and preserve the knowledge that is encoded in program source code. The main challenge is the transition from a one-developer project to a collaborative one. It is the only way to ensure that there is always a minimum number of active developers with the knowledge to change and maintain the code. If this is realized, it is much less likely that the work of collaborators is lost. The transition to a collaborative project is, however, very difficult, because it requires many extra efforts, which initially can slow down the project.

There are a couple of formal requirements to enable a collaborative software development. It requires good documentation, formalized processes, and an adequate tool chain. It also requires modularity and useful abstractions in the code. There is a lot of research and books on this matter. Mostly,

it is not easily transfered to the needs of academic scientific programming which often is forced to use legacy codes and tools. The main source for the concepts presented here is from the book "The Pragmatic Programmer"[28]. This chapter goes on about presenting concepts from that book.

6.1 Source-Code Management

Version control or *source-code management* (SCM) has to solve two problems: to recorde changes and manage collaboration. SCM is the programmers equivalent to the lab book. It allows to recover the last working version after a failed experiment, it records the history of the ideas, and forces the programmer to annotate the changes. SCM software also provides a way how to simultaneously work on the same source tree by providing means to merge forked development branches.

Another important part of using SCM, is to verify which version of the code produced what results. This is important for reproducibility of numerical experiments. A unique version number allows to identify the exact version in a central database. In a distributed SCM cryptographic hashes are used to uniquely identify a version. In collaborative environments, SCM doesn't replace quality assurance or code reviews, but it is a necessary step to start these processes.

6.2 Git, a Fast Distributed Source Code Management Tool

`git` [25, 14] is a software for distributed SCM using SHA1 [49] hashes as version identifiers. `git` is distributed, in the sense that each local instance of the repository has the whole history of all changes stored in an highly efficient compressed data structure, and it does not require a central authoritative repository. `git` organizes the source code in terms of *commits*, *branches*, and *tags*.

`git` manages a directory of files, and subdirectories, a *tree* of source files. Commits are specific stages of a source tree. They are snapshots of the source code with a comment and one or multiple ancestors (parents). As each commit has ancestors, they build a chain constituting the commit history of the code. A commit is identified by a hash, a large binary number, which is a cryptographic checksum over the commit message, the tree, and the parent hashes. Like this, the hash is not only an identifier but also a signature, allowing to verify the integrity of the history and the source code. The hash

is such a large number that it is virtually impossible that two different files have the same hash. This is necessary for distributed architecture to work, because there is no central authority which could ensure the uniqueness of the identifiers.

Commits can be associated with special names known as tags. Tags are useful to identify special versions such as release versions. One `git` repository can have multiple branches of commits. Branches are forks from the main chain of commits in order to work out a change without compromising the main (master) branch. Branching and merging is so fast and comfortable that it is actually good practice to make feature branches to work on new features.

`git` supports several protocols to exchange code. Source code can be shared over an ssh server, over web servers, email, and the local file system. An existing repository is cloned to create a local copy. As the term clone suggests, the local repository has all the same information and authority like the original one. `git` provides means to *fetch*, *pull,* and *push* changes among repositories over various protocols.

As `git` gained popularity quickly, there emerged great services to simplify sharing source code. One of them is *github* [20], a service which very much simplifies exchanging and merging source code.

6.3 Reproducibility

Reproducibility is one of the main principles of science. In software development it usually is fulfilled as one expects the same results if the same program is run with the same inputs. This appears to be trivial. Unfortunately, it gets fairly complex as a program almost never is static. To reproduce published results, one requires the same version of the code that was applied, and complete information about the features and parameters used. Using open source code ensures transparency and access to the means to reproduce the results. The commit hashes of `git` can be used to identify a specific version of the source code which cannot be faked. The hash effectively is a cryptographic signature that allows to verify the integrity of the source code. If someone claims that a specific result was obtained with a specific version of the code identified by its hash, anyone else can verify it reliably.

6.4 Testing

Every program has to be tested in order to verify that it does what it is intended to do. This is obvious, and every scientist working with software does testing in one or the other way. Verification of the algorithm during development is only one aspect of testing. Testing should enable to discover regressions and to simplify refactoring. These aspects get more important, the more mature the project becomes.

Whenever a new feature is implemented one would repeatedly run the program until the output satisfies the expectations. But how are this expectations defined in scientific programming? It is clear that what we really want are specifications that can easily be decided to be passed or failed. Finding these specifications for the results, can be difficult because often the only way to produce the results is the algorithm itself. In physics, we have experimental data we want to model and can use this as reference to decide if the output is reasonable. This does, however, not really assure that the program does exactly what it is intended to do, namely implementing the theoretical model of a physical process.

One solution is, of course, using a reference model of which the solution is known explicitly. But what if this is not available? Any complex algorithm can be broken down to smaller parts which can be verified more easily. This testing of all the units has become an established method called unit testing. The goal of unit testing is writing simple tests against specifications on function or procedure level. The idea is: When a certain code coverage is reached, one has good reasoning that it does what it is intended to do, and bugs are found fast. None of this is proving the correctness but it is much better than just hoping that the code is correct.

The next point is that tests are an integral part of the software. They have their place directly in the source tree of the project. It is a waste to throw away a test environment, once the feature was verified to work correctly. If tests get compiled to a test suite that can be executed automatically and report if the tests passed or failed, one gets a lot of extra value out of testing. It is almost as important to know if a feature *still* works as if it worked once.

An automatic test suite improves the development at many points. It can provide a clear and transparent criterion to accept patches from collaborators. It simplifies code refactoring because it allows to verify each stage more easily, which ultimately leads to better code and more code reuse. And, most important, it allows to check faster if changes break some feature.

The next evolutionary step in that reasoning is test-driven development. Test-driven development means that one would write the test before writing any code that actually implements the feature. In other words: write a test,

bring it to fail, write code to make the test pass.

The benefits of this development style get naturally evident when working on a more complex subsystem which requires a lot of work until one reaches the first working version.

6.5 Modularity

There is a danger of growing complexity in program code: It becomes *spaghetti code*. Spaghetti code means that it is not predictable how changes in one part of the code affect any other part of the program. This side effects increase debugging time and learning time until, in the worst case, stagnation is reached. The answer from software engineering is: modularity. There has been written a fair amount of books on this topic. Most books however, directly address object-oriented programming. As FORTRAN 90 doesn't have objects or classes, the terminology is hard to apply. The core concepts, however, apply to any language. A good reference with a more abstract view in this field is [28]. Therefore a short review of what modularity may mean for scientific software written in FORTRAN will be given below. What are modules? Modules are distinct parts of the program that are orthogonal and coupled minimally to each other by an API.

6.5.1 Orthogonality

Modules should be orthogonal in the sense of orthogonal vectors, as movement along one vector is not observable in the projection on the other. In terms of programming this means that changes in one module should not require changes anywhere else.

This is the reason why having too many global variables leads to very difficult situations. Every single global variable used by more than one module is a coupling factor that destroys orthogonality. The concept to battle this condition is called information hiding. Information hiding uses language features to limit the scope of variables, to enforce better programming style, and to ensure orthogonality. It is, however, necessary that modules exchange information. For this purpose, modules have an API to realize the coupling between them.

Technically one has to ensure that other modules will only access the API and cannot access other functions or data. Best is to use language features to ensure that. Then the compiler will complain at compile time when data or procedures other than those specified in the API are accessed by a different module. This is, basically, what object-oriented languages do.

Those languages have a lot of features to control the scope of attributes and methods such that the author of a module can force other programmers to stick with the methods defined as public API.

6.5.2 Application Programming Interface

The API of a module or a library is a collection of functions and data types which are used to exchange information and call functionality from that module. This is what other programmers need to learn when they need the functionality of the module. Therefore it should be designed to be easy to learn and use. An API is designed for other humans, not for the computer. Also, the API should not change much once the module is released. A good API has the following desirable features:

1. The API should include only the absolutely necessary features. Firstly, it is less to learn, and secondly, it is always easier to add things than to remove something, some other code already relies on.

2. The names of functions should be expressive and really describe what they do. Names should be chosen with utmost care.

3. If possible, follow a scheme for the names and keep the order of parameters consistent.

4. Ideally, there should not be more than three parameters per function call.

Long argument lists are a big error source, and they make code difficult to read. For sure, there are not many functions, that can take only three values as arguments, that are interesting for any kind of advanced API. But arguments don't need to be simple types. FORTRAN allows to define derived types with any desired complexity. With the help of derived types, interfaces can be simplified a lot. Long lists of parameters can be encapsulated in a derive type. The argument is much shorter and the compiler can check the correctness of the derived types at compile time. With these tricks, code can be as readable as sentences in natural language.

Derived types are the objects in FORTRAN. They cannot contain methods in FORTRAN 90, but they do encapsulate data. Using derived types, the interface of API subroutines may be composed of a configuration object, a data object and a short list of parameters. The configuration object would contain parameters that remain the same for a series of API calls and the data object contains the data that is manipulated by the subroutine.

The API could include a function to create a configuration object and set the values to reasonable defaults. The programmer must only set the values that differ from the defaults. Then the configuration object can be used in a couple of function calls before it is destroyed by a deallocation call.

6.5.3 How to Write Modular Code in FORTRAN

Modularity is not about throwing away FORTRAN code and replacing it with C++. It is about conquering complexity with abstraction. In FORTRAN 90, the only language features to create abstraction are subroutines, modules, and derived data types.

Subroutines are blocks of code that can be reused with different input data. They represent the basic concept to avoid code duplication and partition the code into functional units.

Modules can realize name spaces that can be made available through the *use* statement. They may contain a collection of subroutines, constants, and data-type definitions. It is, however, not a good idea to define variables in the module because they would create the same problems as global variables.

Derived data types are rarely used by beginners but have the largest potential for abstraction. They allow to encapsulate logically connected data in a data structure. A data structure can have multiple instances, which is an important contrast to modules. Together with pointers, data structures can be structured hierarchically, building a tree that represents the abstract structure of the data. In their function of encapsulating connected data, data structures help to make interfaces cleaner. The interface of a procedure consists of the arguments and the return values of the subroutines. By using derived types the argument lists get much shorter and easier to use.

Derived types are the closest thing to objects FORTRAN 90 can offer. A derived type in FORTRAN 90 cannot contain methods and is not extensible by inheritance which are the two most important features of object-oriented languages. Without these features, it is still possible to mimic some of the properties of objects in FORTRAN 90. It is, for example, a common practice to have a set of subroutines that all have a particular derived type as their first argument. These subroutines may be seen as *methods* of the object as it is represented by this derived type. The API of the FoX (See Section 4.5) library may serve as a great example for this. FORTRAN 2003 adds language features to actually use types as objects (See section 6.6).

It is very important to avoid spaghetti code, the situation where any change in the code may affect any other part. The barrier where no single person understands the whole code anymore is reached quickly. And when it is reached, new developments will take longer and longer and the quality will

degrade until a simplification and modularization can be realized. Spaghetti code can be avoided in any language but it requires to make smart use of the available abstractions. In FORTRAN, this means that data has to be moved from modules into data structures, and procedures must have clean interfaces and not manipulate global data randomly.

6.6 Newer FORTRAN Standards

FORTRAN 90 was a large step from FORTRAN 77, and is probably the FORTRAN version the most commonly used. There are, however, newer standards: FORTRAN 2003 and FORTRAN 2008.

It is hardly known that, to date, most current compilers support big parts of FORTRAN 2003. `gfortran` from the gnu compiler collection and `ifort` from Intel, for example, both support *procedure pointers* bound by name to a type. This is the basis for object-oriented programming as it effectively implements objects and methods. FORTRAN 2003 even allows to extend derived types and realizes inheritance and polymorphism. These features are relatively new, but definitely in a state where one should begin to use them.

FORTRAN 2008 is a rather small addition to FORTRAN 2003. It mainly adds functionality for parallel programming. The compiler support is currently not as advanced as for FORTRAN 2003. This means it cannot be used for real software projects yet.

6.7 Refactoring

Refactoring is the process of reorganizing software without changing the functionality. The goal of refactoring is obtaining better readable and extensible code by regrouping program parts to modules, changing interfaces and datatypes, or mere renaming of specifiers. A good overview about this is found in Ref. [32].

Getting the software architecture right from the start, such that it never must be changed, does not work in practice. Refactoring is a necessary part of any software project. Sometimes, the refactoring has to amount to a major software refurbishment. A condition for successful refactoring is some kind of test system (Section 6.4) in order to ensure that the functionality does not change during refactoring.

The most relevant types of refactoring are:

Renaming: Source code is read by many people, so having expressive names for identifiers is the easiest and most effective way do reach better

readability.

Removing code in comments: There is no need to keep old code in the source. That is the job of the version-control software.

Redesigning data structures: Data structures are the most important tool to make code better readable.

Redesigning interfaces: Subroutine interfaces with many parameters are hard to use.

Splitting Subprograms: Too long subroutines which operate on too many different data types should be split up into subprograms with a clear scope and task.

It is important to note that these processes can be automated to a significant degree. There is some refactoring support included in IDEs (Integrated Development Environments). As refactoring mostly is search and replace in text files, any advanced text manipulation tool is of great help and can remove some of the error sources and risks.

Chapter 7

The Development Process in `exciting`

We took a great effort to streamline the collaboration in the **`exciting`** project. Collaboration requires a group with common motivation and trust. But there are a few technical prerequisites and processes that are best praxis and address systematic obstacles.

The central body of the collaboration is the source code. Collaboration means, that multiple people edit the code at the same time. All of them expect the code to be in a defined state when they compile and test it. So everyone has a copy of the code from which he or she progresses with a chain of changes, until his, her goal is reached. This chain of changes is called a *branch*. To organize and merge the branches some kind of source code management is absolutely necessary.

`exciting` is an open-source code. The expected advantage of open source is, that openness and transparency help to create higher quality code, facilitate collaboration, and to prevent knowledge from getting lost or locked in. In order to benefit from these theoretical advantages, it is not enough to merely attach the GPL license to the code, and put it on a website. Rather, one needs to develop a whole community process, that can govern an open-source project.

7.1 Simplify Merging

Merging should be fast and easy. When merging is not a problem, it can be done more often and earlier. Collaborators can be confident that their changes will enter the official version and stay usable. This is realized by a couple of elements we newly introduced with the current version. They con-

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="./report.xsl" type="text/xsl"?>
<report>
  <test>
    <name>EQATOMS.OUT</name>
    <description>EQATOMS.OUT      EQATOMS.diff      </description>
    <directory>test01</directory>
    <status>passed</status>
  </test>
 ...
</report>
```

Listing 7.1: Example for a test report. The test report file is encoded in XML.

sist of (1) modern distributed source code management, (2) dynamic build system, (3) automated tests, (4) formal input-file description. This improvements do play together to allow a transparent and save development process. The distributed source code management solves the technical problem of merging two versions of the code in a safe and convenient way (see Section 6.1).

The dynamic build system eliminates the manual editing of `make` files and encourages the use of modular programing. The dependency graph of the modules is computed automatically within seconds to create a make file with all the dependencies set correctly. The program that does this is a Perl script called `mkmf` [52, 7], which is GPL licensed software. By eliminating the need of updating the makefiles completely, merge conflicts in the make files have disappeared in **`exciting`**.

7.2 Test System in `exciting`

The test subsystem is located in the `test` directory in the source tree. In the test directory, there are multiple directories for different tests. The list of tests is controlled by a makefile. By typing "*make test*" the test binaries are compiled, the test runs are executed, and the report scripts that evaluate the results are called. For test reports we use XML files. The reason for this is that there are tools which are simple, powerful, and everywhere available, to process these files. The file formats can be extended with new fields without breaking any of the scripts.

A test report looks like Listing 7.1.

7.2.1 How to Write a Unit Test

Unit tests are test procedures written in the same framework as the code itself. The test routine has access to all the runtime data structures and functions. Like this, small units are accessible for testing. Tests should be as independent as possible even if this is sometimes difficult to archive. A test is passed if the output of a program unit, e.g., a subroutine, matches with the predefined reference data.

exciting includes a simple test-reporting module. To report a test result, one uses the `testreport()` procedure. It takes a logical value for the test status as argument, `.true.` for passed and `.false.` for failed. The procedure writes a report to the report file by using the following variables from the module `modreport`:

```
1 testunitname="LINENGY.OUT"
2 inputf="LINENGY.OUT"
3 outputf="LINENGY.diff"
4 call testreport(passed)
```

7.2.2 How to Write a Black-Box Test

A black box test uses the **exciting** program. Such a test consists of a set of inputs and reference data. The assertion routine can only access outputs written to disk by the **exciting** program.

In order to facilitate the creation of such tests, there is a reporting framework written in Perl available. There are also functions for comparing numerical values in files, to check against the reference data. An example is given in Listing 7.2. This Perl code uses the `Test` module that is included with the **exciting** test system to create a test report about the ARPACK solver.

7.2.3 Generating the Test Summary

After all the tests are executed, a number of test reports are written into the test directories. These reports are summarized by an XSLT template that creates an HTML report including statistics. The statistics tell how many tests were executed, and how many of them passed or failed. The numbers are visualized in a graph like in Figure 7.1.

```
#!Perl (-)
use lib "../perl/";
use lib "../perl/lib/";
use XML::Simple;
use XML::Writer;
use IO::File;
use Test;
$writer= Test::initreport("report.xml");
open INFO, "runarp/INFO.OUT";
$status=failed;
while(<INFO>)
    {
    if (m/\| EXCITING .+stopped/){
    $status="passed";
    }
}
Test::writetestreport({
               directory=>"test02/runarp",
               name=>"arpack run",
               description=>"The test run using arpack finished without
                   errors",
               status=>$status
               }, $writer);

Test::closereport($writer);
```

Listing 7.2: Perl test report interface.

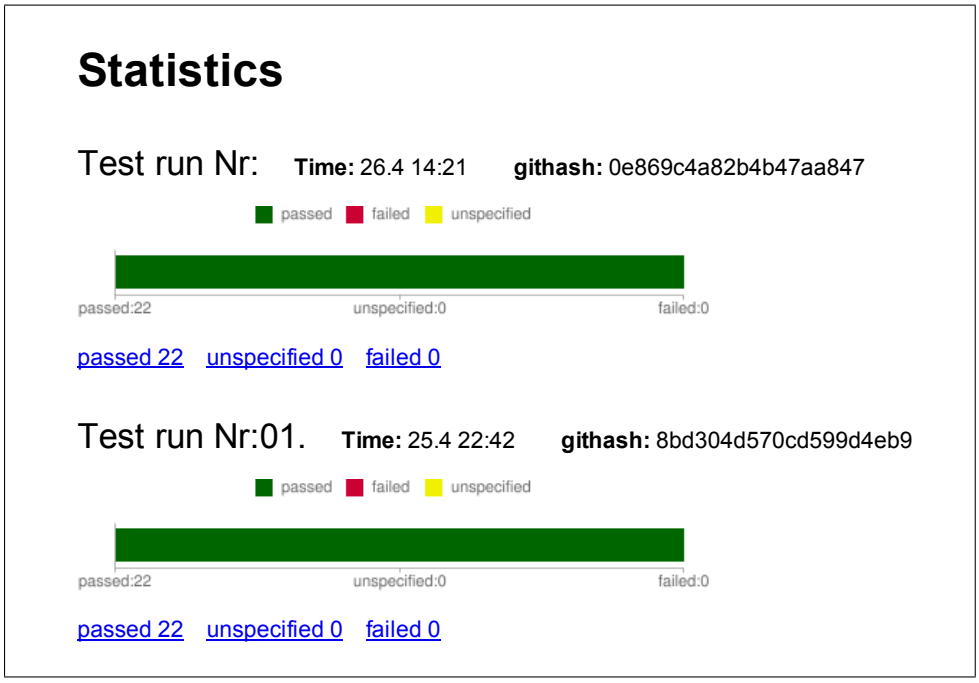

Figure 7.1: Visualization of the Test summary.

7.3 Examples of Modularity in the `exciting` Code

The new input file data structure may serve as an example for the modularity of **`exciting`** (Section 5.3). Also the eigensystem interface, that hides the data in abstract structures, may serve as an example (Section 3.4).

This kind of refactorings will constantly be necessary. It should be good practice to identify hot spots in the code, data that is touched by any new developments and used by many procedures, and find a way to organize the data, simplify interfaces, and get rid of modules with a long list of variable definitions.

7.4 Refactoring of the `exciting` Code

The **`exciting`** code was subject to one significant refactoring: The new input system. The missing separation of the input parameters from the variables was problematic. Initial were in the input parameters written which lead to an uncertainty what the values actually were. Refactoring the

input system, by formalizing the input definition and using a hierarchical data structure made the distinction very clear. The automatically generated parser code introduces a new level of abstraction. The programmer does not need to know how the parsing works in detail. In order to add or change input definitions the programmer has to configure them in the XML Schema, which provides a general model on what inputs may be and how they should be grouped. This step from handwritten read statements to a configurable parser leads to having better and more consistent code.

7.5 Issue Tracking

One more thing that needs organization when the development is shared by multiple people, is tracking and organizing of bugs and feature requests, or more generally tasks or issues. Issue tracking should provide accountability to where and when problems where solved and who has the responsibility for open issues. There is a lot of software available to manage such *issues*, *tasks*, *tickets*, or *bugs*. For the **exciting** project, the implementation was not so much a technical as a cultural challenge. Issue tracking is a complex problem and so is most issue tracking software. The barrier to use it on a regular basis is higher than for SCM where, for example, the immediate benefit is much more apparent. Especially, end users find it repelling to fill out the mask of a complex issue-tracking system. However, it is the end-users feedback which must be tracked and held accountable for. The practical solution involves to use a web-forum software which has a low barrier for users. It allows to track the problem and possible answers and fixes. The project may eventually outgrow this situation but for the time being the forum is an acceptable solution.

7.6 Outlook for the exciting Development Process

As a result of all of these efforts described above we now have a considerable team of developers which work simultaneously on the code. The issues of fighting with merging the code and keeping the reference documentation up to date mainly vanished. So far, we hardly have contributers outside the local research group. There is quite some way to go to improve transparency and lower the barriers of entry, while ensuring a reliable level of quality control.

For a larger software project to persist over time, one has to accept the necessity to restructure the code regularly. The **exciting** code has many

areas that need better structure in order to reach a higher modularity, and lower the barriers of entry. These areas should be addressed one by one, prioritized by their relevance for ongoing projects.

Chapter 8

Conclusions

During the work on this thesis, the **exciting** code went trough a significant transformation. This transformation may be characterized by going from a small, single-scientist project to a collaborative software project.

One important aspect of the work is the evaluation of the status quo at the beginning. The performance measurements suggested bottle-necks that where subsequently addressed. A basic parallelization scheme was implemented, and more suitable algorithms for the core eigen-solver where evaluated and implemented.

The other aspect addressed in this work was the over-all architecture of the **exciting** program. Some of the structures were not suitable for further development. So the whole input-output paradigm was rethought, and the program refactored to accommodate new ideas. The new methods try to appreciate best practices for coding as they evolved in the computer sciences.

The **exciting**@web platform marks an innovative approach to a user interface for the **exciting** program. The advantages of XML technology where exploited to realize not only an interactive input file editor with a job-submission back end, but also a result database with many visualizations and data transformations. These results database opens new opportunities for data presentation, archiving, and data-mining. It is a step in the direction to make the scientific practice ready for the "internet age".

It is important to acknowledge that theoretical physicists are not the only ones to write software. It is mandatory to look sideways to adopt best practices for working with code and computers. If the insights of modern software engineering are turned down, the risk of wasting resources and loosing knowledge gets too high.

This work goes through many aspects of computational methods that came up through a mayor restructuring of the **exciting** code. The project became more suitable for future challenges in physics and scientific software

development.

List of Figures

List of Tables

Listings

Bibliography

[1] C Ambrosch-Draxl. Augmented planewave methods. *Phys. Scr.*, T109:48–53, 2004.

[2] C. Ambrosch-Draxl, S. Sagmeister, Ch. Meisenbichler, and J. Spitaler. Exciting code. [online] http://exciting-code.org, 2009.

[3] O. K. Andersen. Linear methods in band theory. *Phys. Rev. B*, 12:3060, 1975.

[4] E Anderson, Z Bai, C Bischof, S Blackford, J Demmel, J Dongarra, J Du Croz, A Greenbaum, S Hammarling, A McKenney, and et al. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1999.

[5] W E Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quart Appl Math*, 9(17):17–29, 1951.

[6] S. R. Bahn and K. W. Jacobsen. An object-oriented scripting interface to a legacy electronic structure code. *Comput. Sci. Eng.*, 4(3):56–66, MAY-JUN 2002.

[7] V. Balaji. mkmf: f90/cpp dependency analysis and makefile generation. [online] http://www.gfdl.noaa.gov/~vb/mkmf.html, 1 2009.

[8] L S Blackford, J Choi, A Cleary, E D'Azevedo, J Demmel, I Dhillon, J Dongarra, S Hammarling, G Henry, A Petitet, et al., , and R C Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1997.

[9] Peter Blaha, Karleinz Schwarz, and Joachim Luitz. WIEN2k. (Release 97.8), April 1997. [Improved and updated Unix version of the original copyright WIEN code, which was published by P. Blaha, K. Schwarz, P. Sorantin and S. B. Trickey, Comput. Phys. Commun.9, 399 (1990)].

[10] Frederick P Brooks JR. *The mythical man-month*, volume 18. Addison-Wesley, 1995.

[11] C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Math. Comp*, 19:577–593, 1965.

[12] Kieron Burke and friends. The abc of dft. [online], 2007.

[13] Alberto Castro, Heiko Appel, Micael Oliveira, Carlo A Rozzi, Xavier Andrade, Florian Lorenzen, M A L Marques, E K U Gross, and Angel Rubio. Octopus: a tool for the application of time-dependent density functional theory. *Physica Status Solidi B*, 243(11):2465–2488, 2006.

[14] Scott Chacon. *Git Internals*. PeepCode, 2008.

[15] Couthures. Xsltforms. [online] http://www.agencexml.com/xsltforms.htm, 2009.

[16] E R Davidson. Iterative calculation of a few of lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *Journal of Computational Physics*, 17(1):87–94, 1975.

[17] J. K. Dewhurst, S. Sharma, and C. Ambrosch-Draxl. Code development under the RT network EXCITING funded by the EU, contract HPRN-CT-2002-00317, 2005.

[18] Apache Software Foundation. Xalan. [online] http://xml.apache.org/xalan-j/.

[19] Luigi Genovese, Alexey Neelov, Stefan Goedecker, Thierry Deutsch, Seyed Alireza Ghasemi, Alexander Willand, Damien Caliste, Oded Zilberberg, Mark Rayson, Anders Bergman, and et al. Daubechies wavelets as a basis set for density functional pseudopotential calculations. *The Journal of chemical physics*, 129(1):014109, 2008.

[20] GitHub. Github - social coding. [online] http://github.org, 2010.

[21] G H Golub and Charles F Van Loan. *Matrix Computations*, volume 10. The Johns Hopkins University Press, 1996.

[22] Lars Grasedyck and Wolfgang Hackbusch. Construction and arithmetics of h-matrices. *Computing*, 70(4):295–334, 2003.

[23] DOM Interest Group. Document Object Model (DOM). [online] http://www.w3.org/DOM/, 2009.

[24] F Gygi. Architecture of qbox: a scalable first-principles molecular dynamics code. *IBM Journal of Research and Development*, 52(1):137–144, 2008.

[25] Junio C Hamano, Shawn O. Pearce, Linus Torvalds, Johannes Schindelin, Eric Wong, Jakub Narebski, Jeff King, Nicolas Pitre, Paul Mackerras, Simon Hausmann, Johannes Sixt, Petr Baudis, Christian Couder, Alex Riesen, René Scharfe, J. Bruce Fields, Miklos Vajna, Daniel Barkalow, Kay Sievers, Pierre Habouzit, Brandon Casey, Alexandre Julliard, Thomas Rast, Alexander Gavrilov, Frank Lichtenheld, Fredrik Kuivinen, Martin Langhoff, Jonas Fonseca, Gerrit Pape, Nanako Shiraishi, Matthias Lederhofer, Steffen Prohaska, Jay Soffian, Lars Hjemli, Stephan Beyer, Nick Hengeveld, Martin Koegler, and Matthias Urlichs. Git – Fast Version Control System. [online] http://git-scm.com, 08 2009.

[26] P C Hansen. *Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion*, volume 94. SIAM, 1998.

[27] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136:B864, 1964.

[28] A Hunt and D Thomas. *The Pragmatic Programmer*. Addison Wesley, 2000.

[29] J Kavanagh and W Hall. *Grand Challenges in Computing Research 2008*. UK Computing Research Committee, 2008.

[30] Michael Kay. *XSLT and XPath Optimization*. 2004.

[31] W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Phys. Rev.*, 140(4A):A1133–A1138, Nov 1965.

[32] S. Kübeck. *Software-sanierung: Weiterentwicklung, Testen und Refatoring bestehender Software*. mitp-Verlag, 2009.

[33] R. Lehoucq, D. Sorensen, C. Yang, R B Lehoucq, D C Sorensen, and C Yang. *Arpack users' guide: Solution of large scale eigenvalue problems with implicitly restarted Arnoldi methods*. SIAM, 1998.

[34] Lauri Lehtovaara, Ville Havu, and Martti Puska. All-electron time-dependent density functional theory with finite elements: time-propagation approach. *The Journal of chemical physics*, 135(15):154104, 2011.

[35] L. D. Marks and D. R. Luke. Robust mixing for ab initio quantum mechanical calculations. *Phys. Rev. B*, 78(7):075114–+, August 2008.

[36] Wolfgang M. Meier. exist-db open source native xml database. [online] http://exist-db.org/, 5 2010.

[37] U.S. National Library of Medicine" "National Center for Biotechnology Information. Gene. [online] http://www.ncbi.nlm.nih.gov/gene, 2011.

[38] J. P. Perdew, K. Burke, and M. Ernzerhof. Generalized Gradient Approximation Made Simple. *Phys. Rev. Lett.*, 77:3865, 1996.

[39] J. P. Perdew and Y. Wang. Accurate and simple density functional for the electronic exchange energy: Generalized gradient approximation. *Phys. Rev. B*, 33:8800, 1986.

[40] M.J. Rayson and P.R. Briddon. Rapid iterative method for electronic-structure eigenproblems using localised basis functions. *Comput. Phys. Commun.*, 178(2):128–134, January 2008.

[41] Russ Rew, Glenn Davis, Steve Emmerson, Harvey Davies, and Ed Hartne. The netcdf users guide. *Atmospheric Research*, (March), 2008.

[42] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Halstead Press, New York, 1992.

[43] L. J. Sham. Exchange and correlation in density-functional theory. *Phys. Rev. B*, 32:3876, 1985.

[44] D. Singh. Ground–state properties of lanthanum: Treatment of extended–core states. *Phys. Rev. B*, 43(8):6388–6392, March 1991.

[45] D. Singh and H. Krakauer. H-point phonon in molybdenum: Superlinearized augmented-plane-wave calculations. *Phys. Rev. B*, 43(2):1441–1445, Jan 1991.

[46] E. Sjöstedt, L. Nordström, and D. J. Singh. An alternative way of linearizing the augmented plane-wave method. *Sol. Stat. Comm.*, 114:15, 2000.

[47] J. C. Slater. Wave functions in a periodic potential. *Phys. Rev.*, 51:846, 1937.

[48] C. M. Sperberg-McQueen and Henry Thompson. W3C XML Schema. [online] http://www.w3.org/XML/Schema, 4 2000.

[49] Processing Standards. Secure hash standard (shs). *Processing*, (October), 2008.

[50] Daniel Veillard. The xml c parser and toolkit of gnome. [online] http://xmlsoft.org/.

[51] W3C. Xforms 1.1. [online] http://www.w3.org/TR/xforms11/, 10 2009.

[52] Kim Walden. Automatic generation of make dependencies. *Softw., Pract. Exper.*, 14(6):575–585, 1984.

[53] Toby White. FoX. [online] http://www1.gly.bris.ac.uk/ walker/FoX/, 7 2011.

[54] Toby O H White, Peter Murray-Rust, Phil A Couch, Rik P Tyer, Richard P Bruin, Ilian T Todorov, Dan J Wilson, Martin T Dove, Kat F Austen, and Warrington Wa. Application and uses of cml within the eminerals project. *Proceedings of the UK eScience All Hands Meeting 2006*, pages 606–613, 2006.

[55] D M Wood and A Zunger. A new method for diagonalising large matrices. *J. Phys. A.: Math. Gen.*, 18(9):1343–1359, 1985.

Part IV

Appendix

Appendix A

exciting input reference

exciting developers team
(C. Ambrosch-Draxl, Zohreh Basirat, Thomas Dengg,
Rostam Golesorkhtabar, Christian Meisenbichler, Dmitrii Nabok,
Weine Olovsson, Pasquale Pavone, Stephan Sagmeister, Jürgen Spitaler)

About this Document

In order to perform an **exciting** calculation an XML input file called **input.xml** must be provided.

This web page lists all elements and attributes that can be used in the input file of an **exciting** calculation:

- elements are defined according to the general XML conventions (`http://en.wikipedia.org/wiki/XML#Key_terminology`). *Example:* The element `groundstate` (A.11) is used to set up a self-consistent calculation of the ground-state energy.
- attributes are defined according to the general XML conventions (`http://en.wikipedia.org/wiki/XML#Key_terminology`). An attribute is always connected to an element. In **exciting** an attribute generally specifies a parameter or a set of parameters which are connected to the corresponding element. *Example:* The attribute `xctype` (A.11.49) of the element `groundstate` (A.11) defines which exchange-correlation potential is used in the self-consistent calculation.

The input file of an **exciting** calculation is named **input.xml**. A simple example for an input file can be found here (`http://exciting-code.org/input-file-format-overview`). The input file **input.xml** must be a valid XML file and it must contain the root element `input` (A.2).

Unless explicitly stated otherwise, **exciting** uses atomic units ($\hbar = m_e = e = 1$):

- Energies are given in Hartree:

  $1\ \text{Ha} = 2\ \text{Ry} = 27.21138386(68)\ \text{eV} = 4.35926 \cdot 10^{-18}\ \text{J}$
- Lengths are given in Bohr:

  $1\ a_{\text{Bohr}} = 0.52917720859(36)\ \text{Å} = 0.52917720859(36) \cdot 10^{-10}\ \text{m}$

- Magnetic fields are given in units of

  $$1 \text{ a.u.} = \frac{e}{a_{\text{Bohr}}^2} = 1717.2445320376 \text{ Tesla}.$$

Note: The electron charge is positive, so that the atomic numbers Z are negative.

A.1 Input Elements

A.2 Element: input

The XML element `input` (A.2) is the root element of the **exciting** input file. It must contain at least the elements `title` (A.3), `structure` (A.5), and `groundstate` (A.11), each of them must be present only one time.

Contains:	`title` (A.3) (1 times)
	`structure` (A.5) (1 times)
	`groundstate` (A.11) (1 times)
	`structureoptimization` (A.16) (optional)
	`properties` (A.17) (optional)
	`phonons` (A.41) (optional)
	`xs` (A.48) (optional)
	`keywords` (A.4) (optional)
XPath:	`/input`

This element allows for specification of the following attributes:

`scratchpath` (A.2.1)

A.2.1 Attribute: scratchpath

The path to the scratch space where the eigenvector related files, **EVECFV.OUT**, **EVECSV.OUT**, and **OCCSV.OUT** will be written. If the local directory is accessed via a network then `scratchpath` (A.2.1) can be set to a directory on a local disk. The default value is the working directory, *i.e.*, the directory where the program is started.

Type:	anyURI
Default:	"`./`"
Use:	optional
XPath:	`/input/@scratchpath`

A.3 Element: title

The title of the input file, *e.g.*, "Ground-State Calculation for Aluminum".

Type:	string
XPath:	`/input/title`

A.4 Element: keywords

The keywords tag can contain a space separated list of keywords classifying the calculation for archiving purposes. It is not used by the **exciting** program.

Type: string
XPath: /input/keywords

A.5 Element: structure

This element contains all structural information, such as unit-cell parameters as well as type and position of each atom. The presence of the subelement species (A.8) is necessary unless one wants to perform an empty-lattice calculation. The attribute speciespath (C.4.11) must be specified.

Contains: crystal (A.6) (1 times)
species (A.8) (zero or more)
XPath: /input/structure

This element allows for specification of the following attributes:

speciespath (A.5.6) (required), autormt (A.5.1), epslat (A.5.2), molecule (A.5.3), primcell (A.5.4), rmtapm (A.5.5), tshift (A.5.7), vacuum (A.5.8)

A.5.1 Attribute: autormt

If "true", the muffin-tin radius of each species is automatically set according to the variables specified by the attribute rmtapm (A.5.5).

Type: boolean
Default: "false"
Use: optional
XPath: /input/structure/@autormt

A.5.2 Attribute: epslat

This attribute defines the accuracy up to which two vectors can be considered numerically identical. Vectors with lengths less than this are considered zero.

Type: fortrandouble (A.76.1)
Default: "1.0d-6"
Use: optional
Unit: Bohr
XPath: /input/structure/@epslat

A.5.3 Attribute: molecule

If "true", a calculation for an isolated molecule is performed. In this case, the atomic positions specified by the atom (A.9) subelement of the species (A.8) element must be given in cartesian coordinates. The lattice vectors are set up automatically as

$$\mathbf{A}^{(i)} = A_i \, \hat{\mathbf{e}}^{(i)} \qquad i = 1, 2, 3 \tag{A.1}$$

with

$$A_i = \max_{\alpha,\beta} \left| a_i^{(\alpha)} - a_i^{(\beta)} \right| + d_{\text{vac}} \tag{A.2}$$

where $a_i^{(\alpha)}$ is the catesian component of the atom labeled by α in the i-th direction specified by the unit vector $\hat{\mathbf{e}}^{(i)}$. Furthermore, d_{vac} represents the size of the vacuum around the molecule as defined by the attribute vacuum (A.5.8).

Type: boolean
Default: "false"
Use: optional
XPath: /input/structure/@molecule

A.5.4 Attribute: primcell

If "true", the primitive unit cell is determined automatically from the conventional cell defined by the basis vectors given by the basevect (A.7) elements. The primitive unit cell is determined by searching for lattice vectors among all vectors connecting atomic sites and choosing the three shortest ones which produce a unit cell with non-zero volume.

Type: boolean
Default: "false"
Use: optional
XPath: /input/structure/@primcell

A.5.5 Attribute: rmtapm

This attribute assigns the two parameters governing the automatic generation of the muffin-tin radii. When the attribute autormt (A.5.1) is set to "true", the muffin-tin radii are determined according to the following expression

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

where Z_i is the atomic number of the i th species, ζ is stored in rmtapm (A.5.5)(1). The distance between the muffin-tin speheres is determined by the value of rmtapm (A.5.5)(2): When rmtapm (A.5.5)(2)=1, the closest muffin-tin spheres will touch each other.

Type: vect2d (A.76.5)
Default: "0.25d0 0.95d0"
Use: optional
XPath: /input/structure/@rmtapm

A.5.6 Attribute: speciespath

The path to the directory containing the species files. Alternatively, it can be defined as an HTTP URL, in this case the wget (`http://exciting-code.org/wget`) utility must be installed.

Type:	anyURI
Use:	required
XPath:	`/input/structure/@speciespath`

A.5.7 Attribute: tshift

If `"true"`, the crystal is shifted such that the atom closest to the origin is exactly at the origin.

Type:	boolean
Default:	"`true`"
Use:	optional
XPath:	`/input/structure/@tshift`

A.5.8 Attribute: vacuum

Determines the size of the vacuum around the molecule, see the `molecule` (A.5.3) attribute.

Type:	fortrandouble (A.76.1)
Default:	"`10.0d0`"
Use:	optional
Unit:	Bohr
XPath:	`/input/structure/@vacuum`

A.6 Element: crystal

Defines the unit cell of the crystal via the 3 basis vectors.

Contains:	`basevect` (A.7) (3 times)
XPath:	`/input/structure/crystal`

This element allows for specification of the following attributes:

`scale` (A.6.1), `stretch` (A.6.2)

A.6.1 Attribute: scale

Scales all the lattice vectors by the same factor. This is useful for varying the volume.

Type:	fortrandouble (A.76.1)
Default:	"`1.0d0`"
Use:	optional
Unit:	1
XPath:	`/input/structure/crystal/@scale`

A.6.2 Attribute: stretch

Allows for an individual scaling of each lattice vector separately. `"1 1 1"` means no scaling.

Type:	vect3d (A.76.4)
Default:	”`1.0d0 1.0d0 1.0d0` ”
Use:	optional
XPath:	`/input/structure/crystal/@stretch`

A.7 Element: basevect

Defines one basis vector in Cartesian coordinates.

Type:	vect3d (A.76.4)
Unit:	Bohr
XPath:	`/input/structure/crystal/basevect`

A.8 Element: species

Defines the atomic species, *i.e.*, the chemical element. Aatomic coordinates and, optionally, quantities relevant for magnetic calculations are defined in the subelement(s) atom.

Contains:	`atom` (A.9) (1 times or more)
	`LDAplusU` (A.10) (optional)
XPath:	`/input/structure/species`

This element allows for specification of the following attributes:

`speciesfile` (A.8.2) (required), `rmt` (A.8.1)

A.8.1 Attribute: rmt

Defines the muffin-tin radius. This optional parameter allows to override the value either specified in the species file or generated by automatic determination. The muffin-tin radius defines the region around the atomic nucleus where the wave function is expanded in terms of atomic-like functions. In contrast, the interstitial region, *i.e.*, the region not belonging to any muffin-tin sphere, is described by planewaves.

Type:	fortrandouble (A.76.1)
Default:	”`-1.0d0`”
Use:	optional
Unit:	Bohr
XPath:	`/input/structure/species/@rmt`

A.8.2 Attribute: speciesfile

Defines the file that contains the species definition. It is looked up in the species directory specified by `speciespath` (C.4.11). By default, the name of the file is *element*`.xml`, *e.g.*, `Ag.xml`.

Type: anyURI
Use: required
XPath: `/input/structure/species/@speciesfile`

A.9 Element: atom

Defines the position and other attributes of one atom in the unit cell.

Type: no content
XPath: `/input/structure/species/atom`

This element allows for specification of the following attributes:

`coord` (A.9.2) (required), `bfcmt` (A.9.1), `mommtfix` (A.9.3)

A.9.1 Attribute: bfcmt

Muffin-tin external magnetic field in Cartesian coordinates.

Type: vect3d (A.76.4)
Default: "`0.0d0 0.0d0 0.0d0`"
Use: optional
XPath: `/input/structure/species/atom/@bfcmt`

A.9.2 Attribute: coord

Atom position in lattice coordinates.

Type: vect3d (A.76.4)
Use: required
Unit: lattice coordinates
XPath: `/input/structure/species/atom/@coord`

A.9.3 Attribute: mommtfix

The desired muffin-tin moment for a Fixed Spin Moment (FSM) calculation.

Type: vect3d (A.76.4)
Default: "`0.0d0 0.0d0 0.0d0`"
Use: optional
XPath: `/input/structure/species/atom/@mommtfix`

A.10 Element: LDAplusU

The LADplusU element is used to specify the J, U, and l parameters of an atomic species. To switch on the LDAplusU feature one needs to set the `ldapu` (A.11.22) attribute of the groundstate element.

Type: no content
XPath: `/input/structure/species/LDAplusU`

This element allows for specification of the following attributes:

`J` (A.10.1), `U` (A.10.2), `l` (A.10.3)

A.10.1 Attribute: J

Type: fortrandouble (A.76.1)
Default: "`0.0d0`"
Use: optional
XPath: `/input/structure/species/LDAplusU/@J`

A.10.2 Attribute: U

Type: fortrandouble (A.76.1)
Default: "`0.0d0`"
Use: optional
XPath: `/input/structure/species/LDAplusU/@U`

A.10.3 Attribute: l

Type: integer
Default: "`-1`"
Use: optional
XPath: `/input/structure/species/LDAplusU/@l`

A.11 Element: groundstate

The `groundstate` (A.11) element is required for any calculation. Its attributes are the parameters and methods used to calculate the ground-state density.

Contains: `spin` (A.12) (optional)
`solver` (A.13) (optional)
`output` (A.14) (optional)
`libxc` (A.15) (optional)
XPath: `/input/groundstate`

This element allows for specification of the following attributes:

ngridk (A.11.31) (required), autokpt (A.11.1), beta0 (A.11.2), betadec (A.11.3), betainc (A.11.4), cfdamp (A.11.5), chgexs (A.11.6), deband (A.11.7), dlinengyfermi (A.11.8), do (A.11.9), epsband (A.11.10), epschg (A.11.11), epsengy (A.11.12), epsforce (A.11.13), epsocc (A.11.14), epspot (A.11.15), fermilinengy (A.11.16), findlinentype (A.11.17), fracinr (A.11.18), frozencore (A.11.19), gmaxvr (A.11.20), isgkmax (A.11.21), ldapu (A.11.22), lmaxapw (A.11.23), lmaxinr (A.11.24), lmaxmat (A.11.25), lmaxvr (A.11.26), lradstep (A.11.27), maxscl (A.11.28), mixer (A.11.29), nempty (A.11.30), nktot (A.11.32), nosource (A.11.33), nosym (A.11.34), nprad (A.11.35), npsden (A.11.36), nwrite (A.11.37), ptnucl (A.11.38), radkpt (A.11.39), reducek (A.11.40), rgkmax (A.11.41), stype (A.11.42), swidth (A.11.43), symmorph (A.11.44), tevecsv (A.11.45), tfibs (A.11.46), tforce (A.11.47), vkloff (A.11.48), xctype (A.11.49)

A.11.1 Attribute: autokpt

If "true", the set of **k**-points is determined automatically according to the total number of required **k**-points given by nktot (A.11.32).

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/groundstate/@autokpt

A.11.2 Attribute: beta0

Initial value for mixing parameter. Used in linear mixing as choosen with mixer (A.11.29).

Type:	fortrandouble (A.76.1)
Default:	"0.4d0"
Use:	optional
XPath:	/input/groundstate/@beta0

A.11.3 Attribute: betadec

Mixing parameter decrease. Used in linear mixing.

Type:	fortrandouble (A.76.1)
Default:	"0.6d0"
Use:	optional
XPath:	/input/groundstate/@betadec

A.11.4 Attribute: betainc

Mixing parameter increase. Used in linear mixing.

Type:	fortrandouble (A.76.1)
Default:	"1.1d0"
Use:	optional
XPath:	/input/groundstate/@betainc

A.11.5 Attribute: cfdamp

Damping coefficient for characteristic function.

Type: fortrandouble (A.76.1)
Default: "0.0d0"
Use: optional
XPath: /input/groundstate/@cfdamp

A.11.6 Attribute: chgexs

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

Type: fortrandouble (A.76.1)
Default: "0.0d0"
Use: optional
XPath: /input/groundstate/@chgexs

A.11.7 Attribute: deband

Initial band energy step size The initial step length used when searching for the band energy, which is used as the APW linearisation energy. This is done by first searching upwards in energy until the radial wave-function at the muffin-tin radius is zero. This is the energy at the top of the band, denoted E_{t}. A downward search is now performed from E_{t} until the slope of the radial wave-function at the muffin-tin radius is zero. This energy, E_{b}, is at the bottom of the band. The band energy is taken as $(E_{\mathrm{t}} + E_{\mathrm{b}})/2$. If either E_{t} or E_{b} cannot be found then the band energy is set to the default value.

Type: fortrandouble (A.76.1)
Default: "0.0025d0"
Use: optional
Unit: Hartree
XPath: /input/groundstate/@deband

A.11.8 Attribute: dlinengyfermi

Energy difference between linearisation and Fermi energy.

Type: fortrandouble (A.76.1)
Default: "-0.1d0"
Use: optional
Unit: Hartree
XPath: /input/groundstate/@dlinengyfermi

A.11.9 Attribute: do

Decides if the ground state is calculated starting from scratch, using the densities from file or if it is skipped and only its associated input parameters are read in. Also applies fo structural optimization run.

Type:	**choose from:**
	`fromscratch`
	`fromfile`
	`skip`
Default:	"`fromscratch`"
Use:	optional
XPath:	`/input/groundstate/@do`

A.11.10 Attribute: epsband

Energy tolerance for search of linearisation energies.

Type:	fortrandouble (A.76.1)
Default:	"`1.0d-6`"
Use:	optional
Unit:	Hartree
XPath:	`/input/groundstate/@epsband`

A.11.11 Attribute: epschg

Maximum allowed error in the calculated total charge beyond which a warning message will be issued.

Type:	fortrandouble (A.76.1)
Default:	"`1.0d-3`"
Use:	optional
XPath:	`/input/groundstate/@epschg`

A.11.12 Attribute: epsengy

Energy convergence tolerance.

Type:	fortrandouble (A.76.1)
Default:	"`1.0d-4`"
Use:	optional
Unit:	Hartree
XPath:	`/input/groundstate/@epsengy`

A.11.13 Attribute: epsforce

Convergence tolerance for the forces during the SCF run.

Type:	fortrandouble (A.76.1)
Default:	"`5.0d-5`"
Use:	optional
XPath:	`/input/groundstate/@epsforce`

A.11.14 Attribute: epsocc

smallest occupancy for which a state will contribute to the density.

Type:	fortrandouble (A.76.1)
Default:	"1.0d-8"
Use:	optional
XPath:	/input/groundstate/@epsocc

A.11.15 Attribute: epspot

If the RMS change in the effective potential and magnetic field is smaller than epspot (A.11.15), then the self-consistent loop is considered converged and exited. For structural optimization runs this results in the forces being calculated, the atomic positions updated and the loop restarted. See also maxscl (A.11.28).

Type:	fortrandouble (A.76.1)
Default:	"1.0d-6"
Use:	optional
XPath:	/input/groundstate/@epspot

A.11.16 Attribute: fermilinengy

If "true" the linearization energies marked as non-varying are set to the Fermi level plus dlinengyfermi (A.11.8).

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/groundstate/@fermilinengy

A.11.17 Attribute: findlinentype

Select method to determine the linearisation energies.

Type:	**choose from:**
	simple
	advanced
Default:	"advanced"
Use:	optional
XPath:	/input/groundstate/@findlinentype

A.11.18 Attribute: fracinr

Fraction of the muffin-tin radius up to which lmaxinr is used as the angular momentum cut-off.

Type:	fortrandouble (A.76.1)
Default:	"0.25d0"
Use:	optional
XPath:	/input/groundstate/@fracinr

A.11.19 Attribute: frozencore

When set to "true" the frozen core approximation is applied, i.e., the core states are fixed to the atomic states.

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/groundstate/@frozencore

A.11.20 Attribute: gmaxvr

Maximum length of —G— for expanding the interstitial density and potential.

Type:	fortrandouble (A.76.1)
Default:	"12.0d0"
Use:	optional
XPath:	/input/groundstate/@gmaxvr

A.11.21 Attribute: isgkmax

Species for which the muffin-tin radius will be used for calculating gkmax.

Type:	integer
Default:	"-1"
Use:	optional
XPath:	/input/groundstate/@isgkmax

A.11.22 Attribute: ldapu

Type of LDA+U method to be used.

Type:	**choose from:**
	none
	FullyLocalisedLimit
	AroundMeanField
	FFL-AMF-interpolation
Default:	"none"
Use:	optional
XPath:	/input/groundstate/@ldapu

A.11.23 Attribute: lmaxapw

Angular momentum cut-off for the APW functions.

Type:	integer
Default:	"10"
Use:	optional
XPath:	/input/groundstate/@lmaxapw

A.11.24 Attribute: lmaxinr

Close to the nucleus, the density and potential is almost spherical and therefore the spherical harmonic expansion can be truncated a low angular momentum. See also `fracinr` (A.11.18).

Type: integer
Default: "2"
Use: optional
XPath: `/input/groundstate/@lmaxinr`

A.11.25 Attribute: lmaxmat

Angular momentum cut-off for the outer-most loop in the hamiltonian and overlap matrix setup.

Type: integer
Default: "5"
Use: optional
XPath: `/input/groundstate/@lmaxmat`

A.11.26 Attribute: lmaxvr

Angular momentum cut-off for the muffin-tin density and potential.

Type: integer
Default: "6"
Use: optional
XPath: `/input/groundstate/@lmaxvr`

A.11.27 Attribute: lradstep

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

Type: integer
Default: "4"
Use: optional
XPath: `/input/groundstate/@lradstep`

A.11.28 Attribute: maxscl

Upper limit for te self-consistency loop.

Type: integer
Default: "200"
Use: optional
XPath: `/input/groundstate/@maxscl`

A.11.29 Attribute: mixer

Select the mixing (relaxation) scheme for the SCF loop. One has the folloeing options:

Linear mixer (`"lin"`)

Given the input μ^i and output ν^i vectors of the ith iteration, the next input vector to the $(i+1)$th iteration is generated using an adaptive mixing scheme. The jth component of the output vector is mixed with a fraction of the same component of the input vector:

$$\mu_j^{i+1} = \beta_j^i \nu_j^i + (1-\beta_j^i)\mu_j^i, \tag{A.4}$$

where β_j^i is set to β_0 at initialisation and increased by scaling with $\beta_{\rm inc}$ (>1) if $f_j^i \equiv \nu_j^i - \mu_j^i$ does not change sign between loops. If f_j^i does change sign, then β_j^i is scaled by $\beta_{\rm dec}$ (>1). Note that the array `nu` serves for both input and output, and the arrays `mu`, `beta` and `f` are used internally and should not be changed between calls. The routine is initialised at the first iteration and is thread-safe so long as each thread has its own independent work array. Complex arrays may be passed as real arrays with n doubled.

Type:	**choose from:**
	`lin`
	`msec`
	`pulay`
Default:	"`msec`"
Use:	optional
XPath:	`/input/groundstate/@mixer`

A.11.30 Attribute: nempty

Defines the number of eigenstates beyond that required for charge neutrality. When running metals it is not known *a priori* how many states will be below the Fermi energy for each **k**-point. Setting `nempty` (A.50.2) greater than zero allows the additional states to act as a buffer in such cases. Furthermore, magnetic calculations use the first-variational eigenstates as a basis for setting up the second-variational Hamiltonian, and thus `nempty` (A.50.2) will determine the size of this basis set. Convergence with respect to this quantity should be checked.

Type:	integer
Default:	"`5`"
Use:	optional
XPath:	`/input/groundstate/@nempty`

A.11.31 Attribute: ngridk

Number of k grid points along the basis vector directions.

Type:	integertriple (A.76.6)
Use:	required
XPath:	`/input/groundstate/@ngridk`

A.11.32 Attribute: nktot

Used for the automatic determination of the **k**-point mesh from the total number of **k**-points. If `nktot` (A.11.32) is set, then the mesh will be determined in such a way that the number of **k**-points is proportional to the length of the reciprocal lattice vector in each direction and that the total number of **k**-points is less than or equal to `nktot` (A.11.32).

Type: integer
Default: "0"
Use: optional
XPath: `/input/groundstate/@nktot`

A.11.33 Attribute: nosource

When set to `"true"`, source fields are projected out of the exchange-correlation magnetic field. experimental feature.

Type: boolean
Default: "false"
Use: optional
XPath: `/input/groundstate/@nosource`

A.11.34 Attribute: nosym

When set to `"true"` no symmetries, apart from the identity, are used anywhere in the code.

Type: boolean
Default: "false"
Use: optional
XPath: `/input/groundstate/@nosym`

A.11.35 Attribute: nprad

Smallest occupancy for which a state will contribute to the density.

Type: integer
Default: "4"
Use: optional
XPath: `/input/groundstate/@nprad`

A.11.36 Attribute: npsden

oOrder of polynomial for pseudo-charge density.

Type: integer
Default: "9"
Use: optional
XPath: `/input/groundstate/@npsden`

A.11.37 Attribute: nwrite

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

Type: integer
Default: "0"
Use: optional
XPath: `/input/groundstate/@nwrite`

A.11.38 Attribute: ptnucl

The attrubute ptnucl is `"true"` if the nuclei are to be treated as point charges, if `"false"` the nuclei have a finite spherical distribution.

Type: boolean
Default: "true"
Use: optional
XPath: `/input/groundstate/@ptnucl`

A.11.39 Attribute: radkpt

Used for the automatic determination of the **k**-point mesh. If `autokpt` (A.11.1) is set to `"true"` then the mesh sizes will be determined by $n_i = \lambda/|\mathbf{A}_i| + 1$.

Type: fortrandouble (A.76.1)
Default: "40.0d0"
Use: optional
XPath: `/input/groundstate/@radkpt`

A.11.40 Attribute: reducek

If the attribute `reducek` (A.51.11) is `"true"` the **k**-point set is reduced with the crystal symmetries.

Type: boolean
Default: "true"
Use: optional
XPath: `/input/groundstate/@reducek`

A.11.41 Attribute: rgkmax

The parameter `rgkmax` (A.51.12) implicitly determines the number of basis functions and is one of the crucial parameters for the accuracy of the calculation. It represents the product of two quantities: $R_{MT,\,Min}$, the smallest of all muffin-tin radii, and $|\mathbf{G}+\mathbf{k}|_{max}$, the maximum length for the $\mathbf{G}+\mathbf{k}$ vectors. Because each $\mathbf{G}+\mathbf{k}$ vector represents one basis function, `rgkmax` (A.51.12) gives the number of basis functions used for solving the Kohn-Sham equations. Typical values of `rgkmax` (A.51.12) are between 6 and 9. However, for systems with very short bond-lengths, significantly smaller values may be sufficient.

This may especially be the case for materials containing carbon, where `rgkmax` (A.51.12) may be 4.5-5, or hydrogen, where even values between 3 and 4 may be sufficient. In any case, a convergence check is indispensible for a proper choice of this parameter for your system!

Type:	fortrandouble (A.76.1)
Default:	"`7.0d0`"
Use:	optional
XPath:	`/input/groundstate/@rgkmax`

A.11.42 Attribute: stype

A smooth approximation to the Dirac delta function is needed to compute the occupancies of the Kohn-Sham states. The attribute `swidth` (A.48.23) determines the width of the approximate delta function.

Type:	**choose from:**
	`Gaussian`
	`Methfessel-Paxton 1`
	`Methfessel-Paxton 2`
	`Fermi Dirac`
	`Square-wave impulse`

Default:	"`Gaussian`"
Use:	optional
XPath:	`/input/groundstate/@stype`

A.11.43 Attribute: swidth

Width of the smooth approximation to the Dirac delta function (must be greater than zero).

Type:	fortrandouble (A.76.1)
Default:	"`0.001d0`"
Use:	optional
Unit:	Hartree
XPath:	`/input/groundstate/@swidth`

A.11.44 Attribute: symmorph

When set to `"true"` only symmorphic space-group operations are to be considered, i.e. only symmetries without non-primitive translations are used anywhere in the code.

Type:	boolean
Default:	"`false`"
Use:	optional
XPath:	`/input/groundstate/@symmorph`

A.11.45 Attribute: tevecsv

The attribute tevecsv is `"true"` if second-variational eigenvectors are calculated.

Type:	boolean
Default:	"`false`"
Use:	optional
XPath:	`/input/groundstate/@tevecsv`

A.11.46 Attribute: tfibs

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

Type:	boolean
Default:	"`true`"
Use:	optional
XPath:	`/input/groundstate/@tfibs`

A.11.47 Attribute: tforce

Decides if the force should be calculated at the end of the self-consistent cycle.

Type:	boolean
Default:	"`false`"
Use:	optional
XPath:	`/input/groundstate/@tforce`

A.11.48 Attribute: vkloff

The **k**-point offset vector in lattice coordinates.

Type:	vect3d (A.76.4)
Default:	"`0.0d0 0.0d0 0.0d0`"
Use:	optional
XPath:	`/input/groundstate/@vkloff`

A.11.49 Attribute: xctype

Type of exchange-correlation functional to be used

- No exchange-correlation funtional ($E_{\rm xc} \equiv 0$)
- LDA, Perdew-Zunger/Ceperley-Alder, *Phys. Rev. B* **23**, 5048 (1981)
- LSDA, Perdew-Wang/Ceperley-Alder, *Phys. Rev. B* **45**, 13244 (1992)
- LDA, X-alpha approximation, J. C. Slater, *Phys. Rev.* **81**, 385 (1951)
- LSDA, von Barth-Hedin, *J. Phys. C* **5**, 1629 (1972)

- GGA, Perdew-Burke-Ernzerhof, *Phys. Rev. Lett.* **77**, 3865 (1996)
- GGA, Revised PBE, Zhang-Yang, *Phys. Rev. Lett.* **80**, 890 (1998)
- GGA, PBEsol, arXiv:0707.2088v1 (2007)
- GGA, Wu-Cohen exchange (WC06) with PBE correlation, *Phys. Rev. B* **73**, 235116 (2006)
- GGA, Armiento-Mattsson (AM05) spin-unpolarised functional, *Phys. Rev. B* **72**, 085108 (2005)

Type:	**choose from:**
	`LDAPerdew-Zunger`
	`LSDAPerdew-Wang`
	`LDA-X-alpha`
	`LSDA-Barth-Hedin`
	`GGAPerdew-Burke-Ernzerhof`
	`GGArevPBE`
	`GGAPBEsol`
	`GGA-Wu-Cohen`
	`GGAArmiento-Mattsson`
	`EXX`
	`none`
Default:	"`LSDAPerdew-Wang`"
Use:	optional
XPath:	`/input/groundstate/@xctype`

A.12 Element: spin

If the `spin` (A.12) element is present calculation is done with spin polarization.

Type:	no content
XPath:	`/input/groundstate/spin`

This element allows for specification of the following attributes:

`bfieldc` (A.12.1), `fixspin` (A.12.2), `momfix` (A.12.3), `reducebf` (A.12.4), `spinorb` (A.12.5), `spinsprl` (A.12.6), `taufsm` (A.12.7), `vqlss` (A.12.8)

A.12.1 Attribute: bfieldc

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

$$\frac{g_e \alpha}{4} \vec{\sigma} \cdot \mathbf{B}_{\text{ext}}, \tag{A.5}$$

where g_e is the electron g-factor (2.0023193043718). This field is normally used to break spin symmetry for spin-polarised calculations and considered to be infinitesimal with no direct contribution to the total energy. In cases where the magnetic field is finite (for example when computing magnetic response) the external **B**-field energy reported in `INFO.OUT`

should be added to the total by hand. This field is applied throughout the entire unit cell. To apply magnetic fields in particular muffin-tins use the bfcmt (A.9.1) vectors in the atom (A.9) elements. Collinear calculations are more efficient if the field is applied in the z-direction.

Type: vect3d (A.76.4)
Default: "0.0d0 0.0d0 0.0d0 "
Use: optional
XPath: /input/groundstate/spin/@bfieldc

A.12.2 Attribute: fixspin

Type: **choose from:**
none
total FSM
localmt FSM
both
Default: "none"
Use: optional
XPath: /input/groundstate/spin/@fixspin

A.12.3 Attribute: momfix

The desired total moment for a FSM calculation.

Type: vect3d (A.76.4)
Default: "0.0d0 0.0d0 0.0d0"
Use: optional
XPath: /input/groundstate/spin/@momfix

A.12.4 Attribute: reducebf

After each iteration the external magnetic fields are multiplied with reducebf. This allows for a large external magnetic field at the start of the self-consistent loop to break spin symmetry, while at the end of the loop the field will be effectively zero, i.e. infinitesimal. See bfieldc (A.12.1) and atom element.

Type: fortrandouble (A.76.1)
Default: "1.0d0"
Use: optional
XPath: /input/groundstate/spin/@reducebf

A.12.5 Attribute: spinorb

If spinorb (A.12.5) is "true", then a $\boldsymbol{\sigma} \cdot \mathbf{L}$ term is added to the second-variational Hamiltonian.

Type: boolean
Use: optional
XPath: /input/groundstate/spin/@spinorb

A.12.6 Attribute: spinsprl

Set to "true" if a spin-spiral calculation is required. Experimental feature for the calculation of spin-spiral states. See vqlss (A.12.8) for details.

Type: boolean
Default: "false"
Use: optional
XPath: /input/groundstate/spin/@spinsprl

A.12.7 Attribute: taufsm

Type: fortrandouble (A.76.1)
Default: "0.01d0"
Use: optional
XPath: /input/groundstate/spin/@taufsm

A.12.8 Attribute: vqlss

Is the **q**-vector of the spin-spiral state in lattice coordinates. Spin-spirals arise from spinor states assumed to be of the form

$$\Psi_{\mathbf{k}}^{\mathbf{q}}(\mathbf{r}) = \begin{pmatrix} U_{\mathbf{k}}^{\mathbf{q}\uparrow}(\mathbf{r}) e^{i(\mathbf{k}+\mathbf{q}/\mathbf{2})\cdot\mathbf{r}} \\ U_{\mathbf{k}}^{\mathbf{q}\downarrow}(\mathbf{r}) e^{i(\mathbf{k}-\mathbf{q}/\mathbf{2})\cdot\mathbf{r}} \end{pmatrix}. \tag{A.6}$$

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

$$\mathbf{m}^{\mathbf{q}}(\mathbf{r}) = (m_x(\mathbf{r})\cos(\mathbf{q}\cdot\mathbf{r}), m_y(\mathbf{r})\sin(\mathbf{q}\cdot\mathbf{r}), m_z(\mathbf{r})), \tag{A.7}$$

where m_x, m_y and m_z are lattice periodic. See also spinsprl (A.12.6).

Type: vect3d (A.76.4)
Default: "0.0d0 0.0d0 0.0d0"
Use: optional
XPath: /input/groundstate/spin/@vqlss

A.13 Element: solver

Optional configuration options for eigenvector solver.

Type: no content
XPath: /input/groundstate/solver

This element allows for specification of the following attributes:

epsarpack (A.13.1), evaltol (A.13.2), packedmatrixstorage (A.13.3), type (A.13.4)

A.13.1 Attribute: epsarpack

Tolerance parameter for the ARPACK shift invert solver

Type:	fortrandouble (A.76.1)
Default:	"`1.0d-8`"
Use:	optional
XPath:	`/input/groundstate/solver/@epsarpack`

A.13.2 Attribute: evaltol

Error tolerance for the first-variational eigenvalues using the LAPACK Solver

Type:	fortrandouble (A.76.1)
Default:	"`1.0d-8`"
Use:	optional
Unit:	Hartree
XPath:	`/input/groundstate/solver/@evaltol`

A.13.3 Attribute: packedmatrixstorage

In the default calculation the matrix is sored in packed form. When using multi-threaded BLAS setting this parameter to `"false"` increases efficiency.

Type:	boolean
Default:	"`true`"
Use:	optional
XPath:	`/input/groundstate/solver/@packedmatrixstorage`

A.13.4 Attribute: type

Selects the eigenvalue solver for the first variational equation

Type:	**choose from:**
	`Lapack`
	`Arpack`
	`DIIS`
Default:	"`Lapack`"
Use:	optional
XPath:	`/input/groundstate/solver/@type`

A.14 Element: output

Specifications on the file formats for output files.

Type:	no content
XPath:	`/input/groundstate/output`

This element allows for specification of the following attributes:

`state` (A.14.1)

A.14.1 Attribute: state

Selects the file format of the STATE file.

Type:	**choose from:**
	`binary`
	`XML`
Default:	"`binary`"
Use:	optional
XPath:	`/input/groundstate/output/@state`

A.15 Element: libxc

Type:	no content
XPath:	`/input/groundstate/libxc`

This element allows for specification of the following attributes:

`correlation` (A.15.1), `exchange` (A.15.2), `xc` (A.15.3)

A.15.1 Attribute: correlation

Type:	**choose from:**
	XC_LDA_C_WIGNER
	XC_LDA_C_RPA
	XC_LDA_C_HL
	XC_LDA_C_GL
	XC_LDA_C_XALPHA
	XC_LDA_C_VWN
	XC_LDA_C_VWN_RPA
	XC_LDA_C_PZ
	XC_LDA_C_PZ_MOD
	XC_LDA_C_OB_PZ
	XC_LDA_C_PW
	XC_LDA_C_PW_MOD
	XC_LDA_C_OB_PW
	XC_LDA_C_2D_AMGB
	XC_LDA_C_2D_PRM
	XC_LDA_C_vBH
	XC_LDA_C_1D_CSC
	XC_GGA_C_PBE
	XC_GGA_C_LYP
	XC_GGA_C_P86
	XC_GGA_C_PBE_SOL
	XC_GGA_C_PW91
	XC_GGA_C_AM05
	XC_GGA_C_XPBE
	XC_GGA_C_LM
	XC_GGA_C_PBE_JRGX
Default:	"XC_GGA_C_PBE"
Use:	optional
XPath:	/input/groundstate/libxc/@correlation

A.15.2 Attribute: exchange

Type:	**choose from:**
	XC_LDA_X
	XC_LDA_X_2D
	XC_GGA_X_PBE
	XC_GGA_X_PBE_R
	XC_GGA_X_B86
	XC_GGA_X_B86_R
	XC_GGA_X_B86_MGC
	XC_GGA_X_B88
	XC_GGA_X_G96
	XC_GGA_X_PW86
	XC_GGA_X_PW91
	XC_GGA_X_OPTX
	XC_GGA_X_DK87_R1
	XC_GGA_X_DK87_R2
	XC_GGA_X_LG93
	XC_GGA_X_FT97_A
	XC_GGA_X_FT97_B
	XC_GGA_X_PBE_SOL
	XC_GGA_X_RPBE
	XC_GGA_X_WC
	XC_GGA_X_mPW91
	XC_GGA_X_AM05
	XC_GGA_X_PBEA
	XC_GGA_X_MPBE
	XC_GGA_X_XPBE
	XC_GGA_X_2D_B86_MGC
	XC_GGA_X_BAYESIAN
	XC_GGA_X_PBE_JSJR
Default:	"XC_GGA_X_PBE"
Use:	optional
XPath:	/input/groundstate/libxc/@exchange

A.15.3 Attribute: xc

Combined functionals. If set it overrides the exchange and the correlation attributes. The hybrid functionals can be configured but are not supported. They may give nonsense results.

Type: **choose from:**
none
XC_GGA_XC_LB
XC_GGA_XC_HCTH_93
XC_GGA_XC_HCTH_120
XC_GGA_XC_HCTH_147
XC_GGA_XC_HCTH_407
XC_GGA_XC_EDF1
XC_GGA_XC_XLYP
XC_GGA_XC_B97
XC_GGA_XC_B97_1
XC_GGA_XC_B97_2
XC_GGA_XC_B97_D
XC_GGA_XC_B97_K
XC_GGA_XC_B97_3
XC_GGA_XC_PBE1W
XC_GGA_XC_MPWLYP1W
XC_GGA_XC_PBELYP1W
XC_GGA_XC_SB98_1a
XC_GGA_XC_SB98_1b
XC_GGA_XC_SB98_1c
XC_GGA_XC_SB98_2a
XC_GGA_XC_SB98_2b
XC_GGA_XC_SB98_2c
XC_HYB_GGA_XC_B3PW91
XC_HYB_GGA_XC_B3LYP
XC_HYB_GGA_XC_B3P86
XC_HYB_GGA_XC_O3LYP
XC_HYB_GGA_XC_mPW1K
XC_HYB_GGA_XC_PBEH
XC_HYB_GGA_XC_B97
XC_HYB_GGA_XC_B97_1
XC_HYB_GGA_XC_B97_2
XC_HYB_GGA_XC_X3LYP
XC_HYB_GGA_XC_B1WC
XC_HYB_GGA_XC_B97_K
XC_HYB_GGA_XC_B97_3
XC_HYB_GGA_XC_mPW3PW
XC_HYB_GGA_XC_B1LYP
XC_HYB_GGA_XC_B1PW91
XC_HYB_GGA_XC_mPW1PW
XC_HYB_GGA_XC_mPW3LYP
XC_HYB_GGA_XC_SB98_1a
XC_HYB_GGA_XC_SB98_1b
XC_HYB_GGA_XC_SB98_1c
XC_HYB_GGA_XC_SB98_2a
XC_HYB_GGA_XC_SB98_2b
XC_HYB_GGA_XC_SB98_2c
Default: "none"
Use: optional
XPath: /input/groundstate/libxc/@xc

A.16 Element: structureoptimization

The element `structureoptimization` (A.16) activates the optimization of atomic postions.

Type: no content
XPath: `/input/structureoptimization`

This element allows for specification of the following attributes:

`epsforce` (A.16.1), `resume` (A.16.2), `tau0atm` (A.16.3)

A.16.1 Attribute: epsforce

Convergence tolerance for the forces during a structural optimization run.

Type: fortrandouble (A.76.1)
Default: "`5.0d-5`"
Use: optional
XPath: `/input/structureoptimization/@epsforce`

A.16.2 Attribute: resume

Resumption of a structural optimization run using the density in `STATE.OUT`, but with positions from `input.xml`.

Type: boolean
Default: "`false`"
Use: optional
XPath: `/input/structureoptimization/@resume`

A.16.3 Attribute: tau0atm

Parameter determining the step size for structural optimization.

In each step m of a structural optimization run, atom α is displaced according to

$$\mathbf{r}_\alpha^{m+1} = \mathbf{r}_\alpha^m + \tau_\alpha^m \left(\mathbf{F}_\alpha^m + \mathbf{F}_\alpha^{m-1} \right), \tag{A.8}$$

i.e., the magnitude of the displacement in step m is proportional to τ_α^m. For the initial step, τ_α^0 is set to `tau0atm` (A.16.3). If the forces of two subsequent steps have the same sign, τ_α^m is increased by τ_α^0. Otherwise, τ_α^m is reset to τ_α^0.

Type: fortrandouble (A.76.1)
Default: "`0.2d0`"
Use: optional
XPath: `/input/structureoptimization/@tau0atm`

A.17 Element: properties

Properties listed in this element can be calculated from the groundstate. It works also from a saved state from a previous run.

Contains: bandstructure (A.18) (optional)
STM (A.19) (optional)
wfplot (A.20) (optional)
dos (A.21) (optional)
LSJ (A.22) (optional)
masstensor (A.23) (optional)
chargedensityplot (A.24) (optional)
exccplot (A.25) (optional)
elfplot (A.26) (optional)
mvecfield (A.27) (optional)
xcmvecfield (A.28) (optional)
electricfield (A.29) (optional)
gradmvecfield (A.30) (optional)
fermisurfaceplot (A.31) (optional)
EFG (A.32) (optional)
mossbauer (A.33) (optional)
momentummatrix (A.34) (optional)
dielectric (A.35) (optional)
moke (A.37) (optional)
expiqr (A.38) (optional)
elnes (A.39) (optional)
eliashberg (A.40) (optional)
XPath: `/input/properties`

A.18 Element: bandstructure

If present a banstructure is calculated.

Contains: plot1d (A.66)
XPath: `/input/properties/bandstructure`

This element allows for specification of the following attributes:

character (A.18.1), scissor (A.18.2)

A.18.1 Attribute: character

Band structure plot which includes angular momentum characters for every atom.

Type: boolean
Default: "false"
Use: optional
XPath: `/input/properties/bandstructure/@character`

A.18.2 Attribute: scissor

Value to shift bandgap.

Type:	fortrandouble (A.76.1)
Default:	"0.0d0"
Use:	optional
Unit:	Hartree
XPath:	/input/properties/bandstructure/@scissor

A.19 Element: STM

Contains:	plot2d (A.68) (optional)
XPath:	/input/properties/STM

A.20 Element: wfplot

Wavefunction plot.

Contains:	kstlist (A.72) (1 times) plot1d (A.66) (optional) plot2d (A.68) (optional) plot3d (A.70) (optional)
XPath:	/input/properties/wfplot

A.21 Element: dos

If present a DOS calculation is started.

DOS and optics plots require integrals of the kind

$$g(\omega_i) = \frac{\Omega}{(2\pi)^3} \int_{\mathrm{BZ}} f(\mathbf{k}) \delta(\omega_i - e(\mathbf{k})) d\mathbf{k}. \tag{A.9}$$

These are calculated by first interpolating the functions $e(\mathbf{k})$ and $f(\mathbf{k})$ with the trilinear method on a much finer mesh whose size is determined by ngrdos (A.42.1). Then the ω-dependent histogram of the integrand is accumulated over the fine mesh. If the output function is noisy then either ngrdos (A.42.1) should be increased or nwdos (A.42.4) decreased. Alternatively, the output function can be artificially smoothed up to a level given by nsmdos (A.42.2). This is the number of successive 3-point averages to be applied to the function g.

Type:	no content
XPath:	/input/properties/dos

This element allows for specification of the following attributes:

lmirep (A.21.1), ngrdos (A.21.2), nsmdos (A.21.3), nwdos (A.21.4), scissor (A.21.5), sqados (A.21.6), winddos (A.21.7)

A.21.1 Attribute: lmirep

When lmirep is set to "true", the spherical harmonic basis is transformed into one in which the site symmetries are block diagonal. Band characters determined from the density matrix expressed in this basis correspond to irreducible representations, and allow the partial DOS to be resolved into physically relevant contributions, for example eg and t2g.

Type: boolean
Default: "false"
Use: optional
XPath: /input/properties/dos/@lmirep

A.21.2 Attribute: ngrdos

Type: integer
Default: "100"
Use: optional
XPath: /input/properties/dos/@ngrdos

A.21.3 Attribute: nsmdos

Type: integer
Default: "0"
Use: optional
XPath: /input/properties/dos/@nsmdos

A.21.4 Attribute: nwdos

Type: integer
Default: "500"
Use: optional
XPath: /input/properties/dos/@nwdos

A.21.5 Attribute: scissor

Type: fortrandouble (A.76.1)
Default: "0.0d0"
Use: optional
Unit: Hartree
XPath: /input/properties/dos/@scissor

A.21.6 Attribute: sqados

Spin-quantization axis in Cartesian coordinates used when plotting the spin-resolved DOS (z-axis by default).

Type: vect3d (A.76.4)
Default: "0.0d0 0.0d0 1.0d0"
Use: optional
XPath: /input/properties/dos/@sqados

A.21.7 Attribute: winddos

Frequency/energy window for the DOS or optics plot.

Type: vect2d (A.76.5)
Default: "-0.5d0 0.5d0"
Use: optional
Unit: Hartree
XPath: /input/properties/dos/@winddos

A.22 Element: LSJ

Output L, S and J expectation values.

Contains: kstlist (A.72) (optional)
XPath: /input/properties/LSJ

A.23 Element: masstensor

Compute the effective mass tensor at the **k**-point given by vklem.

Type: no content
XPath: /input/properties/masstensor

This element allows for specification of the following attributes:

deltaem (A.23.1), ndspem (A.23.2), vklem (A.23.3)

A.23.1 Attribute: deltaem

The size of the **k**-vector displacement used when calculating numerical derivatives for the effective mass tensor.

Type: fortrandouble (A.76.1)
Default: "0.025d0"
Use: optional
XPath: /input/properties/masstensor/@deltaem

A.23.2 Attribute: ndspem

The number of **k**-vector displacements in each direction around vklem when computing the numerical derivatives for the effective mass tensor.

Type: integer
Default: "1"
Use: optional
XPath: /input/properties/masstensor/@ndspem

A.23.3 Attribute: vklem

The **k**-point in lattice coordinates at which to compute the effective mass tensors.

Type:	vect3d (A.76.4)
Default:	"`0.0d0 0.0d0 0.0d0`"
Use:	optional
XPath:	`/input/properties/masstensor/@vklem`

A.24 Element: chargedensityplot

Plot the charge density

Contains:	`plot1d` (A.66) (optional) `plot2d` (A.68) (optional) `plot3d` (A.70) (optional)
XPath:	`/input/properties/chargedensityplot`

This element allows for specification of the following attributes:

`plotgradient` (A.24.1)

A.24.1 Attribute: plotgradient

Calculate and plot the module of the density gradient

Type:	boolean
Default:	"`false`"
Use:	optional
XPath:	`/input/properties/chargedensityplot/@plotgradient`

A.25 Element: exccplot

Exchange-correlation and Coulomb potential plots.

Contains:	`plot1d` (A.66) (optional) `plot2d` (A.68) (optional) `plot3d` (A.70) (optional)
XPath:	`/input/properties/exccplot`

A.26 Element: elfplot

Electron localization function (ELF).

Contains:	`plot1d` (A.66) (optional) `plot2d` (A.68) (optional) `plot3d` (A.70) (optional)
XPath:	`/input/properties/elfplot`

A.27 Element: mvecfield

Plot of magnetization vector field.

Contains: plot2d (A.68) (optional)
plot3d (A.70) (optional)
XPath: /input/properties/mvecfield

A.28 Element: xcmvecfield

Plot of exchange-correlation magnetic vector field.

Contains: plot2d (A.68) (optional)
plot3d (A.70) (optional)
XPath: /input/properties/xcmvecfield

A.29 Element: electricfield

Writes the electric field to file.

Contains: plot2d (A.68) (optional)
plot3d (A.70) (optional)
XPath: /input/properties/electricfield

A.30 Element: gradmvecfield

Plot of he gradient of the magnetic vector field.

Contains: plot1d (A.66) (optional)
plot2d (A.68) (optional)
plot3d (A.70) (optional)
XPath: /input/properties/gradmvecfield

A.31 Element: fermisurfaceplot

Writes Fermi surface data to file.

Type: no content
XPath: /input/properties/fermisurfaceplot

This element allows for specification of the following attributes:

nstfsp (A.31.1), separate (A.31.2)

A.31.1 Attribute: nstfsp

Number of states to be included in the Fermi surface plot file.

Type: integer
Default: "6"
Use: optional
XPath: /input/properties/fermisurfaceplot/@nstfsp

A.31.2 Attribute: separate

Type: boolean
Default: "false"
Use: optional
XPath: /input/properties/fermisurfaceplot/@separate

A.32 Element: EFG

Calculation of electric field gradient (EFG), contact charge.

Type: no content
XPath: /input/properties/EFG

A.33 Element: mossbauer

Type: no content
XPath: /input/properties/mossbauer

A.34 Element: momentummatrix

Matrix elements of the momentum operator (legacy version, required by dielectric-element).

Type: no content
XPath: /input/properties/momentummatrix

This element allows for specification of the following attributes:

fastpmat (A.34.1)

A.34.1 Attribute: fastpmat

apply generalised DFT correction of L. Fritsche and Y. M. Gu, Phys. Rev. B 48, 4250 (1993)

Type: boolean
Default: "true"
Use: optional
XPath: /input/properties/momentummatrix/@fastpmat

A.35 Element: dielectric

Linear optical response (without local field effects, legacy version).

Contains: optcomp (A.36)
XPath: /input/properties/dielectric

This element allows for specification of the following attributes:

intraband (A.35.1), scissor (A.35.2), usegdft (A.35.3)

A.35.1 Attribute: intraband

The intraband attribute is "true" if the intraband term is to be added to the optical matrix (q=0)

Type: boolean
Default: "false"
Use: optional
XPath: /input/properties/dielectric/@intraband

A.35.2 Attribute: scissor

Type: fortrandouble (A.76.1)
Default: "0.0d0"
Use: optional
Unit: Hartree
XPath: /input/properties/dielectric/@scissor

A.35.3 Attribute: usegdft

apply generalised DFT correction of L. Fritsche and Y. M. Gu, Phys. Rev. B 48, 4250 (1993)

Type: boolean
Default: "false"
Use: optional
XPath: /input/properties/dielectric/@usegdft

A.36 Element: optcomp

The components of the first- or second-order optical tensor to be calculated.

Type: integertriple (A.76.6)
Default: "1 1 1"
XPath: /input/properties/dielectric/optcomp

A.37 Element: moke

Type: no content
XPath: `/input/properties/moke`

A.38 Element: expiqr

Contains: `kstlist` (A.72) (optional)
XPath: `/input/properties/expiqr`

A.39 Element: elnes

Type: no content
XPath: `/input/properties/elnes`

This element allows for specification of the following attributes:

`vecql` (A.39.1)

A.39.1 Attribute: vecql

Gives the q-vector in lattice coordinates for calculating ELNES.

Type: vect3d (A.76.4)
Default: "`0.0d0 0.0d0 0.0d0`"
Use: optional
XPath: `/input/properties/elnes/@vecql`

A.40 Element: eliashberg

Type: no content
XPath: `/input/properties/eliashberg`

This element allows for specification of the following attributes:

`mustar` (A.40.1)

A.40.1 Attribute: mustar

Coulomb pseudopotential, $\mu*$, used in the McMillan-Allen-Dynes equation.

Type: fortrandouble (A.76.1)
Default: "`0.15d0`"
Use: optional
XPath: `/input/properties/eliashberg/@mustar`

A.41 Element: phonons

Phonon frequencies and eigen vectors for an arbitrary q-point.

Contains:	`qpointset` (A.74) (optional) `phonondos` (A.42) (optional) `phonondispplot` (A.43) (optional) `reformatdynmat` (A.44) (optional) `interpolate` (A.45) (optional) `parts` (A.46) (optional)
XPath:	`/input/phonons`

This element allows for specification of the following attributes:

`ngridq` (A.41.3) (required), `deltaph` (A.41.1), `do` (A.41.2), `phonontype` (A.41.4), `reduceq` (A.41.5)

A.41.1 Attribute: deltaph

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

Type:	fortrandouble (A.76.1)
Default:	"`0.03d0`"
Use:	optional
XPath:	`/input/phonons/@deltaph`

A.41.2 Attribute: do

Decides if the phonon calculation is skipped or recalculated or continued from file.

Type:	**choose from:**
	`fromscratch`
	`skip`

Default:	"`fromscratch`"
Use:	optional
XPath:	`/input/phonons/@do`

A.41.3 Attribute: ngridq

Number of q grid points along the basis vector directions.

Type:	integertriple (A.76.6)
Use:	required
XPath:	`/input/phonons/@ngridq`

A.41.4 Attribute: phonontype

Decides which method (supercell or linear response) is used to perform the phonon calculation.

Type:	**choose from:**
	`supercell`
	`linearresponse`
Default:	"`supercell`"
Use:	optional
XPath:	`/input/phonons/@phonontype`

A.41.5 Attribute: reduceq

The attribute `reduceq` is set to `"true"` if the q-point set is to be reduced with the crystal symmetries.

Type:	boolean
Default:	"`true`"
Use:	optional
XPath:	`/input/phonons/@reduceq`

A.42 Element: phonondos

Phonon density of states.

Type:	no content
XPath:	`/input/phonons/phonondos`

This element allows for specification of the following attributes:

`ngrdos` (A.42.1), `nsmdos` (A.42.2), `ntemp` (A.42.3), `nwdos` (A.42.4)

A.42.1 Attribute: ngrdos

Type:	integer
Default:	"`100`"
Use:	optional
XPath:	`/input/phonons/phonondos/@ngrdos`

A.42.2 Attribute: nsmdos

Type:	integer
Default:	"`0`"
Use:	optional
XPath:	`/input/phonons/phonondos/@nsmdos`

A.42.3 Attribute: ntemp

Number of temperatures points for the calculation of the thermodynamical properties from the phonon density of states.

Type: integer
Default: "10"
Use: optional
XPath: /input/phonons/phonondos/@ntemp

A.42.4 Attribute: nwdos

Type: integer
Default: "500"
Use: optional
XPath: /input/phonons/phonondos/@nwdos

A.43 Element: phonondispplot

Phonon dispersion plot.

Contains: plot1d (A.66)
XPath: /input/phonons/phonondispplot

A.44 Element: reformatdynmat

Reads in the dynamical matrix rows from the corresponding files and outputs them in the DYNMAT*.OUT files, taking into account symmetrization and the accoustic sumrule.

Type: no content
XPath: /input/phonons/reformatdynmat

A.45 Element: interpolate

Interpolates the phonon frequencies on a given q-point set.

Type: no content
XPath: /input/phonons/interpolate

This element allows for specification of the following attributes:

ngridq (A.45.1) (required), vqloff (A.45.2), writeeigenvectors (A.45.3)

A.45.1 Attribute: ngridq

q-point grid for interpolation.

Type: integertriple (A.76.6)
Use: required
XPath: /input/phonons/interpolate/@ngridq

A.45.2 Attribute: vqloff

The q-point offset vector in lattice coordinates.

Type: vect3d (A.76.4)
Default: "0.0d0 0.0d0 0.0d0"
Use: optional
XPath: /input/phonons/interpolate/@vqloff

A.45.3 Attribute: writeeigenvectors

Set to true if the phonon eigenvectors are to be interpolated and output in addition to the phonon frequencies.

Type: boolean
Default: "false"
Use: optional
XPath: /input/phonons/interpolate/@writeeigenvectors

A.46 Element: parts

Contains: dopart (A.47) (zero or more)
XPath: /input/phonons/parts

A.47 Element: dopart

Type: no content
XPath: /input/phonons/parts/dopart

This element allows for specification of the following attributes:

id (A.47.1) (required)

A.47.1 Attribute: id

This attribute is used to trigger lower-level tasks and is mainly used for testing, debugging, and the testing of new features. Do not use it unless you know what you are doing.

Type: string
Use: required
XPath: /input/phonons/parts/dopart/@id

A.48 Element: xs

If this element is present with valid configuration, the macroscopic dielectric function and related spectroscopic quantities in the linear regime are calculated through either time-dependent DFT (TDDFT) or the Bethe-Salpeter equation (BSE).

Contains:	tddft (A.49) (optional) screening (A.50) (optional) BSE (A.51) (optional) transitions (A.52) (optional) qpointset (A.74) (1 times) tetra (A.59) (optional) energywindow (A.62) (1 times) plan (A.60) (optional)
XPath:	/input/xs

This element allows for specification of the following attributes:

xstype (A.48.27) (required), broad (A.48.1), dbglev (A.48.2), dfoffdiag (A.48.3), emattype (A.48.4), emaxdf (A.48.5), epsdfde (A.48.6), fastemat (A.48.7), fastpmat (A.48.8), gqmax (A.48.9), gqmaxtype (A.48.10), lmaxapw (A.48.11), lmaxapwwf (A.48.12), lmaxemat (A.48.13), lmaxmat (A.48.14), nempty (A.48.15), ngridk (A.48.16), ngridq (A.48.17), nosym (A.48.18), reducek (A.48.19), reduceq (A.48.20), rgkmax (A.48.21), scissor (A.48.22), swidth (A.48.23), tappinfo (A.48.24), tevout (A.48.25), vkloff (A.48.26)

A.48.1 Attribute: broad

Lorentzian broadening for all spectra

Type:	fortrandouble (A.76.1)
Default:	"0.01d0"
Use:	optional
Unit:	Hartree
XPath:	/input/xs/@broad

A.48.2 Attribute: dbglev

Debugging level. Any value > 0 will produce additional debug output. The large the value, the more information is output.

Type:	integer
Default:	"0"
Use:	optional
XPath:	/input/xs/@dbglev

A.48.3 Attribute: dfoffdiag

"true" if also off-diagonal tensor elements for the interacting response function are to be calculated.

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/xs/@dfoffdiag

A.48.4 Attribute: emattype

Type of matrix element generation (band-combinations). Should only be referenced for experimental features.

Type:	integer
Default:	"1"
Use:	optional
XPath:	/input/xs/@emattype

A.48.5 Attribute: emaxdf

Energy cutoff for the unoccupied states in the Kohn-Sham response function and screening. This parameter ensures a cutoff at the specified energy and is defined in addition to the nempty (A.50.2) parameter.

Type:	fortrandouble (A.76.1)
Default:	"1.0d10"
Use:	optional
XPath:	/input/xs/@emaxdf

A.48.6 Attribute: epsdfde

The smallest energy difference for which the square of its inverse will be considered in the Kohn-Sham response function.

Type:	fortrandouble (A.76.1)
Default:	"1.0d-8"
Use:	optional
Unit:	Hartree
XPath:	/input/xs/@epsdfde

A.48.7 Attribute: fastemat

If set to "true", a fast method to calculate APW-lo, lo-APW and lo-lo parts of the **q**-dependent matrix elements in the muffin-tin is used.

Type:	boolean
Default:	"true"
Use:	optional
XPath:	/input/xs/@fastemat

A.48.8 Attribute: fastpmat

If set to "true", a fast method to calculate APW-lo, lo-APW and lo-lo parts of the momentum matrix elements in the muffin-tin is used.

Type:	boolean
Default:	"true"
Use:	optional
XPath:	/input/xs/@fastpmat

A.48.9 Attribute: gqmax

$|\mathbf{G}+\mathbf{q}|$ cutoff for Kohn-Sham response function, screening and for expansion of Coulomb potential

Type:	fortrandouble (A.76.1)
Default:	"0.0d0"
Use:	optional
XPath:	/input/xs/@gqmax

A.48.10 Attribute: gqmaxtype

Defines the way the gqmax cutoff is applied for the selection of the G-vectors. For "$|G+q|$" G vectors are selected such that $\mathbf{G}+\mathbf{q}$ lies within the gqmax (A.48.9) cutoff. For "$|G|$" G vectors are selected such that $\mathbf{G}$ lies within the gqmax cutoff.

Type:	**choose from:**
	|G+q|
	|G|
Default:	"|G+q|"
Use:	optional
XPath:	/input/xs/@gqmaxtype

A.48.11 Attribute: lmaxapw

Angular momentum cut-off for the APW functions.

Type:	integer
Default:	"10"
Use:	optional
XPath:	/input/xs/@lmaxapw

A.48.12 Attribute: lmaxapwwf

Maximum angular momentum for APW functions for q-dependent matrix elements.

Type:	integer
Default:	"-1"
Use:	optional
XPath:	/input/xs/@lmaxapwwf

A.48.13 Attribute: lmaxemat

Maximum angular momentum for Rayleigh expansion of $\mathbf{q}$-dependent plane wave factor.

Type:	integer
Default:	"3"
Use:	optional
XPath:	/input/xs/@lmaxemat

A.48.14 Attribute: lmaxmat

Angular momentum cut-off for the outer-most loop in the hamiltonian and overlap matrix setup.

Type: integer
Default: "5"
Use: optional
XPath: `/input/xs/@lmaxmat`

A.48.15 Attribute: nempty

Number of empty states. This parameter determines the energy cutoff for the excitation spectra. For determining the number of states related to an energy cutoff, perform one iteration of a SCF calculation, setting `nempty` (A.50.2) to a higher value and check the `EIGVAL.OUT`.

Type: integer
Default: "5"
Use: optional
XPath: `/input/xs/@nempty`

A.48.16 Attribute: ngridk

k-point grid sizes.

Type: integertriple (A.76.6)
Default: "1 1 1"
Use: optional
XPath: `/input/xs/@ngridk`

A.48.17 Attribute: ngridq

q-point grid sizes.

Type: integertriple (A.76.6)
Default: "1 1 1"
Use: optional
XPath: `/input/xs/@ngridq`

A.48.18 Attribute: nosym

`nosym` (A.51.8) is `"true"` if no symmetry information should be used

Type: boolean
Default: "false"
Use: optional
XPath: `/input/xs/@nosym`

A.48.19 Attribute: reducek

reducek (A.51.11) is "true" if **k**-points are to be reduced (with crystal symmetries).

Type: boolean
Default: "false"
Use: optional
XPath: /input/xs/@reducek

A.48.20 Attribute: reduceq

reducek (A.51.11) is "true" if q-points are to be reduced (with crystal symmetries).

Type: boolean
Default: "true"
Use: optional
XPath: /input/xs/@reduceq

A.48.21 Attribute: rgkmax

Smallest muffin-tin radius times gkmax.

Type: fortrandouble (A.76.1)
Default: "7.0d0"
Use: optional
XPath: /input/xs/@rgkmax

A.48.22 Attribute: scissor

Scissors correction to correct the conduction band energies.

Type: fortrandouble (A.76.1)
Default: "0.0d0"
Use: optional
Unit: Hartree
XPath: /input/xs/@scissor

A.48.23 Attribute: swidth

Width of the smooth approximation to the Dirac delta function (must be ¿ 0).

Type: fortrandouble (A.76.1)
Default: "0.001d0"
Use: optional
Unit: Hartree
XPath: /input/xs/@swidth

A.48.24 Attribute: tappinfo

"true" to append info to output file.

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/xs/@tappinfo

A.48.25 Attribute: tevout

"true" if energy outputs are in eV.

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/xs/@tevout

A.48.26 Attribute: vkloff

The **k**-point set offset. All **k**-points of a regular **k**-mesh (a mesh containing the Gamma point) are shifted by a constant vector given by $(\text{vkloff}_1/N_1, \text{vkloff}_2/N_2, \text{vkloff}_3/N_3)$, where (N_1, N_2, N_3) is the division of the **k**-point mesh. It should be selected such that all symmetries among the **k**-points from the regular (non-shifted) mesh are broken. An exception is the case of optical spectra without local field effects where symmetries among **k**-points are explicitly taken into account.

Type:	vect3d (A.76.4)
Default:	"0.0d0 0.0d0 0.0d0 "
Use:	optional
XPath:	/input/xs/@vkloff

A.48.27 Attribute: xstype

Should TDDFT be used or BSE.

Type:	**choose from:**
	TDDFT
	BSE
Use:	required
XPath:	/input/xs/@xstype

A.49 Element: tddft

Type:	no content
XPath:	/input/xs/tddft

This element allows for specification of the following attributes:

acont (A.49.1), alphalrc (A.49.2), alphalrcdyn (A.49.3), aresdf (A.49.4), aresfxc (A.49.5), betalrcdyn (A.49.6), do (A.49.7), fxcbsesplit (A.49.8), fxctype (A.49.9), intraband (A.49.10), kerndiag (A.49.11), lindhard (A.49.12), lmaxalda (A.49.13), mdfqtype (A.49.14), nwacont (A.49.15), torddf (A.49.16), tordfxc (A.49.17)

A.49.1 Attribute: acont

Set to "true" if analytic continuation from the imaginary axis to the real axis is to be performed.

Type: boolean
Default: "false"
Use: optional
XPath: /input/xs/tddft/@acont

A.49.2 Attribute: alphalrc

α-parameter for the static long range contribution (LRC) model xc kernel.

Type: fortrandouble (A.76.1)
Default: "0.0d0"
Use: optional
XPath: /input/xs/tddft/@alphalrc

A.49.3 Attribute: alphalrcdyn

α-parameter for the dynamical long range contribution (LRC) model xc kernel.

Type: fortrandouble (A.76.1)
Default: "0.0d0"
Use: optional
XPath: /input/xs/tddft/@alphalrcdyn

A.49.4 Attribute: aresdf

Set to "true" if to consider the anti-resonant part for the dielectric function.

Type: boolean
Default: "true"
Use: optional
XPath: /input/xs/tddft/@aresdf

A.49.5 Attribute: aresfxc

Set to "true" if to consider the anti-resonant part for the MBPT derived xc-kernels.

Type: boolean
Default: "true"
Use: optional
XPath: /input/xs/tddft/@aresfxc

A.49.6 Attribute: betalrcdyn

β-parameter for the dynamical long range contribution (LRC) model xc kernel.

Type:	fortrandouble (A.76.1)
Use:	optional
XPath:	`/input/xs/tddft/@betalrcdyn`

A.49.7 Attribute: do

Decides if the TDDFT calculation is to be resumed starting from a new xc kernel or is to be skipped.

Type:	**choose from:**
	`fromscratch`
	`fromkernel`
Default:	"`fromscratch`"
Use:	optional
XPath:	`/input/xs/tddft/@do`

A.49.8 Attribute: fxcbsesplit

Split parameter for degeneracy in energy differences of MBPT derived xc kernels. See A. Marini, Phys. Rev. Lett., 91, (2003) 256402.

Type:	fortrandouble (A.76.1)
Default:	"`1.0d-5`"
Use:	optional
Unit:	Hartree
XPath:	`/input/xs/tddft/@fxcbsesplit`

A.49.9 Attribute: fxctype

Defines which xc kernel is to be used.

Type:	**choose from:**
	`RPA`
	`LRCstatic_NLF`
	`LRCstatic`
	`LRCdyn_NLF`
	`LRCdyn`
	`ALDA`
	`MB1_NLF`
	`MB1`
Default:	"`RPA`"
Use:	optional
XPath:	`/input/xs/tddft/@fxctype`

A.49.10 Attribute: intraband

The intraband attribute is "true" if the intraband term is to be added to the optical matrix (q=0).

Type: boolean
Default: "false"
Use: optional
XPath: /input/xs/tddft/@intraband

A.49.11 Attribute: kerndiag

Set to "true" if only diagonal part of xc-kernel is to be used.

Type: boolean
Default: "false"
Use: optional
XPath: /input/xs/tddft/@kerndiag

A.49.12 Attribute: lindhard

Set to "true" if Lindhard-like function is to be calculated.

Type: boolean
Default: "false"
Use: optional
XPath: /input/xs/tddft/@lindhard

A.49.13 Attribute: lmaxalda

Angular momentum cutoff for Rayleigh expansion of exponential factor for ALDA-kernel.

Type: integer
Default: "3"
Use: optional
XPath: /input/xs/tddft/@lmaxalda

A.49.14 Attribute: mdfqtype

Treatment of macroscopic dielectric function for $\mathbf{Q}$-point outside of Brillouin zone. A value of 0 uses the full $\mathbf{Q}$ and the $(\mathbf{0},\mathbf{0})$ component of the microscopic dielectric matrix is used. A value of 1 invokes a decomposition $\mathbf{Q} = \mathbf{q} + \mathbf{G_q}$ and the $(\mathbf{Q_q},\mathbf{Q_q})$ component of the microscopic dielectric matrix is used.

Type: integer
Default: "0"
Use: optional
XPath: /input/xs/tddft/@mdfqtype

A.49.15 Attribute: nwacont

Number of energy intervals (on imaginary axis) for analytic continuation.

Type:	integer
Default:	"0"
Use:	optional
XPath:	`/input/xs/tddft/@nwacont`

A.49.16 Attribute: torddf

Set to `"true"` if to consider the time-ordered version of the dielectric function.

Type:	boolean
Default:	"`false`"
Use:	optional
XPath:	`/input/xs/tddft/@torddf`

A.49.17 Attribute: tordfxc

Set to `"true"` if to consider the time-ordered version of xc kernel (MBPT derived kernels only).

Type:	boolean
Default:	"`false`"
Use:	optional
XPath:	`/input/xs/tddft/@tordfxc`

A.50 Element: screening

Type:	no content
XPath:	`/input/xs/screening`

This element allows for specification of the following attributes:

`do` (A.50.1), `nempty` (A.50.2), `ngridk` (A.50.3), `nosym` (A.50.4), `reducek` (A.50.5), `rgkmax` (A.50.6), `screentype` (A.50.7), `vkloff` (A.50.8)

A.50.1 Attribute: do

Decides if the calculation of the screening is done from scratch or is to be skipped.

Type:	**choose from:**
	`fromscratch`
	`skip`
Default:	"`fromscratch`"
Use:	optional
XPath:	`/input/xs/screening/@do`

A.50.2 Attribute: nempty

Number of empty states.

Type:	integer
Default:	"0"
Use:	optional
XPath:	/input/xs/screening/@nempty

A.50.3 Attribute: ngridk

k-point grid sizes for screening.

Type:	integertriple (A.76.6)
Default:	"0 0 0"
Use:	optional
XPath:	/input/xs/screening/@ngridk

A.50.4 Attribute: nosym

nosym (A.51.8) is "true" if no symmetry information should be used for screening.

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/xs/screening/@nosym

A.50.5 Attribute: reducek

reducek (A.51.11) is "true" if **k**-points are to be reduced with crystal symmetries for screening.

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/xs/screening/@reducek

A.50.6 Attribute: rgkmax

The smallest muffin-tin radius times gkmax for screening.

Type:	fortrandouble (A.76.1)
Default:	"0.0d0"
Use:	optional
XPath:	/input/xs/screening/@rgkmax

A.50.7 Attribute: screentype

Defines which type of screening is to be used.

Type:	**choose from:**
	`full`
	`diag`
	`noinvdiag`
	`longrange`
Default:	"`full`"
Use:	optional
XPath:	`/input/xs/screening/@screentype`

A.50.8 Attribute: vkloff

k-point offset for screening.

Type:	vect3d (A.76.4)
Default:	"`-1.0d0 -1.0d0 -1.0d0`"
Use:	optional
XPath:	`/input/xs/screening/@vkloff`

A.51 Element: BSE

Type:	no content
XPath:	`/input/xs/BSE`

This element allows for specification of the following attributes:

`aresbse` (A.51.1), `bsedirsing` (A.51.2), `bsetype` (A.51.3), `fbzq` (A.51.4), `lmaxdielt` (A.51.5), `nexcitmax` (A.51.6), `nleblaik` (A.51.7), `nosym` (A.51.8), `nstlbse` (A.51.9), `nstlbsemat` (A.51.10), `reducek` (A.51.11), `rgkmax` (A.51.12), `sciavbd` (A.51.13), `sciavqbd` (A.51.14), `sciavqhd` (A.51.15), `sciavqwg` (A.51.16), `sciavtype` (A.51.17), `scrherm` (A.51.18), `vkloff` (A.51.19)

A.51.1 Attribute: aresbse

Is set to `"true"` if to consider the anti-resonant part for the BSE spectrum.

Type:	boolean
Default:	"`true`"
Use:	optional
XPath:	`/input/xs/BSE/@aresbse`

A.51.2 Attribute: bsedirsing

`"true"` if effective singular part of direct term of BSE Hamiltonian is to be used.

Type:	boolean
Default:	"`false`"
Use:	optional
XPath:	`/input/xs/BSE/@bsedirsing`

A.51.3 Attribute: bsetype

Defines which parts of the BSE Hamiltonian are to be considered.

Type:	**choose from:**
	`IP`
	`RPA`
	`singlet`
	`triplet`
Default:	"`singlet`"
Use:	optional
XPath:	`/input/xs/BSE/@bsetype`

A.51.4 Attribute: fbzq

Set to `"true"` if q-point set is to be taken from the first Brillouin zone.

Type:	boolean
Default:	"`false`"
Use:	optional
XPath:	`/input/xs/BSE/@fbzq`

A.51.5 Attribute: lmaxdielt

Angular momentum cutoff of the spherical harmonics expansion of the dielectric matrix.

Type:	integer
Default:	"`14`"
Use:	optional
XPath:	`/input/xs/BSE/@lmaxdielt`

A.51.6 Attribute: nexcitmax

Maximum number of excitons to be considered in a BSE calculation.

Type:	integer
Default:	"`100`"
Use:	optional
XPath:	`/input/xs/BSE/@nexcitmax`

A.51.7 Attribute: nleblaik

Number of points used for the Lebedev-Laikov grids must be selected according to V.I. Lebedev, and D.N. Laikov, Doklady Mathematics, 59 (1999) 477.

Type:	integer
Default:	"`5810`"
Use:	optional
XPath:	`/input/xs/BSE/@nleblaik`

A.51.8 Attribute: nosym

Set to "true" if no symmetry information should be used for BSE.

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/xs/BSE/@nosym

A.51.9 Attribute: nstlbse

Range of bands included for the BSE calculation. The first pair of numbers corresponds to the band index for local orbitals and valence states (counted from the lowest eigenenergy), the second pair corresponds to the band index of the conduction states (counted from the Fermi level).

Type:	integerquadrupel (A.76.7)
Default:	"0 0 0 0"
Use:	optional
XPath:	/input/xs/BSE/@nstlbse

A.51.10 Attribute: nstlbsemat

Range of bands for calculating the screening and matrix elements needed for solving the BSE. The first pair of numbers corresponds to the band index for local orbitals and valence states (counted from the lowest eigenenergy), the second pair corresponds to the band index of the conduction states (counted from the Fermi level).

Type:	integerquadrupel (A.76.7)
Default:	"0 0 0 0"
Use:	optional
XPath:	/input/xs/BSE/@nstlbsemat

A.51.11 Attribute: reducek

reducek (A.51.11) is "true" if **k**-points are to be reduced with crystal symmetries for BSE.

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/xs/BSE/@reducek

A.51.12 Attribute: rgkmax

Smallest muffin-tin radius times gkmax.

Type:	fortrandouble (A.76.1)
Default:	"0.0d0"
Use:	optional
XPath:	/input/xs/BSE/@rgkmax

A.51.13 Attribute: sciavbd

"true" if the body of the screened Coulomb interaction is to be averaged (q=0).

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/xs/BSE/@sciavbd

A.51.14 Attribute: sciavqbd

"true" if the body of the screened Coulomb interaction is to be averaged (q!=0).

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/xs/BSE/@sciavqbd

A.51.15 Attribute: sciavqhd

"true" if the head of the screened Coulomb interaction is to be averaged (q!=0).

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/xs/BSE/@sciavqhd

A.51.16 Attribute: sciavqwg

"true" if the wings of the screened Coulomb interaction are to be averaged (q!=0).

Type:	boolean
Default:	"false"
Use:	optional
XPath:	/input/xs/BSE/@sciavqwg

A.51.17 Attribute: sciavtype

Defines how the screened Coulomb interaction matrix is to be averaged (important for the singular terms).

Type:	**choose from:**
	spherical
	screendiag
	invscreendiag
Default:	"spherical"
Use:	optional
XPath:	/input/xs/BSE/@sciavtype

A.51.18 Attribute: scrherm

Method of how an almost Hermitian matrix is inverted. A value of 0: invert full matrix (matrix is allowed to be not strictly Hermitian); a value of 1: take the Hermitian average for inversion; a value of 2: assume Hermitian and use the upper triangle; a value of 3: assume Hermitian and use the lower triangle.

Type: integer
Default: "0"
Use: optional
XPath: `/input/xs/BSE/@scrherm`

A.51.19 Attribute: vkloff

k-point offset for BSE.

Type: vect3d (A.76.4)
Default: "`-1.0d0 -1.0d0 -1.0d0`"
Use: optional
XPath: `/input/xs/BSE/@vkloff`

A.52 Element: transitions

Describe transitions between Kohn-Sham states for the calculation of the Kohn-Sham response function (and screening) here. Individual transitions as well as a range (or a list) of initial and final states can be defined.

Contains: `individual` (A.53) (optional)
`ranges` (A.55) (optional)
`lists` (A.57) (optional)
XPath: `/input/xs/transitions`

A.53 Element: individual

A list of individual transitions consisting of an initial state a final state and a **k**-point is given here. If the list is empty, no transitions are considered.

Contains: `trans` (A.54) (zero or more)
XPath: `/input/xs/transitions/individual`

A.54 Element: trans

An individual transition consisting of an initial state a final state and a **k**-point is given here. Values of zero correspond to the inclusion of all initial and final states and all **k**-points and can be used as "wildcards" (default). Therefore, an empty element amounts to include all transitions.

Type: no content
XPath: `/input/xs/transitions/individual/trans`

This element allows for specification of the following attributes:

`action` (A.54.1), `final` (A.54.2), `initial` (A.54.3), `kpointnumber` (A.54.4)

A.54.1 Attribute: action

Select to include or exclude states. If a state is included as well as excluded several times the last definition (in the sequence of individual transitions) counts.

Type:	**choose from:**
	`include`
	`exclude`
Default:	"`include`"
Use:	optional
XPath:	`/input/xs/transitions/individual/trans/@action`

A.54.2 Attribute: final

Final state of individual transition. A value of zero (default) means to include all states.

Type:	integer
Default:	"`0`"
Use:	optional
XPath:	`/input/xs/transitions/individual/trans/@final`

A.54.3 Attribute: initial

Initial state of individual transition. A value of zero (default) means to include all states.

Type:	integer
Default:	"`0`"
Use:	optional
XPath:	`/input/xs/transitions/individual/trans/@initial`

A.54.4 Attribute: kpointnumber

Number of **k**-points to be considered. A value of zero (default) means to include all **k**-points.

Type:	integer
Default:	"`0`"
Use:	optional
XPath:	`/input/xs/transitions/individual/trans/@kpointnumber`

A.55 Element: ranges

A list of ranges of transitions (initial state as well as final state ranges) and a **k**-point are given here. An empty list amounts to no transitions at all.

Contains:	`range` (A.56) (zero or more)
XPath:	`/input/xs/transitions/ranges`

A.56 Element: range

A range of transitions (for initial as well as final states) is given here. A range consists of a "start" and a "stop" values as well as a **k**-point. Values of zero correspond to starting at the first state and stopping at the last state and considering all **k**-points. They can be used as "wildcards" (default). Therefore, an empty element corresponds to the full initial/final state range for all **k**-points.

Type: no content
XPath: `/input/xs/transitions/ranges/range`

This element allows for specification of the following attributes:

`statestype` (A.56.4) (required), `action` (A.56.1), `kpointnumber` (A.56.2), `start` (A.56.3), `stop` (A.56.5)

A.56.1 Attribute: action

Select to include or exclude states. If a state is included as well as excluded several times the last definition (in the sequence of individual transitions) counts.

Type: **choose from:**
`include`
`exclude`
Default: "`include`"
Use: optional
XPath: `/input/xs/transitions/ranges/range/@action`

A.56.2 Attribute: kpointnumber

Number of **k**-point to be considered. A value of zero (default) means to include all **k**-point.

Type: integer
Default: "`0`"
Use: optional
XPath: `/input/xs/transitions/ranges/range/@kpointnumber`

A.56.3 Attribute: start

Start value (first state) for range. A value of zero (default) means to start from the first state.

Type: integer
Default: "`0`"
Use: optional
XPath: `/input/xs/transitions/ranges/range/@start`

A.56.4 Attribute: statestype

Select for initial or final state range.

Type:	**choose from:**
	`initialstates`
	`finalstates`
Use:	required
XPath:	`/input/xs/transitions/ranges/range/@statestype`

A.56.5 Attribute: stop

Stop value (last state) for range. A value of zero (default) means to stop at the last state (no upper limit).

Type:	integer
Default:	"0"
Use:	optional
XPath:	`/input/xs/transitions/ranges/range/@stop`

A.57 Element: lists

A list of initial and final state entries to be considered for transitions. An empty list amounts to no transitions at all.

Contains:	`istate` (A.58) (zero or more)
XPath:	`/input/xs/transitions/lists`

A.58 Element: istate

An initial or final state and corresponding **k**-point is given here. Values of zero correspond to considering all initial/final states for all **k**-points. They can be used as "wildcards" (default). Therefore, an empty element corresponds to the full initial/final state set for all **k**-points.

Type:	no content
XPath:	`/input/xs/transitions/lists/istate`

This element allows for specification of the following attributes:

`statestype` (A.58.4) (required), `action` (A.58.1), `kpointnumber` (A.58.2), `state` (A.58.3)

A.58.1 Attribute: action

Select to include or exclude states. If a state is included as well as excluded several times the last definition (in the sequence of individual transitions) counts.

Type:	**choose from:**
	`include`
	`exclude`
Default:	"`include`"
Use:	optional
XPath:	`/input/xs/transitions/lists/istate/@action`

A.58.2 Attribute: kpointnumber

Number of **k**-point to be consider. A value of zero (default) means to include all **k**-point.

Type:	integer
Default:	"`0`"
Use:	optional
XPath:	`/input/xs/transitions/lists/istate/@kpointnumber`

A.58.3 Attribute: state

The state to be considered. A value of zero (default) means to include all states.

Type:	integer
Default:	"`0`"
Use:	optional
XPath:	`/input/xs/transitions/lists/istate/@state`

A.58.4 Attribute: statestype

Select for initial or final state list.

Type:	**choose from:**
	`initialstates`
	`finalstates`
Use:	required
XPath:	`/input/xs/transitions/lists/istate/@statestype`

A.59 Element: tetra

Type:	no content
XPath:	`/input/xs/tetra`

This element allows for specification of the following attributes:

`cw1k` (A.59.1), `kordexc` (A.59.2), `qweights` (A.59.3), `tetradf` (A.59.4), `tetraocc` (A.59.5)

A.59.1 Attribute: cw1k

Type: boolean
Default: "false"
Use: optional
XPath: /input/xs/tetra/@cw1k

A.59.2 Attribute: kordexc

Type: boolean
Default: "false"
Use: optional
XPath: /input/xs/tetra/@kordexc

A.59.3 Attribute: qweights

Choice of weights and nodes for the tetrahedron method and non-zero Q-point.

Type: integer
Default: "1"
Use: optional
XPath: /input/xs/tetra/@qweights

A.59.4 Attribute: tetradf

"true" if tetrahedron method is used for the **k**-space integration in the Kohn-Sham response function.

Type: boolean
Default: "false"
Use: optional
XPath: /input/xs/tetra/@tetradf

A.59.5 Attribute: tetraocc

Type: boolean
Default: "false"
Use: optional
XPath: /input/xs/tetra/@tetraocc

A.60 Element: plan

Contains: doonly (A.61) (zero or more)
XPath: /input/xs/plan

A.61 Element: doonly

Type: no content
XPath: `/input/xs/plan/doonly`

This element allows for specification of the following attributes:

`task` (A.61.1) (required)

A.61.1 Attribute: task

Type: **choose from:**
`xsgeneigvec`
`tetcalccw`
`writepmatxs`
`writeemat`
`df`
`df2`
`idf`
`scrgeneigvec`
`scrtetcalccw`
`scrwritepmat`
`screen`
`scrcoulint`
`exccoulint`
`bse`
`kernxc_bse`
`writebandgapgrid`
`writepmat`
`dielectric`
`writepmatasc`
`pmatxs2orig`
`writeematasc`
`writepwmat`
`emattest`
`x0toasc`
`x0tobin`
`fxc_alda_check`
`kernxc_bse3`
`testxs`
`xsestimate`
`xstiming`
`testmain`
`portstate(1)`
`portstate(2)`
`portstate(-1)`
`portstate(-2)`
Use: required
XPath: `/input/xs/plan/doonly/@task`

A.62 Element: energywindow

Type: no content
XPath: `/input/xs/energywindow`

This element allows for specification of the following attributes:

`intv` (A.62.1), `points` (A.62.2)

A.62.1 Attribute: intv

energy interval lower and upper limits.

Type: vect2d (A.76.5)
Default: "`-0.5d0 0.5d0`"
Use: optional
XPath: `/input/xs/energywindow/@intv`

A.62.2 Attribute: points

number of points to be sampled linearly inside the energy interval including the lower limit.

Type: integer
Default: "`500`"
Use: optional
XPath: `/input/xs/energywindow/@points`

A.63 Reused Elements

The following elements can occur more than once in the input file. There for they are listed separately.

A.64 Element: origin

Type: no content
XPath: `/origin`
Parent: `/plot2d/parallelogram`
`/plot3d/box`

This element allows for specification of the following attributes:

`coord` (A.64.1)

A.64.1 Attribute: coord

Type: vect3d (A.76.4)
Use: optional
XPath: `/origin/@coord`

A.65 Element: point

Type: no content
XPath: /point
Parent: /plot1d/path
/plot2d/parallelogram
/plot3d/box

This element allows for specification of the following attributes:

coord (A.65.1) (required), label (A.65.2)

A.65.1 Attribute: coord

Type: vect3d (A.76.4)
Use: required
XPath: /point/@coord

A.65.2 Attribute: label

Type: string
Default: ""
Use: optional
XPath: /point/@label

A.66 Element: plot1d

The element plot1d specifies sample points along a path. The coordinate space (lattice or cartesian)is chosen in the context of the parent.

Contains: path (A.67) (1 times)
XPath: /plot1d
Parent: /input/phonons/phonondispplot
/input/properties/bandstructure
/input/properties/wfplot
/input/properties/chargedensityplot
/input/properties/exccplot
/input/properties/elfplot
/input/properties/gradmvecfield

A.67 Element: path

Contains: point (A.65) (2 times or more)
XPath: /plot1d/path

This element allows for specification of the following attributes:

steps (A.67.2) (required), outfileprefix (A.67.1)

A.67.1 Attribute: outfileprefix

Type: string
Use: optional
XPath: `/plot1d/path/@outfileprefix`

A.67.2 Attribute: steps

Type: integer
Use: required
XPath: `/plot1d/path/@steps`

A.68 Element: plot2d

Defines a 2d plot domain.

Contains: `parallelogram` (A.69) (1 times)
XPath: `/plot2d`
Parent: `/input/properties/STM`
`/input/properties/wfplot`
`/input/properties/chargedensityplot`
`/input/properties/exccplot`
`/input/properties/elfplot`
`/input/properties/mvecfield`
`/input/properties/xcmvecfield`
`/input/properties/electricfield`
`/input/properties/gradmvecfield`

A.69 Element: parallelogram

Contains: `origin` (A.64) (1 times)
`point` (A.65) (2 times)
XPath: `/plot2d/parallelogram`

This element allows for specification of the following attributes:

`grid` (A.69.1) (required), `outfileprefix` (A.69.2)

A.69.1 Attribute: grid

Type: integerpair (A.76.8)
Use: required
XPath: `/plot2d/parallelogram/@grid`

A.69.2 Attribute: outfileprefix

Type: string
Use: optional
XPath: `/plot2d/parallelogram/@outfileprefix`

A.70 Element: plot3d

Defines a 3d plot domain.

Contains:	box (A.71) (1 times)
XPath:	/plot3d
Parent:	/input/properties/wfplot /input/properties/chargedensityplot /input/properties/exccplot /input/properties/elfplot /input/properties/mvecfield /input/properties/xcmvecfield /input/properties/electricfield /input/properties/gradmvecfield

A.71 Element: box

Contains:	origin (A.64) (1 times) point (A.65) (3 times)
XPath:	/plot3d/box

This element allows for specification of the following attributes:

grid (A.71.1) (required), outfileprefix (A.71.2)

A.71.1 Attribute: grid

Type:	integertriple (A.76.6)
Use:	required
XPath:	/plot3d/box/@grid

A.71.2 Attribute: outfileprefix

Type:	string
Use:	optional
XPath:	/plot3d/box/@outfileprefix

A.72 Element: kstlist

The kstlist element is used in the LSJ and wavefunction plot element This is a user-defined list of **k**-point and state index pairs which are those used for plotting wavefunctions and writing **L**, **S** and **J** expectation values.

Contains:	pointstatepair (A.73) (1 times or more)
XPath:	/kstlist
Parent:	/input/properties/wfplot /input/properties/LSJ /input/properties/expiqr

A.73 Element: pointstatepair

The element `pointstatepair` defines a **k**-point and state index pair.

Type: integerpair (A.76.8)
XPath: `/kstlist/pointstatepair`

A.74 Element: qpointset

Contains: `qpoint` (A.75) (1 times or more)
XPath: `/qpointset`
Parent: `/input/phonons`
`/input/xs`

A.75 Element: qpoint

a q-point is given in reciprocal space coordinates

Type: vect3d (A.76.4)
XPath: `/qpointset/qpoint`

A.76 Data Types

The Input definition uses derived data types. These are described here.

A.76.1 Type fortrandouble

The type `fortrandouble` allows to use the letters `"eEdDqQ"` for exponent operators. This alters in what precision the number is parsed.

A.76.2 Type vector

A vector is a space separated list of floating point numbers.
Example: `"1.3 2.3e4 3 90"`

A.76.3 Type integerlist

List of space separated integers.

A.76.4 Type vect3d

Three dimensional vector as three space separated floating point numbers.

A.76.5 Type vect2d

Three dimensional vector as three space separated floating point numbers.

A.76.6 Type integertriple

Space separated list of three integers.
Example: `"1 2 3"`

A.76.7 Type integerquadrupel

Space separated list of three integers.
Example: `"1 2 3 4"`

A.76.8 Type integerpair

Space separated list of two integers
Example: `"1 2"`

Appendix B

Species file format reference

exciting developers team
(C. Ambrosch-Draxl, Zohreh Basirat, Thomas Dengg,
Rostam Golesorkhtabar, Christian Meisenbichler, Dmitrii Nabok,
Weine Olovsson, Pasquale Pavone, Stephan Sagmeister, Jürgen Spitaler)

About this Document

This document describes the file format for the species definitions.

B.1 Input Elements

B.2 Element: spdb

Species-database element contains the species element sp (B.3)

Contains: sp (B.3)
XPath: /spdb

B.3 Element: sp

A species is an atom type definition containing all information to construct the basis functions.

Contains: muffinTin (B.4) (1 times)
atomicState (B.5) (1 times or more)
basis (B.6) (1 times)
lorb (B.8) (zero or more)
XPath: /spdb/sp

This element allows for specification of the following attributes:

chemicalSymbol (B.3.1) (required), mass (B.3.2) (required), z (B.3.4) (required), name (B.3.3)

B.3.1 Attribute: chemicalSymbol

Chemical Symbol.

Type: ID
Use: required
XPath: `/spdb/sp/@chemicalSymbol`

B.3.2 Attribute: mass

Mass in m_e.

Type: fortrandouble (B.11.1)
Use: required
XPath: `/spdb/sp/@mass`

B.3.3 Attribute: name

Optional element name.

Type: string
Use: optional
XPath: `/spdb/sp/@name`

B.3.4 Attribute: z

Atomic number.

Type: fortrandouble (B.11.1)
Use: required
XPath: `/spdb/sp/@z`

B.4 Element: muffinTin

This element gives the size of the muffin tin radius and the resolution of the radial functions.

Type: no content
XPath: `/spdb/sp/muffinTin`

This element allows for specification of the following attributes:

`radialmeshPoints` (B.4.1) (required), `radius` (B.4.2) (required), `rinf` (B.4.3) (required), `rmin` (B.4.4) (required)

B.4.1 Attribute: radialmeshPoints

Number of data points for radial atomic functions.

Type: integer
Use: required
XPath: `/spdb/sp/muffinTin/@radialmeshPoints`

B.4.2 Attribute: radius

The radius of the muffin tin sphere.

Type: fortrandouble (B.11.1)
Use: required
XPath: `/spdb/sp/muffinTin/@radius`

B.4.3 Attribute: rinf

Radius from which the influence on the potential is regarded to be negligible.

Type: fortrandouble (B.11.1)
Use: required
XPath: `/spdb/sp/muffinTin/@rinf`

B.4.4 Attribute: rmin

The radius where radial mesh begins.

Type: fortrandouble (B.11.1)
Use: required
XPath: `/spdb/sp/muffinTin/@rmin`

B.5 Element: atomicState

The `atomicState` (B.5) element lists the atomic states that schould be used to approximate the wavefunction in the sphere. They can be marked as core or none core electrons by the `core` (B.5.1) attribute. Core electrons are threated separately by numeric itegration.

Type: no content
XPath: `/spdb/sp/atomicState`

This element allows for specification of the following attributes:

`core` (B.5.1) (required), `kappa` (B.5.2) (required), `l` (B.5.3) (required), `n` (B.5.4) (required), `occ` (B.5.5) (required)

B.5.1 Attribute: core

If `true`, state is threated as core state in the calculation.

Type: boolean
Use: required
XPath: `/spdb/sp/atomicState/@core`

B.5.2 Attribute: kappa

Relativistic quantum number.

Type:	integer
Use:	required
XPath:	`/spdb/sp/atomicState/@kappa`

B.5.3 Attribute: l

Azimuthal quantum number.

Type:	integer
Use:	required
XPath:	`/spdb/sp/atomicState/@l`

B.5.4 Attribute: n

Principal quantum number.

Type:	integer
Use:	required
XPath:	`/spdb/sp/atomicState/@n`

B.5.5 Attribute: occ

Ocuppation number.

Type:	fortrandouble (B.11.1)
Use:	required
XPath:	`/spdb/sp/atomicState/@occ`

B.6 Element: basis

Defines APW basis.

Contains:	`wf` (B.10) (1 times or more)
	`exception` (B.7) (zero or more)
XPath:	`/spdb/sp/basis`

This element allows for specification of the following attributes:

`order` (B.6.1) (required)

B.6.1 Attribute: order

Type:	integer
Use:	required
XPath:	`/spdb/sp/basis/@order`

B.7 Element: exception

This element allows for defining exceptions to the APW basis functions.

Contains: wf (B.10) (1 times or more)
XPath: /spdb/sp/basis/exception

This element allows for specification of the following attributes:

l (B.7.1)

B.7.1 Attribute: l

Spezifies the azimuthal quantum number for which the exception applies.

Type: integer
Use: optional
XPath: /spdb/sp/basis/exception/@l

B.8 Element: lorb

Local orbital (APW+lo or LAPW+lo).

Contains: wf (B.10) (1 times or more)
XPath: /spdb/sp/lorb

This element allows for specification of the following attributes:

l (B.8.1) (required)

B.8.1 Attribute: l

Azimuthal quantum number for which the local orbital is defined.

Type: integer
Use: required
XPath: /spdb/sp/lorb/@l

B.9 Reused Elements

The following elements can occur more than once in the input file. There for they are listed separately.

B.10 Element: wf

Defines the radial part of an atomic wavefunction. This functions used to construct a lapw orbital. The actual basis functions for the calculation inside the MT are linear combinations of these and Y_{lm}.

Type: no content
XPath: `/wf`
Parent: `/spdb/sp/basis`
`/spdb/sp/basis/exception`
`/spdb/sp/lorb`

This element allows for specification of the following attributes:

`matchingOrder` (B.10.1) (required), `searchE` (B.10.2) (required), `trialEnergy` (B.10.3) (required)

B.10.1 Attribute: matchingOrder

Gives the order of the derivative that must be matched to the plain wave.

Type: integer
Use: required
XPath: `/wf/@matchingOrder`

B.10.2 Attribute: searchE

If `true` the energy of the radial wave function, E_0 is optimized to match the boundary condition

$$\psi(R_{MT}) = 0. \tag{B.1}$$

Type: boolean
Use: required
XPath: `/wf/@searchE`

B.10.3 Attribute: trialEnergy

Energy level of the radial wave function (initial condition for numerical radial Schrodinger equation)

Type: fortrandouble (B.11.1)
Use: required
XPath: `/wf/@trialEnergy`

B.11 Data Types

The Input definition uses derived data types. These are described here.

B.11.1 Type fortrandouble

The type `fortrandouble` allows to use the letters `"eEdDqQ"` for exponent operators. This alters in what precision the number is parsed.

Appendix C

spacegroup input reference

exciting developers team
(C. Ambrosch-Draxl, Zohreh Basirat, Thomas Dengg,
Rostam Golesorkhtabar, Christian Meisenbichler, Dmitrii Nabok,
Weine Olovsson, Pasquale Pavone, Stephan Sagmeister, Jürgen Spitaler)

About this Document

This document describes the input file format for the spacegroup tool.

C.1 Input Elements

C.2 Element: symmetries

The symmetries file format used by the **spacegroup** tool to generate structures and supercells as defined by `lattice` (C.4) from the knowledge of Wyckoff positions and the space group. The space group is specified by the attribute `HermannMauguinSymbol` (C.2.1). The root element is `symmetries` (C.2).

Contains:	`title` (C.3) (1 times)
	`lattice` (C.4) (1 times)
	`WyckoffPositions` (C.5) (optional)
XPath:	`/symmetries`

This element allows for specification of the following attributes:

`HermannMauguinSymbol` (C.2.1) (required)

C.2.1 Attribute: HermannMauguinSymbol

Herman Mauguin symbol giving the spacegroup

Type: string
Use: required
XPath: `/symmetries/@HermannMauguinSymbol`

C.3 Element: title

Type: string
XPath: `/symmetries/title`

C.4 Element: lattice

The lattice element defines lattice from a,b,c, and angles.

Type: no content
XPath: `/symmetries/lattice`

This element allows for specification of the following attributes:

`a` (C.4.1) (required), `ab` (C.4.2) (required), `ac` (C.4.3) (required), `b` (C.4.4) (required), `bc` (C.4.5) (required), `c` (C.4.6) (required), `epslat` (C.4.7), `ncell` (C.4.8), `primcell` (C.4.9), `scale` (C.4.10), `speciespath` (C.4.11), `stretch` (C.4.12)

C.4.1 Attribute: a

Type: fortrandouble (C.9.1)
Use: required
Unit: Bohr
XPath: `/symmetries/lattice/@a`

C.4.2 Attribute: ab

Angle between lattice vector a and b in degrees.

Type: fortrandouble (C.9.1)
Use: required
Unit: Degree
XPath: `/symmetries/lattice/@ab`

C.4.3 Attribute: ac

Angle between lattice vector a and c in degrees.

Type: fortrandouble (C.9.1)
Use: required
Unit: Degree
XPath: `/symmetries/lattice/@ac`

C.4.4 Attribute: b

Type: fortrandouble (C.9.1)
Use: required
Unit: Bohr
XPath: `/symmetries/lattice/@b`

C.4.5 Attribute: bc

Angle between lattice vector b and c in degrees.

Type: fortrandouble (C.9.1)
Use: required
Unit: Degree
XPath: `/symmetries/lattice/@bc`

C.4.6 Attribute: c

Type: fortrandouble (C.9.1)
Use: required
Unit: Bohr
XPath: `/symmetries/lattice/@c`

C.4.7 Attribute: epslat

Type: fortrandouble (C.9.1)
Default: "`1.0d-6`"
Use: optional
XPath: `/symmetries/lattice/@epslat`

C.4.8 Attribute: ncell

Number of repeated cells in each direction.

Type: integertriple (C.9.6)
Default: "`1 1 1`"
Use: optional
XPath: `/symmetries/lattice/@ncell`

C.4.9 Attribute: primcell

Type: boolean
Default: "`false`"
Use: optional
XPath: `/symmetries/lattice/@primcell`

C.4.10 Attribute: scale

Scales all the lattice vectors by the same factor. This is useful for varying the volume.

Type: fortrandouble (C.9.1)
Default: "1"
Use: optional
XPath: /symmetries/lattice/@scale

C.4.11 Attribute: speciespath

Type: string
Default: "http://xml.exciting-code.org/species/"
Use: optional
XPath: /symmetries/lattice/@speciespath

C.4.12 Attribute: stretch

Allows for an individual scaling of each lattice vector separately. "1 1 1" means no scaling.

Type: vect3d (C.9.4)
Default: "1.0d0 1.0d0 1.0d0 "
Use: optional
XPath: /symmetries/lattice/@stretch

C.5 Element: WyckoffPositions

Contains: wspecies (C.6) (zero or more)
XPath: /symmetries/WyckoffPositions

C.6 Element: wspecies

Contains: wpos (C.7) (zero or more)
XPath: /symmetries/WyckoffPositions/wspecies

This element allows for specification of the following attributes:

speciesfile (C.6.1)

C.6.1 Attribute: speciesfile

Type: string
Use: optional
XPath: /symmetries/WyckoffPositions/wspecies/@speciesfile

C.7 Element: wpos

Type: no content
XPath: `/symmetries/WyckoffPositions/wspecies/wpos`

This element allows for specification of the following attributes:

`coord` (C.7.1)

C.7.1 Attribute: coord

Type: vect3d (C.9.4)
Use: optional
XPath: `/symmetries/WyckoffPositions/wspecies/wpos/@coord`

C.8 Reused Elements

The following elements can occur more than once in the input file. There for they are listed separately.

C.9 Data Types

The Input definition uses derived data types. These are described here.

C.9.1 Type fortrandouble

The type `fortrandouble` allows to use the letters `"eEdDqQ"` for exponent operators. This alters in what precision the number is parsed.

C.9.2 Type vector

A vector is a space separated list of floating point numbers.
Example: `"1.3 2.3e4 3 90"`

C.9.3 Type integerlist

List of space separated integers.

C.9.4 Type vect3d

Three dimensional vector as three space separated floating point numbers.

C.9.5 Type vect2d

Three dimensional vector as three space separated floating point numbers.

C.9.6 Type integertriple

Space separated list of three integers.
Example: `"1 2 3"`

C.9.7 Type integerquadrupel

Space separated list of three integers.
Example: `"1 2 3 4"`

C.9.8 Type integerpair

Space separated list of two integers
Example: `"1 2"`