BAND

a Fortran program for bandstructure calculations

Contents

This chapter gives an extensive discussion of the computer program with many details concerning the software. We start with a list of acronyms and abbreviations that will be used and a global flow chart.

BZ	Brillouin Zone	NAO	numerical atomic orbital
DF	density functional	PW	plane wave
DOS	density of states	SCF	self consistent field
irrep	irreducible representation	SSA	stable state approach
LCAO	Linear combination of atomic orbitals	STO	Slater type orbital
		XC	exchange and correlation
LD	local density		

Flow chart

BAND

INIT	initiation of variables
GETINP	input analysis
GEMTRY	geometry master routine
POINTS	symmetry unique points and weights for numerical integration
SYMCRY	space group operators
ATMSET	division of atoms in sets of symmetry equivalent ones
SYMPRJ	point group operators in reciprocal space, derived from the space group operators
SYMADD	inclusion of inversion symmetry into the k-space point group
KPNT	integration points and related data for integrals over the BZ
PREPAR	master routine for the preparation of crystal functions
RADIAL	master routine for radial parts of one-center function
RADFNC	master routine for generation of radial function tables
DIRAC	numerical solution of DF equations for spherically symmetric free atoms; output tables (NAOs, atomic densities and coulomb potentials)
SLTORB	STOs, tables
FITRAD	Slater type fit functions (STF's) and potential functions
RADMAX	maximum radial extension of any tabulated function (valence, fit, atomic density, atomic coulomb potential)
CELLS	coordinates of lattice points that are relevant for the calculation of bloch sums
CELMAX	for each radial function table: the number of cells needed in the bloch sum, derived from its radial extension
FITSYM	number of totally symmetric fit functions
NUMGRD	master routine for computation of function values in the crystal integration points: interpolation from the radial tables, bloch summation by a loop over the relevant cells
RPNTID	generation of all symmetry equivalent integration points, from the unique ones

RPNTRE	organization of the file with points: blocks of points with a pre-determined maximum length (the vector length)
VMULTI	multipole potentials: potential values in the integration points due to a Bravais lattice of multipoles; each {*lm*}-multipole; Bravais lattices centered on each atom; symmetrized functions by combining symmetry equivalent atoms
ATOMIC	master routine for data related to the free-atom charges and coulomb potentials
ELSTAT	electrostatic interaction between unrelaxed free atoms (energy term in the cohesive energy)
ATMFNC	superposition of the atomic densities and coulomb potentials in the integration points (reference charge density and potential for the SCF procedure)
BASORT	master routine for the crystal core and valence functions
PLANEW	characteristics of the planes waves in the valence basis (lattice points in reciprocal space)
BASPNT	bloch summations of one-center basis functions (core and valence); PWs are added to the valence function set
BASCOR	explicit orthogonalization of the valence basis on all core states
BASOVL	overlap matrix of the basis
BASTRA	transformation to an orthonormal basis
HAMFIX	fixed part of the hamiltonian matrix (in the orthonormal basis): kinetic energy and coulomb potential due to the free atoms; matrices (one for each k-point) are stored on file
FITORT	master routine for the fit functions
FITPNT	interpolation and bloch summation ($\boldsymbol{k}$=0); combination into functions that are totally symmetric with respect to all space group operators
FITOVL	overlap matrix of the (symmetrized) fit set
FITTRA	transformation to an orthonormal basis
(*end of NUMGRD*)	
(*end of PREPAR*)	
REORGF	reorganization of all files containing function values in the integration points: fewer blocks of points / more points per block
SCF	master routine for self consistency iterations
SCFTST	test termination conditions of the SCF procedure
RHOPMT,	charge density in the integration points from the density matrix (RHOPMT) or
RHOPSI	from the eigenstates (RHOPSI)
RHOFIT	fit coefficients: expansion of the deformation density in fit functions, to solve (approximately) the Poisson equation
RHOPOT	the potential (coulomb plus XC), computed from the density and the fit coefficients
EIGSYS	evaluation and diagonalization of the hamiltonian matrices in the respective k-points
FERMI	master routine for k-space analysis

FERMIE	fermi energy from the energy bands; occupation numbers for all one-electron states
PMATRX	the density matrix in the representation of the crystal valence basis

(*end of SCF*)

PRPRTS	master routine for the analysis of results; computation of properties
ENERGY	cohesive and total energy
CHARGE	charge distribution over the atoms by numerical integration
DOS	master routine for the (total and partial) density of states
DOSTRA	back transformation of the self-consistent wave functions to the original, non-orthogonal bloch-valence basis: for the partial densities of states and for the Mulliken population analysis
DOSCAL	partial and total density of states for a sequence of energy values
POPANA	Mulliken population analysis
FORMFA	master routine for X-ray structure factors (form factors)
PLANEW	characteristics of the relevant plane waves
CELRED	reduction to the symmetry unique set of plane waves
FORMF1	computation of the X-ray factors: fourier transformation of the charge density by numerical integration

(*end of PRPRTS*)

(*end of BAND*)

1 Introduction

BAND is a density functional (DF) program for electronic structure calculations on periodic systems. The programming language is FORTRAN77. Where in the text explicit reference is made to the software, subroutine names are in small capitals (SCF, INIT), variables are underlined (pot, overlp) and key-words used in input are outlined (`mix`, `lattice`). Details of the implementation will be discussed in a number of 'software sections'; these are referred to as SS^input, SS^basis, and so on.

Energies and lengths are in atomic units (hartree, bohr), unless explicitly stated differently.

formalism

In the Kohn-Sham theory of the DF method [Hohenberg and Kohn 1964, Kohn and Sham 1965] the two-particle coulomb interaction $1/r_{12}$ between electrons is replaced by the sum of two one-particle operators. The first is the coulomb potential due to the average charge distribution. The second is the exchange-correlation (XC) potential, representing exchange and correlation effects in an average way. The XC potential is a functional of the charge distribution.

With the usual approximation of motionless point nuclei the hamiltonian equation reads

$$H\psi_n(\boldsymbol{k};\boldsymbol{r}) \equiv \{ T + V_C(\boldsymbol{r}) + V_{XC}(\boldsymbol{r}) \}\ \psi_n(\boldsymbol{k};\boldsymbol{r}) = e_n(\boldsymbol{k})\ \psi_n(\boldsymbol{k};\boldsymbol{r}) \tag{1.1}$$

T is the kinetic energy operator, $-\Delta/2$ in atomic units; $V_C(\boldsymbol{r})$ is the total coulomb potential, due to the nuclear charges and the electron cloud; $V_{XC}(\boldsymbol{r})$ is the XC potential. Relativistic effects have not (yet) been included in BAND.
$\psi_n(\boldsymbol{k};\boldsymbol{r})$ is a one-electron state with wave vector $\boldsymbol{k}$; $\boldsymbol{k}$ serves also as a symmetry label, denoting the irreducible representations (irreps) of the translation group corresponding to the Bravais lattice. It is a (pseudo) continuous variable that may assume all values in the (first) Brillouin Zone (BZ).
The solutions $\{ e_n(\boldsymbol{k}), \psi_n(\boldsymbol{k};\boldsymbol{r}) \}$ vary continuously with $\boldsymbol{k}$ and form thus 'bands'; the subscript n, which enumerates the distinct solutions in $\boldsymbol{k}$, is called the band index.

The translation properties of irrep $\boldsymbol{k}$ are expressed by (Bloch's theorem)

$$\psi_n(\boldsymbol{k};\boldsymbol{r}+\boldsymbol{R}) = e^{i\boldsymbol{k}\cdot\boldsymbol{R}}\ \psi_n(\boldsymbol{k};\boldsymbol{r}) \tag{1.2}$$

where $\boldsymbol{R}$ is any point of the Bravais lattice. The states $\psi_n(\boldsymbol{k};\boldsymbol{r})$ are computed as linear combinations of basis functions ϕ. These are chosen of course to belong to the same irrep and are labeled accordingly: $\phi_i(\boldsymbol{k};\boldsymbol{r})$, i=1,2,...

The electronic charge density $\rho(\boldsymbol{r})$ is obtained by a summation over all occupied states, i.e. all states with energy $e_n(\boldsymbol{k})$ below the fermi energy e_F. The summation includes the integration in k-space

$$\rho(\boldsymbol{r}) = \sum_{n} \int_{BZ} d\boldsymbol{k}\ \hat{o}\,\psi_n(\boldsymbol{k};\boldsymbol{r})\,\hat{o}^2\ \ \theta(e_F - e_n(\boldsymbol{k}))\ \ (1.3)$$

$\theta(x)$ is the Heaviside step function: $\theta=1$ ($x>0$) or 0 ($x<0$).
The fermi energy e_F is determined by the total amount of electronic charge Q per unit cell

$$\int_{unit\ cell} \rho(\boldsymbol{r})\, d\boldsymbol{r} = Q \qquad (1.4)$$

The potentials $V_C(\boldsymbol{r})$ and $V_{XC}(\boldsymbol{r})$ in (1.1) are computed from the density $\rho(\boldsymbol{r})$ and hence they depend on the solutions $\psi_n(\boldsymbol{k};\boldsymbol{r})$. The *self-consistent* solutions are found by an iterative self-consistent field (SCF) procedure.

applicability and results

Any type of periodic system can be handled. The systems will be called *n*-dimensional crystals, by which we understand polyatomic systems with periodicity in *n* directions; *n* may be 3 ('normal' bulk crystals), 2 (slabs) or 1 (polymers).
  It seems natural to include also the case $n=0$ (molecules). BAND is based on the same theoretical model as the Amsterdam DF molecule program and the two are very similar as regards the general set-up. In fact they share a substantial portion of their software. A future melting together may well be undertaken and should not present fundamental problems.
The two spins can be treated independently: both spin-restricted and spin-unrestricted calculations are possible.
For $V_{XC}(\boldsymbol{r})$ several forms advocated in the literature are available in the program: the classical X_α , the Gunnarsson-Lundqvist (GL) and the Vosko-Wilk-Nusair (VWN) formulas. Any of these can be elected in a calculation. Other varieties are easily implemented if desired.
Apart from the self consistent solution of (1.1), the program computes the total and cohesive energies, X-ray factors, total and partial densities of states and it performs a Mulliken population analysis.
There are no restrictions on computational parameters such as a maximum number of atoms, basis functions and so on. Of course computer resources may limit the applicability. To have an indication: a slab containing 18 transition metal atoms and two first row atoms per 2D unit cell (periodic chemisorption of CO on a copper layer) is a *big* system, as regards both CP-time and disc-usage. If the basis set and other parameters are chosen to achieve high accuracy (0.001a.u. in the cohesive energy), the program will take ten or more hours on a Cyber205 and handle ~10^9 words of data. Bulk silicon is computed with fair precision (better than 0.01a.u. in the cohesive energy) in 20 minutes and stores 7×10^7 words on disc during the run.

  We will discuss in other sections how the demands on storage facilities and computer time may be cut down by future developments of the program, primarily by making better use of symmetry properties than is done currently (SS^symmetry). In the mentioned example of Cu-CO adsorption for instance this

would reduce costs by roughly an order of magnitude. The gain in the silicon calculation would be even more, due to the high symmetry.

An improvement in efficiency may also result from a refining of the SCF procedure (the Stable State Approach, see SS^basis).

basic characteristics of the method

starting up, one-center function tables

BAND contains a fully numerical Herman-Skillman [Herman and Skillman 1963] type subprogram DIRAC. DIRAC solves the DF equations for the spherically symmetric free atoms from which the crystal is built up. The superposition of the atomic densities and the corresponding potential is used to start up the self-consistency iterations for the crystal. Cohesive energies are computed with respect to these atoms. The atomic one-electron states are optionally used in the basis set for the crystal.

All data from DIRAC such as the atomic density, the coulomb potential and the orbitals are obtained in the form of tables $f(r_i)$, i=1,2,.. , giving the values of $f(r)$ for a sequence of radial values. Other one-center functions (see below) are represented in the same way, as radial tables, even if they could alternatively be treated analytically.

The radial grid points r_i of a table constitute a logarithmic mesh: r_{i+1} /r_i =constant. The grid-characteristics, i.e. the number of points, the quotient-constant and the smallest value r_1 are the same for all one-center functions associated with a certain type of atoms. The radial meshes may be different for different types of atoms.

basis

The program employs two types of basis functions $\phi(\boldsymbol{k};\boldsymbol{r})$:
a) Bloch functions [Bloch 1928]. These are computed as linear combinations of localized functions $\chi(\boldsymbol{r})$

$$\phi(\boldsymbol{k};\boldsymbol{r}) = \sum_{\boldsymbol{R}} e^{i\boldsymbol{k}\cdot\boldsymbol{R}}\ \chi(\boldsymbol{r}-\boldsymbol{R}) \tag{1.5}$$

The phase factors $e^{i\boldsymbol{k}\cdot\boldsymbol{R}}$ in the summation over the direct lattice points $\boldsymbol{R}$ assure that $\phi(\boldsymbol{k};\boldsymbol{r})$ has the correct translational symmetry (1.2). The localized functions $\chi(\boldsymbol{r})$ are one-center functions, centered on some atom α. They have the form

$$\chi(\boldsymbol{r}) = Z_{lm}(\Omega_\alpha)\ P(r_\alpha) \tag{1.6}$$

The subscript α signifies that the coordinates are relative to the position of atom α. $Z_{lm}(\Omega)$ is a (real valued) spherical harmonic (23.30) and $P(r)$ is a radial function.
Two types of radial functions are used in the program. They are the radial parts of

a1) Slater type orbitals (STOs): $P(r)=r^{l+n}\,e^{-\alpha r}$.
a2) Numerical atomic orbitals (NAOs) from DIRAC.

b) Plane waves (PWs): $\phi(\boldsymbol{k};\boldsymbol{r})=e^{i(\boldsymbol{k}+\boldsymbol{K})\cdot\boldsymbol{r}}$. $\boldsymbol{K}$ is a point of the reciprocal Bravais lattice. Plane waves are used only for 3-dimensional crystals. In other systems the asymptotic behaviour away from the crystal is unsuitable to describe bound electron states.

Note: although the use of a pure PW valence basis is possible (orthogonalized on the core states, as the case might be, see below), such an application should be undertaken cautiously. All integrals are evaluated numerically and no use is made of the analytical integrability of PWs. The integration scheme has not been devised for rapidly oscillating functions and hence it will be less accurate for them, or, if the number of integration points is accordingly increased, we are confronted with higher costs. A high-precision PW calculation with large numbers of PWs may therefore be less efficient.
The cause of this is purely historical: BAND has been developed as an LCAO program; the PWs have been included later and are (currently) treated in the same way as other functions.

frozen core

Some of the NAOs may be specified to be core states. These are not iteratively computed in the crystal SCF procedure. Each (valence) basis function is orthogonalized on all core states by explicitly projecting out the core functions.

integrals

Integrals in real space over the crystal unit cell, such as overlaps and hamiltonian matrix elements, are evaluated numerically. The applied integration scheme is based on product Gauss formulas and a partitioning of space in specific regions: atomic polyhedra ('cells'), inside each polyhedron an atomic 'core; sphere, and the 'outer' region far away from the nuclei (not for 3D crystals). Any desired accuracy can be achieved, but of course with a concomitant number of integration points [chapter III].

k-space

Integrations over the BZ, for instance to compute the density (1.3), are performed numerically, using the analytic-quadratic method [Wiesenekker *et al.* 1988, Wiesenekker and Baerends 1990]. Any quantity F is approximated as

$$F \equiv \sum_n \int_{BZ} d\boldsymbol{k}\; F_n(\boldsymbol{k})\;\; \theta(e_F - e_n(\boldsymbol{k})\,) \approx \sum_n \sum_{\boldsymbol{k}} o_n(\boldsymbol{k})\;\; F_n(\boldsymbol{k}) \qquad (1.7)$$

$F_n(\boldsymbol{k})$ is the contribution to F from the eigenstate $\psi_n(\boldsymbol{k};\boldsymbol{r})$; in case of the density for instance $F=\rho(\boldsymbol{r})$, $F_n(\boldsymbol{k}) = |\psi_n(\boldsymbol{k};\boldsymbol{r})|^2$. The integration weights $o_n(\boldsymbol{k})$ may obviously be associated with occupation numbers for the states $\psi_n(\boldsymbol{k};\boldsymbol{r})$. Henceforth we will refer to them in this way, although the analytic-quadratic method occasionally results in 'non-physical' negative occupation numbers.

The analytic-quadratic method implicitly makes a piecewise quadratic expansion in the variable $\boldsymbol{k}$ both of $F_n(\boldsymbol{k})$ and of the energy function $e_n(\boldsymbol{k})$. The occupation numbers are then determined such, that these second order polynomials are integrated exactly. The method is accurate and converges quickly for all types of systems: isolators, metals and semiconductors [Wiesenekker *et al.* 1988, Wiesenekker and Baerends 1990].

Poisson's equation

The coulomb potential $V_C(\boldsymbol{r})$ due to the nuclear charges Z_α and the electronic density $\rho(\boldsymbol{r})$ is defined in the program as

$$V_C(\boldsymbol{r}) = \int \left\{ \rho(\boldsymbol{r}') + \sum_\alpha Z_\alpha \, \delta(\boldsymbol{r}'-\boldsymbol{r}_\alpha) \right\} \hat{o}\, \boldsymbol{r}-\boldsymbol{r}' \hat{o}^{-1} \, d\boldsymbol{r}' \tag{1.8}$$

The tables from DIRAC give the coulomb potentials $V_\alpha(\boldsymbol{r})$ due to the spherically symmetric atomic densities $\rho_\alpha(\boldsymbol{r})$ plus nuclei Z_α. Defining the deformation density $\rho_{def}(\boldsymbol{r})$ as the difference between the crystal charge distribution and the superposition of atomic densities gives

$$V_C(\boldsymbol{r}) = \sum_\alpha V_\alpha(\boldsymbol{r}) + \int \rho_{def}(\boldsymbol{r}') \, \hat{o}\, \boldsymbol{r}-\boldsymbol{r}' \hat{o}^{-1} \, d\boldsymbol{r}' \tag{1.9}$$

The form in which $\rho_{def}(\boldsymbol{r})$ is obtained precludes an analytical evaluation of the second term and the singularity of the denumerator makes application of the normal numerical integration scheme unsuitable. The problem is solved by a fitting procedure as introduced by Baerends *et al.* [1973]. A set of fit functions $f_i(\boldsymbol{r})$ is chosen such that *a)* the density $\rho_{def}(\boldsymbol{r})$ can accurately be expanded in them and *b)* the corresponding coulomb potentials $f_i^C(\boldsymbol{r})$ are easily evaluated. Then, with

$$f_i^C(\boldsymbol{r}) \equiv \int f_i(\boldsymbol{r}') \, \hat{o}\, \boldsymbol{r}-\boldsymbol{r}' \hat{o}^{-1} \, d\boldsymbol{r}' \tag{1.10}$$

$$\rho_{def}(\boldsymbol{r}) \approx \sum c_i \, f_i(\boldsymbol{r}) \tag{1.11}$$

the coulomb potential is approximated by

$$V_C(\boldsymbol{r}) \approx \sum_\alpha V_\alpha(\boldsymbol{r}) + \sum c_i \, f_i^C(\boldsymbol{r}) \tag{1.12}$$

The fit functions are derived from one-center 'atomic' functions $\xi(\boldsymbol{r})=Z_{lm}(\Omega_\alpha) \, P(r_\alpha)$. This form allows an easy evaluation of the corresponding potential functions $\xi^C(\boldsymbol{r})$ (SS^coulomb potential) .

The density is a symmetric function, invariant under all operators of the space group. The *generating* functions $\xi(\boldsymbol{r})$ are therefore combined into symmetric functions $f(\boldsymbol{r})$; these are the crystal fit functions. The same combination coefficients yield the associated fit potentials $f^C(\boldsymbol{r})$ from the potential functions $\xi^C(\boldsymbol{r})$.

2 Implementation

We deal with the implementation on two levels. First we survey the flow of normal execution. We do this from a conceptual point of view, neglecting details and simplifying matters for sake of clarity. This gives insight in the organization of BAND and tells us globally what happens when and where.
In the second stage we look closer by giving attention to special aspects in a series of Software Sections (SS^basis, SS^input, SS^files, etc). There we examine input specifications, usage of files, treatment of symmetry, details of some algorithms and so on. Apart from being useful in itself as documentation, this may be of aid in case of error-tracing and when modifications or extensions of BAND are contemplated. A few suggestions for improvements of the existing code have been added.

survey

BAND consists conceptually of three parts. The heart is part two, the iterative SCF procedure to find the self-consistent solution of the hamiltonian equations. Part one is the preparation for this: input reading, geometry analysis, construction of the valence basis, etc.; it starts with subroutine INIT, where several variables are initiated. In part three the results are analyzed and various properties are computed.
The global flow-chart at the beginning of this chapter lists the main routines (names in capitals) with a summary of their purposes. The indentations reflect the hierarchical structure; for instance: INIT, GEMTRY and PREPAR are called by the main program (BAND); GETINP in turn is called from INIT.

crystal functions

The primary goal of PREPAR is the preparation of various crystal functions (valence, fit, potential, density). These functions are treated as purely *numerical* functions, whatever their origin; they are represented by the values in the crystal integration points. These values are in most cases (the only exception being the PWs in the valence basis) defined and computed as bloch sums: the contributions from the various cells are added in a loop over the crystal lattice points.

As said before, all primitive one-center functions, whose contributions are to be added, are represented by tables containing the radial values for a number of distances from their respective 'origins'. They are stored on various files, together with the angular quantum numbers which define their angular dependency.

The preparation of crystal functions thus consists roughly of two steps: *a)* generation of the radial tables (DIRAC, SLTORB, FITRAD), *b)* interpolation from the tables to compute them in the crystal integration points and bloch summation (ATMFNC, BASPNT, FITPNT).

bloch sums

Depending on the size of the unit cell and the dimensionality of the crystal, the loops over the cells (bloch summations) may involve several thousands of terms and take a substantial time. To reduce the costs, BAND attempts to limit these loops. For this purpose each involved radial function table is analyzed to find out at which distance the function becomes negligible; the corresponding number of cells, different for each function, is determined (CELMAX) and used in the bloch summation procedures.
As a preliminary the maximum extension of *any* radial function is determined (RADMAX); this is used to set up an appropriate list of lattice points (cell coordinates: CELLS).

orthonormal function sets

The valence function set and the fit function set are both transformed to an orthonormal basis via diagonalization of the overlap matrices (BASOVL, BASTRA, respectively FITOVL, FITTRA). This is convenient in the subsequent employment: some computational procedures are simplified and memory usage in the SCF procedure is alleviated substantially (the overlap matrices may be rather large).
Linear dependency is controlled by checking the eigenvalues of the overlap matrix.

symmetry and integration points

Symmetry plays an important role in reducing computational efforts (SS^symmetry). Furthermore the crystal density and potential are explicitly symmetrized to preserve the symmetry of the hamiltonian (symmetry breaking is prevented).
Some symmetry-analyzing routines (SYMCRY, ATMSET) are subordinate to the numerical integration routine (POINTS) for the following reason:
POINTS is the master routine of an extensive and sophisticated numerical integration package for polyatomic systems [chapter III]; input are the coordinates of the atoms and the lattice structure, plus specifications concerning the required precision. In order to generate an integration scheme that reflects the symmetry of the polyatomic system, the symmetry operators are determined and the atoms are organized in groups of symmetry equivalent ones. This symmetry information, in fact only a by-product of POINTS, is subsequently used in various parts of BAND.

In principle it is trivial to derive the point group operators in *k*-space from the space group operators: the point group parts of the affine symmetry transformations are symmetry operators in *k*-space. However, in an n-dimensional crystal *k*-space has only *n* dimensions, which may be lower than 3. Moreover BAND optionally neglects dispersion in certain directions in *k*-space. In general the *k*-space operators are therefore obtained by projection into the space with the appropriate dimensionality (SYMPRJ).
Inversion is a symmetry operator in *k*-space, regardless of the space group. So, if it is not yet present after the 'projection' procedure, it must be added (SYMADD).

workspace and vector length

FITSYM determines only the *number* of symmetric fit functions; this is used for appropriate dimensioning of some arrays (BAND simulates dynamical allocation: SS^workspace).

The integration points are organized in *blocks* of points. All related data on files, such as the values of various crystal functions in the points, are structured accordingly. The processing of these data thus takes place in a loop over the blocks. The maximum number of points per block, npx, i.e. the block length, is computed to organize work space in an optimal way (SS^workspace).

In comparison with the preparation part the SCF part has effectively more workspace available (in particular thanks to the absence of various overlap matrices) to load in core e.g. the valence functions in a block of integration points. Increasing the lengths of the blocks (REORGF) leads then to longer vector lengths in vector operations, for instance in the evaluation of hamiltonian matrix elements by numerical integration. This results in a substantial improvement of efficiency (depending on the machine).

multipole lattice sums

BAND solves the Poisson equation by a fitting procedure for the deformation density. The corresponding fit potentials are lattice sums of functions that are asymptotically multipole potentials. Naturally these lattice summations are split in a rapidly converging summation of exponentially decaying functions and the lattice sum of point-multipole potentials. The first term is treated in the fit section (FITPNT) by a straightforward loop over lattice points. The second term gives the familiar solid state problem of slowly (or even conditionally) convergent lattice sums. VMULTI evaluates the pure multipole lattice sums (by an unconventional algorithm: SS^coulomb potential); the data are stored on file itvmul, read again in FITPNT and combined with the other term to the fit potentials.

Finally all fit data (function values and potential values) are stored on file to be used in the SCF procedure.

SCF

When all crystal functions have been prepared, the SCF procedure is started. The initial density and the corresponding potential are given as the 'sum-of-overlapping-free-atoms'. Iteratively the following steps are then executed:

a. numerical integration of the potential matrix elements; this is added to the fixed part of the hamiltonian (kinetic energy). The hamiltonian matrices (one for each *k*-point) are diagonalized (EIGSYS).
b. analysis of the energy bands to determine the fermi energy and the occupation numbers for the one-electron states (FERMI).
c. computation of the density matrix in the representation of the basis functions (P-matrix) (PMATRX).
d. construction of the charge density in the integration points from the P-matrix (RHOPMT).
e. expansion of the deformation density in the fit functions, for the solution of Poisson's equation (RHOFIT).

f. calculation of the new potential (RHOPOT) from the fit functions (coulomb potential) and the density (XC potential).

SCFTST tests various conditions for the termination of the SCF procedure (convergence, insufficient time, maximum number of iterations).

post-SCF

After the SCF procedure BAND computes various properties. This is organized in the master section PRPRTS. PRPRTS calls subsequently

a. ENERGY to compute the bonding energy, partitioned in various terms, and the crystal total energy.
b. CHARGE to determine by numerical integration the charge distribution over the atoms.
c. DOS, for an analysis of the density of states (total and partial) and a Mulliken population analysis.
d. FORMFA to compute the X-ray structure factors (form factors).

Software Sections

In the remainder of this chapter we focus attention on the implementation. Theoretical remarks and derivations are included, but some acquaintance with the theory of solid state, quantum chemistry and numerical mathematics is assumed. The treated subroutines and subjects are listed in the Software Reference Lists A and B, after the last Software Section.

A program like BAND is apt to be used on several types of computers. Furthermore its size and complexity make it probable that errors are hidden in it, even after extensive testing. We have adhered therefore to standard FORTRAN77. The use of 'smart and dirty tricks' and of machine-dependent FORTRAN dialects may let the program run faster, but it is heavily paid for in human time, when implementation on another machine is on stage, or when bugs show up.

It is hardly avoidable however that the program be machine-dependent in some respects. The imperfection of compilers, as regards the generation of the most efficient code, may lead in some cases to unreasonably bad performance. Although this is in principle a matter of awaiting better compilers and 'not our business', it is more practical to take measures. In particular we have done so with vectorizable loops. Consider

```
    do 20 j=1,m-1                                              (2.1)
       do 10 i=1,n
10     a(i,j+1)=a(i,j)+b(i)
20  continue
```

The inner loop should be executed in vector mode, but the compiler may not consider it vectorizable (e.g. suspicion of recurrence) and generate scalar code. *If* it is vectorized, memory banking conflicts may occur when the inner loop length *n* is very large.
To solve these problems we utilize a number of special subroutines for vector operations. The code above is replaced by

```
   do 20 j=1,m–1                                                    (2.2)
      call vauvw(n,a(1,j),b,a(1,j+1))
20 continue
```

Subroutine VAUVW (*v*ector *a*ddition: $u+v=w$) adds two vectors and stores the result into a third. Similar routines multiply, divide, perform triadic operations etcetera. The disadvantage is the overhead due to the call; this is relatively small if n is large enough. The advantage is that the problems are transferred from a very large number of similar pieces of code to a small number of simple routines.
The vectorization problem is solved automatically in this way: the simplicity of the routines implicates already that any compiler will generate vector code for them. We might even comfortably use explicit vector statements if the local FORTRAN dialect allows so, because the resettings to standard FORTRAN are few and straightforward.
Memory banking conflicts are solved by splitting up vectors that are too long. A special constant lveccp, the maximum vector length for CP-operations, controls this; it is of course machine-dependent and should be adapted to the situation. lveccp can be set via input, with the key `cp vectors`.
All explicit machine-dependency in BAND is controlled by a few similar constants; these are stored in common block MACHIN.

BAND frequently tests the state of affairs during execution. Many of these tests are superfluous if the input is sensible and the code is correct. The tests are there, because neither of these conditions is guaranteed. It would not be the first time that a bug in the program, showing up in a new type of calculation, is noticed and traced in this way. Warnings are issued if intermediate results are suspicious and in some cases BAND stops, to avoid the waste of human time and computer resources.

We have attempted in several ways to make the program flexible and easy to use. In particular input is largely optional, with reasonable default settings for all omitted specifications; input is governed by *keys* (strings, usually single words). This is discussed in SS^input. In many other Software Sections the related input options and the corresponding keys are mentioned. Keys will be typed outlined (`integration`).

3 Basis

The construction of the crystal basis functions is organized in BASORT. The generating one-center functions χ, produced in DIRAC and/or SLTORB for the NAOs and STOs respectively are on file itpsi and the kinetic energy functions $-(\Delta/2)\chi$ on itkin.
The plane waves $e^{i(\boldsymbol{k}+\boldsymbol{K})\cdot\boldsymbol{r}}$ in the basis are characterized by the lattice points $\boldsymbol{K}$ of the reciprocal lattice. BASORT calls PLANEW to generate these vectors from the input-specified number of stars nwavst (SS^input). The resulting number of plane waves is nvalwv. The total number of valence basis functions for each k-point is nbas.

general

BASPNT interpolates the one-center functions, computes their bloch sums and adds the plane waves to the basis set. This is done for a number of k-points simultaneously (SS^workspace); after BASPNT the k-points are processed one at a time.

BASOVL calculates and diagonalizes the overlap matrix of the basis $\{\phi\}$. From the eigensystem the transformation to an orthonormal basis is constructed. Let S be the overlap matrix and $\{E,\Lambda\}$ the eigen system: $S=E\Lambda E^{+}$. Define the transformation matrix U as

$$U_{ij} = E_{ji} \quad \lambda_i^{-1/2} \tag{3.1}$$

The transformed functions ϕ'

$$\phi_i' = \sum_j U_{ij}\ \phi_j \tag{3.2}$$

are orthonormal:

$$\phi_i' \hat{o} \phi_j' = \sum_k U_{ik}\ \phi_k \hat{o} \sum_l U_{jl}\ \phi_l = \sum_{kl} E^{*}_{ki}\, E_{lj}\, (\lambda_i\lambda_j)^{-1/2}\ S_{kl} =$$
$$= \sum_m \Big(\sum_k E^{*}_{ki}E_{km}\Big)\Big(\sum_l E_{lj}E^{*}_{lm}\Big)\ \lambda_m(\lambda_i\lambda_j)^{-1/2} = \delta_{im}\ \delta_{jm}\ \lambda_m(\lambda_i\lambda_j)^{-1/2} = \delta_{ij} \tag{3.3}$$

We used here that the eigenvectors of S are orthonormal. BASTRA performs the transformation.
The final orthonormal valence basis and the transformed kinetic energy functions $(-\Delta/2)\phi'$ are written to (scratch) files itbas0 and itkbas. These are used (and deleted again) in HAMFIX to construct the matrices T and H_0 . T is the kinetic energy matrix in the orthonormal basis; it is used to calculate the valence kinetic energy of the crystal after completion of the SCF procedure (SS^energy).
$H_0 \equiv T+\sum V_\alpha$ is the fixed part of the hamiltonian, consisting of the sum-of-atoms coulomb potential and the kinetic energy. In the SCF part this is combined with the iteratively computed XC potential and the

coulomb potential of the deformation density. The H_0 -matrix is written to file ith0, the kinetic energy matrix T to itdatm.
HAMFIX copies also the valence function values from itbas0 to itbas, where the data for all *k*-points are accumulated for later use in the SCF part.

linear dependency

The eigenvalues of the overlap matrix are used to check linear dependency of the valence basis. Strictly we have dependency only if S is singular, that is, if at least one of the eigenvalues is zero. Non-zero but very small eigenvalues are already hazardous however. The corresponding eigenfunctions are linear combinations of the original functions with large (and opposite) coefficients. Small numerical errors, for instance in the kinetic energy of the primitive functions, are blown up and indeed we find in practice that the results become unreliable.
Therefore BAND stops (in BASOVL) if the smallest eigenvalue is less than some criterium scrval. scrval is input with the key `scrval`, `basis overlap` or `basis depend`; (first and second) defaults are 10^{-4} and 10^{-5} .

The coefficients in all eigenvectors with small eigenvalues indicate which of the basis functions cause the dependency problems. We have summarized this information in 'dependency coefficients', defined as

$$w_i = \sum_j \frac{\text{scrval}}{\lambda_j} \hat{o} E_{ij} \hat{o}^2 \qquad (3.4)$$

The summation runs over all eigenvectors; λ_j is the eigenvalue and E_{ij} is the coefficient of the *i*-th function in the *j*-th eigenvector. The dependency coefficients are printed if they exceed a threshold. This threshold depends on iprntp, the general print option for the preparation part.

frozen core

Some of the one-electron states computed in DIRAC may be specified to be core states (SS^input). Their function tables are written to the files itpsic and the kinetic energy functions to itkinc. Like the valence set they are interpolated and combined in a bloch sum by BASPNT.

Core states are assumed to remain fixed when the atom is embedded in the crystal. A necessary, though not sufficient condition is of course that core states of different atoms have no overlap; in particular they must not extend into neighbouring (Wigner-Seitz) cells. Their bloch sum consists therefore effectively of only one term for any evaluation point $\boldsymbol{r}$. So they display no dispersion and the bloch 'sum' has to be constructed only for the Γ-point $\boldsymbol{k}$=0. This is the ideal situation. In practice however one may, e.g. for computational reasons, wish to define larger core spaces, containing states that have non-negligible dispersion. To take care of that the core is constructed for every *k*-point separately; the computational overhead is relatively small because the bloch-sum for core states is obtained always in a very short loop over lattice points. Core

dispersion may be neglected by specifying the input key `no core dispersion`. In that case the core bloch sums are computed only once (for the Γ-point) and used for all *k*-points.

The overlap matrix of the core is computed (BASOVL) and the transformation performed to an orthonormal set (BASTRA), in the same way as it is done for the valence basis. Ideally this is trivial: the overlap matrix should be the unit matrix. It is useful however to check this: the numerical integration may be inaccurate and, more importantly, some of the core functions may be too diffuse, thereby violating the implicit assumptions about core states. The eigenvalues of the core-overlap matrix are compared with a criterium scrcor (input by key `scrcor`, `core overlap` or `core depend`; defaults are 0.98 and 0.90). BAND terminates when the test reveals that the frozen core approximation is unjustified.

The core functions are kept frozen and they are not processed in the SCF procedure. The valence basis must then be orthogonal on the core. This is achieved by explicitly projecting out the core components (before the valence set is orthonormalized).

$$\phi_i^{valence} \leftarrow \phi_i^{valence} - \sum_j \phi_j^{core} \; \phi_j^{core} \,\hat{o}\, \phi_i^{valence} = \phi_i^{valence} - \sum_j S_{ji}\, \phi_j^{core} \qquad (3.5)$$

BASCOR computes the core-valence overlap matrix S and orthogonalizes the valence on the core. The core function values are stored on files itcor and itkcor. The functions are retrieved again from these files when the valence basis is constructed; after the orthogonalization of the valence space on the core the core-files are deleted.

valence-core dependency

The frozen core approximation introduces an additional linear dependency problem: the valence space and the core space may have a vector (almost) in common. Analogous to the situation in the valence set itself this must be checked as it may lead to the blowing up of numerical errors. (In fact we realized this problem in a sequence of calculations on diamond; varying a double-ζ STO-basis we found an unexpected lowering of the total energy by several eV for a particular valence set; indeed the valence basis contained effectively almost the carbon 1s core orbital.)
BAND deals with this by computing for each *k*-point the maximum overlap between normalized vectors in the core and valence space respectively. This is analyzed in BASOVL, where the transformation to an orthonormal valence basis is to be computed.
The core basis has already been orthonormalized at that moment. A general core state, with normalization condition, is then

$$f^{core} = \sum c_i \, \phi_i^{core} \qquad (3.6a)$$

$$\boldsymbol{c}^{\dagger} \, \boldsymbol{c} = 1 \qquad (3.6b)$$

The valence basis is not (yet) orthonormal. Denote the valence overlap matrix by V; a general valence function with normalization is

$$f^{valence} = \sum d_j \, \phi_j^{valence} \tag{3.7a}$$

$$\boldsymbol{d}^\dagger \, V\boldsymbol{d} = 1 \tag{3.7b}$$

Let S be the rectangular core-valence overlap matrix, computed in BASCOR, $S_{ij} = \phi_i^{core} \, \hat{o} \, \phi_j^{valence}$. In BASOVL the valence set has already been orthogonalized on the core, but the matrices V and S above refer to the valence set *before* the core-orthogonalization; the core-orthogonalized functions are denoted $\bar{\phi}^{valence}$ and their overlap matrix W.

$$\bar{\phi}_i^{valence} = \phi_i^{valence} - \sum_j \; S_{ji} \, \phi_j^{core} \tag{3.8}$$

W is the overlap matrix actually computed in BASOVL. The relation between V and W is

$$V_{ij} = \phi_i^{valence} \, \hat{o} \, \phi_j^{valence} = \bar{\phi}_i^{valence} + \sum_k \; S_{ki} \, \phi_k^{core} \, \hat{o} \, \bar{\phi}_j^{valence} + \sum_l \; S_{lj} \, \phi_l^{core} =$$

$$= W_{ij} + \sum_{kl} \; S_{ki}^{*} \, S_{lj} \, \delta_{kl} = W_{ij} + (S^\dagger \, S)_{ij} \tag{3.9}$$

where we used the orthonormality of the core set and orthogonality of $\bar{\phi}^{v}$ on all core functions. The overlap between a general core and valence function is

$$O \equiv \sum_i \phi_i^{core} \, \hat{o} \, \sum d_j \, \phi_j^{valence} \equiv \boldsymbol{c}^\dagger \, S\boldsymbol{d} \tag{3.10}$$

O is a complex number. We maximize therefore the real scalar OO^* under variation of the coefficients $\boldsymbol{c}$ and $\boldsymbol{d}$. The normalization conditions are incorporated with Lagrange multipliers. Define

$$F = OO^* - \lambda \boldsymbol{c}^\dagger \, \boldsymbol{c} - \mu \boldsymbol{d}^\dagger \, V\boldsymbol{d} = \boldsymbol{c}^\dagger \, S\boldsymbol{d}\boldsymbol{d}^\dagger \, S^\dagger \, \boldsymbol{c} - \lambda \boldsymbol{c}^\dagger \, \boldsymbol{c} - \mu \boldsymbol{d}^\dagger \, V\boldsymbol{d} \tag{3.11}$$

From the variational equations we derive

$$\frac{\partial F}{\partial \boldsymbol{c}^\dagger} = 0 \qquad S\boldsymbol{d}O^* - \lambda \boldsymbol{c} = 0 \qquad \boldsymbol{c} = \frac{O^*}{\lambda} \, S\boldsymbol{d} \tag{3.12}$$

$$\frac{\partial F}{\partial \boldsymbol{d}^\dagger} = 0 \qquad OS^\dagger \, \boldsymbol{c} - \mu V\boldsymbol{d} = 0 \qquad S^\dagger \, S\boldsymbol{d} = \frac{\mu\lambda}{OO^*} \, V\boldsymbol{d} \tag{3.13}$$

Multiplication from the left by $V^{-1/2}$ gives

$$(V^{-1/2} S^\dagger \, SV^{-1/2})(V^{+1/2}\boldsymbol{d}) = V^{-1/2}\left(\frac{\mu\lambda}{OO^*}\right) V^{+1/2}(V^{+1/2}\boldsymbol{d}) \qquad G\boldsymbol{d}' = \varepsilon \boldsymbol{d}' \tag{3.14}$$

with

$$G = V^{-1/2} S^{\dagger}\, SV^{-1/2} \qquad \varepsilon = \frac{\mu\lambda}{OO^*} \qquad \boldsymbol{d}' = V^{+1/2}\boldsymbol{d} \tag{3.15}$$

(3.14) is an eigenvalue equation for the hermitian matrix *G*. Combining (3.10) and (3.12) gives

$$O \equiv \boldsymbol{c}^{\dagger}\, S\boldsymbol{d} = \boldsymbol{c}^{\dagger}\, \boldsymbol{c}\, \frac{\lambda}{O^*} \qquad \lambda = OO^* \tag{3.16}$$

Similarly

$$O \equiv \boldsymbol{c}^{\dagger}\, S\boldsymbol{d} = \left(\frac{O}{\lambda}\, \boldsymbol{d}'^{\dagger} S^{\dagger}\right) S\boldsymbol{d} = \frac{O}{\lambda}\, \boldsymbol{d}^{\dagger}\, V^{-1/2} S^{\dagger}\, SV^{-1/2}\boldsymbol{d}' = \frac{O}{\lambda}\, \varepsilon\, \boldsymbol{d}'^{\dagger}\, \boldsymbol{d}' \qquad \varepsilon = \lambda \tag{3.17}$$

So that $\varepsilon(=\lambda=\mu)=OO^*$. The maximum overlap between core and valence can then be defined as $(\varepsilon_{max})^{1/2}$.

implementation

In BASCOR, where the original valence set is orthogonalized on the (orthonormal) core, the matrix $(S^{\dagger}\, S)$ is computed from the core-valence overlap matrix *S*. The variables in the program are sdagsr and sdagsi for the real and imaginary parts respectively

This is added in BASOVL to *W* (ovlre, ovlim), the overlap matrix of the valence set that has been orthogonalized on the core: $V=W+(S^{\dagger}\, S)$. *V* is then diagonalized to construct $V^{-1/2}$ from its eigensystem:

$$V = U\Omega\, U^{\dagger} \qquad V^{-1/2} = U\Omega^{-1/2}\, U^{\dagger} \tag{3.18}$$

Next we obtain *G* by the similarity transformation (3.15) (routine SIMTRF). Finally *G* is diagonalized to find the maximum eigenvalue ε_{max} . $\delta\equiv(1-\sqrt{\varepsilon_{max}})$ is used as the measure for the core-valence dependency. (This is analogous to the smallest eigenvalue of the overlap matrix of a set of functions.) δ is compared with the criterium scrcv (input by key `scrcv` or `core-valence`; defaults 10^{-4} , 10^{-5}).

Like in the analysis of the valence basis itself, dependency coefficients are computed (3.4), which tell us which valence functions are dominant in the overlap with the core space. If the valence and core are found to be dependent the program stops (in BASOVL) and prints the dependency coefficients.

the Stable State Approach (SSA)

The Stable State Approach (SSA) is a proposal for (future) improvement of accuracy and efficiency of the SCF procedure. Conceptually it resembles the frozen core approximation; it is based on the following consideration: the convergence of the one-particle eigenstates $\psi_n(\boldsymbol{k};\boldsymbol{r})$ towards the final self consistent solution will in general be quite different for the various states. It is well conceivable that some states attain rapidly their final form while others converge slowly. If we keep track of the developments and remove

from the valence basis the states that do not change anymore, the efficiency is enhanced in the remaining cycles: less data to keep on file, fewer matrix elements to compute, etcetera.

This is (partially) counterbalanced by the necessary transformation of the basis to single out the stable states. Analogous to the frozen core treatment the new function basis must be orthogonal to the fixed states. The transformation overhead scales as the basis size squared and may therefore be appreciable. Possibly a net gain results only if several stable states are removed at a time; such 'details' should of course be incorporated in a thoughtful implementation.

It may be noted here, as a side remark, that the reduction of the function space may not come only (or not even predominantly) from finding stable *occupied* states, but also from the detection and removal of stable *virtual* states.

basic set-up

The eigenstates are expressed in the basis functions: $\psi_i = \sum c_{ij}\, \phi_j$. A state ψ_i is stable if its coefficient vector $\{c_{ij}\}_j$ *and* the number of electrons in it, the occupation number n_i are constant within some tolerance. This is easily checked if we have the coefficient matrices and occupation numbers on file for the last few cycles.

In this formulation some of the possibilities of the SSA may be missed. Two examples:
1) suppose ψ_1 and ψ_2 are two degenerate occupied states with, of course, equal occupation numbers. Obviously any orthogonal transformation of $\{\psi_1, \psi_2\}$ yields the same physical situation. So, if we have a stable space, spanned by ψ_1 and ψ_2 , we may not detect it when the computed eigenvectors rotate arbitrarily around from one cycle to another.
2) suppose $\psi_1 .. \psi_N$ are (non-degenerate) virtual states and at the next cycles we find *N* virtual states spanned by the same functions. We can then remove these functions as they are apparently irrelevant for the crystal electrons; the individual virtual eigenstates need not be stable for this, only the space spanned by them.
Both these cases are detected if we do not examine the individual eigenstates but the P-matrices (density matrices). Denote by P^n the P-matrix in the *n*-th cycle and choose as its representation not the employed basis functions but the eigenstates of cycle n_0 . P^{n_0} is then diagonal and the diagonal elements are the occupation numbers. A state ψ_i is stable and can be singled out if at the following cycles the corresponding diagonal element P_{ii} remains the same *and* its off-diagonal elements P_{ij} , $j \neq i$, are zero.

The criterium for stability has to be determined by experimentation. It must scale somehow with the mixing parameter parmix employed in the iterative update of the crystal potential (SS^iteration).

the SSA and the frozen core

The frozen core approximation has some fundamental problems. In the first place the core states may not be strictly orthonormal. Whereas they are explicitly orthonormalized in order to facilitate the projecting out of

the core components from the valence basis, the corresponding energy term is not evaluated and implicates thus an unknown error in the computed energy. The tests on the eigenvalues of the overlap matrix, as discussed above, is only a very rough safety guard against extreme cases.
In the second place the relaxation of the crystal potential, especially its change near the nuclei means that even if the core states were orthonormal from the start, they may not be eigenstates of the final crystal hamiltonian. This has consequences both for the energy of the core space and for the valence states. This aspect has usually no severe consequences but nevertheless it is unsatisfactory.

The SSA alleviates this problem because we may include (part of) the core in the valence basis. The corresponding crystal eigenstates will soon be converged (if the frozen core assumption is reasonable) so that they have to be processed only during a few cycles. In principle we could thus discard the frozen core idea altogether in the SSA, but in some situations (heavy atoms, many *k*-points) the enlarged basis size and the resulting amount of data on file may be problematic, even if it would be for only two or three cycles. (Data storage can and should be reduced however by a more sophisticated treatment of symmetry than it is currently being done in BAND (SS^symmetry)).

4 BZ-integration

Two types of integrals over the Brillouin Zone occur: the 'volume' integral and the 'surface' integral, respectively

$$J(E) = \sum_n \int f_n(\boldsymbol{k})\ \theta(E - e_n(\boldsymbol{k}))\, \mathrm{d}\boldsymbol{k} \tag{4.1a}$$

$$I(E) = \sum_n \int f_n(\boldsymbol{k})\ \delta(E - e_n(\boldsymbol{k}))\, \mathrm{d}\boldsymbol{k} = \sum_n \int_{e_n(\boldsymbol{k})=E} \frac{f_n(\boldsymbol{k})}{\hat{\mathrm{o}}\ e_n(\boldsymbol{k})\hat{\mathrm{o}}}\, \mathrm{d}S \tag{4.1b}$$

$e_n(\boldsymbol{k})$ is the dispersion relation of the n-th energy band and $f_n(\boldsymbol{k})$ is the contribution to the integral from the one-electron state $\psi_n(\boldsymbol{k};\boldsymbol{r})$. The results depend on the energy parameter E. The surface and volume integral are related by $I(E)=\mathrm{d}J(E)/\mathrm{d}E$.
Example: the electronic charge density $\rho(\boldsymbol{r})$ is given by the volume integral

$$\rho(\boldsymbol{r}) = \sum_n \int \hat{\mathrm{o}}\,\psi_n(\boldsymbol{k};\boldsymbol{r})\,\hat{\mathrm{o}}^2\ \theta(e_F - e_n(\boldsymbol{k}))\, \mathrm{d}\boldsymbol{k} \tag{4.2}$$

e_F is the fermi energy.
As discussed in SS^symmetry the integrations can effectively be restricted to integrals over the symmetry unique region in *k*-space, the irreducible wedge of the BZ; we will take this for granted here.

The eigensystems ($e_n(\boldsymbol{k})$, $\psi_n(\boldsymbol{k},\boldsymbol{r})$, and hence any derived quantities $f_n(\boldsymbol{k})$) are computed in discrete points of the BZ. The integrals (4.1a) and (4.1b) are thus evaluated numerically. The presence of the θ- and δ-functions in (4.1) makes 'normal' methods of numerical integration, such as Gauss-Legendre, unsuitable. BAND employs the (repeated) analytic quadratic procedure [Wiesenekker *et al.* 1988, Wiesenekker and Baerends 1990]: the (irreducible wedge of the) BZ is first written as a conjunction of (large) *basic* simplices; these are further partitioned into smaller simplices. In each (small) simplex $e_n(\boldsymbol{k})$ and $f_n(\boldsymbol{k})$ are both approximated by a quadratic form; the resulting quadratic integrals are evaluated exactly (analytically).

The precision in the total accumulated integral can be increased by using a finer partitioning of the BZ. Whereas the presence of the θ- and δ-function makes the integrals (4.1) a little complicated, $e_n(\boldsymbol{k})$ and $f_n(\boldsymbol{k})$ themselves are smooth. The piecewise quadratic approximations converge therefore rapidly with the partitioning (compare the commonly used repeated Simpson integration for 1D line integrals). Since the sub integrals, employing the quadratic forms, are calculated exactly, the total integral approximation is expected to be accurate and quickly converging. This is borne out in practice.

For the construction of the quadratic approximations in a given simplex we need the function values of $e_n(\boldsymbol{k})$ and $f_n(\boldsymbol{k})$ in sufficiently many points to solve the defining linear systems of equations. It is convenient is to use the vertices and the midpoints of the edges of the simplex.
Even without a further partitioning this implies already several *k*-points for each basic simplex. BAND optionally uses fewer points, but of course a quadratic approximation is not possible then.

The integration accuracy is determined by the parameter kinteg. Kinteg is the number of sample points on any edge of a (large) basic simplex. Kinteg=1 implicates the use of only the Γ-point $\boldsymbol{k}$=0; the numerical integral over the BZ is then reduced to a single term; we may call this the zero-th order approximation.

Kinteg=2 leads to first order (=linear) approximations: the basic simplices that span the irreducible wedge of the BZ are generated; the only integration points are the vertices of these simplices, the midpoints of the edges are not involved. A linear approximation of the functions is then feasible and the ensuing integrals are evaluated analytically.

Kinteg=3 is the lowest setting that allows a quadratic approximation; the basic simplices are used directly, i.e. no further partitioning takes place. With kinteg=5 every edge of a basic simplex is bisectioned, yielding two intervals with 3 points each (one point is shared). Each basic simplex is then divided into 2^n smaller simplices, in each of which the quadratic method is applied (*n* is the dimensionality of the BZ). Similarly with kinteg=7 we get 3^n smaller simplices and so on.
When kinteg is chosen *even* (4,6,..) it is not possible to have a partitioning of non-overlapping intervals such that each has 3 points; one remains with only two points so that in some part of the BZ we have to resort to the linear approximation. In such a case the linear method is applied throughout: all edges are partitioned into intervals with two points only. This is the approach of the commonly employed 'linear tetrahedron method' [Lehman and Andersen 1972], which is by far inferior in performance to the quadratic

one [Wiesenekker *et al.* 1988, Wiesenekker and Baerends 1990] . For higher integration accuracies (kinteg≥3) one should therefore stick to *odd* values.
Kinteg is input; key: `k integ`, `k space`, or `BZ`; first and second defaults: 3, 5. A (non-fatal) warning message is issued by the program when *K* is chosen even.

data structure

KPNT generates the data structure needed for the BZ integration: a list of kt *k*-points xyzkpt(3,kt) and a list of nsimpl simplices, represented as a pointer set ksimpl(nvertk,nsimpl). Nvertk is the number of points associated with each simplex; this depends on the type of approximation and on the dimensionality; in two dimensions for instance nvertk is 3 for the linear approximation (vertices only) and 6 for the quadratic approximation (with the midpoints).
The values ksimpl(i,j), *i*=1..nvertk indicate which points in the list xyzkpt() belong to the *j*-th simplex.

The generated simplices span precisely an irreducible wedge of the BZ. Some of the *k*-points may nevertheless be symmetry equivalent, e.g. by a Bravais translation of the reciprocal lattice. KPNTEQ, called from KPNT, checks the symmetry relations and generates a list of *equivalence indices* kequiv(kt). Kequiv(k_0) is the index of a point equivalent to point k_0 . For all k: kequiv(k)≤k; the symmetry unique points have kequiv(k)=k. Only in these points the hamiltonian equation is solved. The other points serve to describe the simplices and to generate the (linear or quadratic) function approximations. The total number of symmetry unique *k*-points is kuniqu.

The data arrays xyzkpt, ksimpl and kequiv are stored on file itbz; the scalar variables kt, kuniqu, nsimpl and nvertk are in common FIXDAT.

simplices and points

The dimensionality of the BZ is ndimk and the primitive *k*-space lattice vectors are stored in the left-upper ndimk×ndimk part of array bvec(3,3).

ndimk=0
This trivial case implies the zero-th order approximation: only the Γ-point, one 'simplex', one point per simplex (nvertk=1).

ndimk=1
As inversion is always a *k*-space symmetry operator, the irreducible wedge is the interval $(0, \frac{1}{2}$ bvec(1,1)). This is also the basic simplex. Depending on kinteg exceeding 3, the interval is partitioned to obtain smaller simplices.

The partitioning of a simplex in smaller ones is (for any dimensionality) performed by SIMPLS. SIMPLS delivers both the pointer structure ksimpl() and the list of points xyzkpt(). In case of more than one basic

simplex (in two or three dimensional regions, see below) SIMPLS is called for each of them. SIMPLS updates the data arrays at every call, merging the data resulting from the current basic simplex with the existing structure.

ndimk=2

LATTPT generates a few stars of lattice points (in *k*-space) around the origin. The bisecting 'planes' (lines) halfway them define the Wigner-Seitz BZ as the polygon inside these lines. POLYGN removes the redundant lines to retain only the sides of the polygon. POLYG1 computes then the vertices of the polygon and PLGIRR calculates the vertices of the irreducible wedge (a polygon again). The basic simplices are now the triangles spanned by the origin and the edges of the irreducible polygon. For each of these SIMPLS is called for further partitioning into smaller triangles.

ndimk=3

LATTPT gives again the surrounding lattice points; the planes halfway them define the Wigner-Seitz polyhedron. POLYHE removes the redundant planes and computes also the vertices of the polygonal faces of the polyhedron. The symmetry unique faces are subsequently rotated to the xy-plane (PYRROT), reduced to the irreducible part (PLGIRR) and then back rotated again to the original frame. The basic simplices are defined by the origin and the triangles that span the irreducible polygons on the faces. SIMPLS may further partition them.

The geometric routines POLYGN, PLGIRR, POLYHE and SIMPLS are discussed in SS^geometry.

Since SIMPLS only delivers the (ndimk+1) vertices of the small simplices we have to add the midpoints of the edges later (for the quadratic approximation). This is done in KPNT after the generation of all simplices. The pointer array ksimpl() is updated with pointers to the midpoints.

integration

The BZ integrals are evaluated numerically. The fixed set of integration points xyzkpt() is stored on file itbz. The weights $o_n(\boldsymbol{k})$, that is, the occupation numbers for the one-electron states $\psi_n(\boldsymbol{k};\boldsymbol{r})$ are computed by OCCUPA from the energy bands $e_n(\boldsymbol{k})$ and an energy parameter E (4.1).
During the SCF procedure we need only one specific integral over the BZ: the charge density

$$\rho(\boldsymbol{r}) = \sum_{n\boldsymbol{k}} o_n(\boldsymbol{k}) \; \hat{o}\, \psi_n(\boldsymbol{k};\boldsymbol{r}) \,\hat{o}^2 \qquad (4.3)$$

The constraint that the density contain the number of electrons qelec,

$$\sum_{nk} o_n(\boldsymbol{k}) = \text{qelec} \qquad (4.4)$$

defines the fermi energy e_F (the energy parameter E in (4.1a)) .

FERMIE repeatedly calls OCCUPA with trial values for the fermi energy until condition (4.4) is satisfied. The resulting occupation numbers are then passed back to the master routine FERMI, which calls OCCSTA. OCCSTA writes them to file, together with the corresponding eigenstates for further processing later. Since the energy bands $e_n(\boldsymbol{k})$ change as the SCF iterations proceed the fermi energy and the occupation numbers may also vary; they are computed anew at every cycle.

OCCUPA sets up a loop over the (small) simplices in which the irreducible BZ has been partitioned and calls OCCUPS for each of them, to obtain the occupation numbers for that simplex. OCCUPS discriminates several cases: in the first place whether a surface or a volume integral is requested, in the second place the order of the interpolation (linear or quadratic) and finally the dimensionality ndimk of the BZ.

linear approximation

The algorithm is discussed in detail by Wiesenekker *et al.* [1988]. The formulas have been implemented in BZINTL.

quadratic approximation

The 1D case is more or less obvious (routines VIQD1 and VJQD1 for surface and volume integration respectively). For the 2D case we refer to Wiesenekker *et al.* [1988], where all technical aspects are explained. The formulas have been implemented in QUAD2 and a few auxiliary routines. The 3D case finally is fairly complicated [Wiesenekker and Baerends 1990]. Implementation (QUAD3) is on its way. A satisfactory alternative, currently applied in BAND, is the hybrid-quadratic method: routine HYBRID.

hybrid-quadratic approximation

As discussed by Wiesenekker *et al.* [1988] we need for the quadratic method the integrals

$$V_h(E) = \int_{simplex} h(\boldsymbol{k})\, \theta(E - e_n(\boldsymbol{k}))\, d\boldsymbol{k} \qquad (4.5)$$

where the functions $h(\boldsymbol{k})$ are all monomials up to second order (and similarly for the surface integrals with the θ-function replaced by the δ-function, c.f. (4.1a) and (4.1b)). The occupation numbers are computed from the monomial integrals V_h as fixed linear combinations of them. The fundamental problem is therefore to compute the integrals V_h .
In the hybrid method we construct first explicitly the quadratic approximation of the energy dispersion $e_n(\boldsymbol{k})$ in a particular simplex (one of the small simplices in which the BZ has been divided): $e^{quad}(\boldsymbol{k})$ say. This simplex is then partitioned in still smaller simplices (using the same algorithm as in SIMPLS. For each of these sub-simplices the energy values $e_n(\boldsymbol{k})$ in the vertices are computed from the analytical form $e^{quad}(\boldsymbol{k})$ and used to calculate the monomial integrals with the *linear* interpolation method (BZINTL). The contributions from all small sub-simplices are added and yield finally the required monomial integrals over the original simplex.

The hybrid method is essentially a two step proces: first a quadratic approximation is made. This is then further approximated by a piecewise linear form. Obviously the precision of the result depends on the hybrid partitioning. The number of additional hybrid partition steps is kmesh. This is input by the key `kmesh`; first and second defaults are kmesh=2 and kmesh=3. Thusfar we have never encountered a system where the values 2 and 3 (or higher) resulted in significantly different results. Apparently the first default value is adequate.

remark

The hybrid method is often used in a different form by integrating only the monomials up to order 1 (instead of 2) over the sub simplices [MacDonald *et al.* 1979]. The compounded integrals over the large simplex are then used to compute the occupation numbers, but for lack of information, these correspond of course to an approximation of the property function $f(\boldsymbol{k})$ to first order only. If we assume that the hybrid partitioning is pushed to its limit, then our form approaches the analytic quadratic results, while the simplified form is equivalent to the linear-quadratic approximation: linear approximation of $f(\boldsymbol{k})$ and quadratic approximation of $e_n(\boldsymbol{k})$; as demonstrated by Wiesenekker *et al.* [1988] this gives results that are by far inferior to the fully quadratic approximation.

temperature

BAND attempts to determine the occupation numbers in accordance with a finite temperature (the fermi-dirac distribution). In routine FERMIE, where the fermi energy and the occupation numbers are determined, we have to call OCCUPA therefore with a (small) array of energy values instead of only the (trial) fermi energy. The energy values are distributed around the trial fermi energy and the resulting sets of occupation numbers, one set for each of the energies, are combined; this amounts to a numerical integration of the fermi-dirac distribution. See SS^temperature for a more detailed account.

efficiency

The determination of the fermi energy (FERMIE) at every cycle of the SCF procedure could be rather time consuming (especially in 3D crystals) due to the following inefficiency. For each trial fermi energy all bands might be analysed, piecewise-quadratically fitted etcetera to determine the occupation numbers. In general however the majority of the bands are either completely occupied or empty. The occupation numbers for such bands do not depend on their energy functions $e_n(\boldsymbol{k})$. It is therefore unnecessary to compute them again and again. This aspect is taken care of as follows

a) the fixed set of occupation numbers for all *k*-points in a completely occupied band are determined at the start of the SCF procedure and stored in a separate array.

b) at every cycle, when the fermi energy and the occupation numbers are to be determined, the energy range of every band is computed first. For each trial energy it is then quickly surveyed which bands are empty, fully occupied or to be analyzed in more detail. The determination of the band widths occurs in routine EMNMXB, with three auxiliary routines EMNMX1, 2 and -3 (for 1-, 2- and 3D crystals). The same piecewise quadratic approximations for the energy bands are applied in fact by EMNMXB, but this is done only once (per cycle), not for every trial (fermi-)energy.

In one case (at least) the occupation numbers are zero for some specific *k*-points, even if the band is not empty. This known special case occurs for the quadratic integration in a 2D crystal. Each fully occupied simplex has zero occupations for the vertices (and occupations 1/3 for the mid-points of the edges). This is an accidental effect of the integration scheme [Nooijen *et al.* 1990, Wiesenekker *et al.* 1988]. We do not know whether there are more of such special cases.
Suppose now that for *all* bands the occupation numbers are zero in a particular *k*-point. If this remains so during the whole calculation it must be possible to avoid most of the computational effort associated with that *k*-point, such as the evaluation of matrix elements and so on.
Thusfar we have not worked out how to proceed with this possibility. As a preliminary for future work on this we have implemented an array kzero(kt) in SCF, which is passed on to several sub-ordinated routines. kzero stores for each *k*-point whether (and in how many subsequent cycles) the occupation numbers are zero for all bands. Output messages are issued whenever such *k*-points are found.

5 Charge density

evaluation

The solution of the hamiltonian equation in each *k*-point yields (EIGSYS) the one-electron eigenstates as linear combinations of the basis functions

$$\psi_n(\boldsymbol{k};\boldsymbol{r}) = \sum_j c_{nj}(\boldsymbol{k})\ \phi_j(\boldsymbol{k};\boldsymbol{r}) \tag{5.1}$$

The analysis of the energy bands $e_n(\boldsymbol{k})$ gives (FERMIE) the occupation numbers $o_n(\boldsymbol{k})$. There are then two ways to compute the charge density in the crystal integration points.

1. From the eigenstates:

$$\rho(\boldsymbol{r}) = \sum_{n\boldsymbol{k}} o_n(\boldsymbol{k})\ \rho_n(\boldsymbol{k};\boldsymbol{r}) \tag{5.2}$$

$\rho_n(\boldsymbol{k};\boldsymbol{r})$ is the crystal orbital density

$$\rho_n(\boldsymbol{k};\boldsymbol{r}) = \left|\sum_j c_{nj}(\boldsymbol{k})\ \phi_j(\boldsymbol{k};\boldsymbol{r})\right|^2 \tag{5.3}$$

With kt *k*-points, nbas basis functions and nband occupied states the amount of work per integration point is proportional to kt×nband×nbas.

2. From the density matrices. First we construct the P-matrix (PMATRX) in each *k*-point.

$$P_{ij}^{\boldsymbol{k}} = \sum_{n=1}^{nband} \sum_{ij} c_{ni}^{*}(\boldsymbol{k})\ c_{nj}(\boldsymbol{k}) \tag{5.4}$$

Since $P^{\boldsymbol{k}}$ is hermitian we have to store and process only the upper triangle; define

$$\begin{aligned} \bar{P}_{ij}^{\boldsymbol{k}} &= 2\ P_{ij}^{\boldsymbol{k}} \qquad i<j \\ \bar{P}_{ii}^{\boldsymbol{k}} &= P_{ii}^{\boldsymbol{k}} \end{aligned} \tag{5.5}$$

This gives

$$\rho(\boldsymbol{r}) = \sum_{\boldsymbol{k}} \sum_{i\le j} \mathrm{Re}\ [\bar{P}_{ij}^{\boldsymbol{k}}\ \ \phi_i^{*}(\boldsymbol{k};\boldsymbol{r})\ \ \phi_j(\boldsymbol{k};\boldsymbol{r})\ ] \tag{5.6}$$

The amount of work per integration point is proportional to kt×nbas×(nbas+1)/2.
The construction of the P-matrices themselves can be neglected as they do not depend on the integration points.

Depending on the number of occupied bands and the number of basis functions one or the other method is more efficient. The break-even point may be expected to occur for nband≈nbas/2, but of course it is also determined by details of the computational procedures, in particular by the types of (vector) operations involved and hence it may even be machine dependent.
In big calculations the evaluation of the charge density is a major part of the SCF procedure and it is crucial to pick the most efficient method. Both ways have been implemented; RHOPSI uses the eigenstates, RHOPMT employs the density matrices. The times used for them are measured and stored in an array rhotim(2). The values are compared by SCF to make its choice.
Rhotim is initiated at zero for both and either of the two is used at the first cycle; its timing is then updated. At the next cycle the other method is applied because it has still zero time-measurement and appears therefore more efficient. At the third cycle a sensible decision can be made.

remarks

1. The execution times may change somewhat from cycle to cycle, for instance as a consequence of varying circumstances in the computer during the run. Future developments of BAND (see e.g. SS^basis: the SSA) may also induce variations in the cycle-times. To keep the decision procedure flexible, SCF reduces by a small amount at each cycle the stored time-measurement of the not-used alternative. Eventually this will cause it to appear more efficient than its rival, so that it is applied and clocked again.
   The disadvantage that the less efficient procedure is used from time to time is relatively small because the artificial reduction of the stored timing is small (5% per cycle).

2. As discussed in SS^SCF-linearization the charge density is at some cycles evaluated as a linear combination of previous densities. This is then a completely different situation of course and the timings of the exact evaluations via RHOPSI and RHOPMT are not updated.
   The approximate evaluation does not depend on the choice discussed above: the expansion coefficients are determined by comparing the density matrices P^k , k=1..kt with those of previous cycles (so these matrices have to be computed anyway). The corresponding density functions are on file itrstr and have only to be combined (see SS^charge density). This could be done in either routine, RHOPSI or RHOPMT. After the *exact* evaluation however the data on itrstr have to be updated. The required additional lines of code have therefore been implemented in both routines. On general principles SCF chooses between the two routines also in the approximation case. The array rhotim has thus four elements instead of two: rhotim(2,2).

analysis

The file with integration points, itpnt, contains for each point the index of the atom nearest to that point: abs(idatom()); positive and negative values of idatom indicate whether the point is inside the atomic sphere or in the interstitial region. Routine CHARGE, called from the properties section PRPRTS, applies this information to calculate the distribution of the final self-consistent density over the atoms. This is done both for the initial sum-of-atoms charge and for the deformation density. The data are presented separately for the densities inside and outside the atomic spheres. In spin-polarized calculations the analysis is performed for each spin.
The file itdatm contains the sum-of-atoms total density and valence density in all integration points; the crystal valence density is on itrho.

The computed spin and charge polarizations may be compared with the Mulliken populations calculated in POPANA. Often the agreement is poor. The discrepancies may be attributed to the arbitrariness in both methods. In the Mulliken analysis the choice of basis functions influences the outcome, whereas in the numerical integration employed by CHARGE, one may doubt the association of points with the atoms nearest by: heavy atoms might for instance be attributed more space around them than hydrogen.
Any criterium related to the distances and e.g. the nuclear charges might be used and is easily implemented. The index numbers idatom() are determined in RPNTID; adaptations of the algorithm should be straightforward.

further developments
The analysis in CHARGE is very global and one would like to have more detailed information. An important possibility is to generate *plots* of the density. This would be an interesting extension of BAND, because such pictures provide a direct and instructive survey of the charge distribution; they are frequently published and discussed in the literature. A relatively simple and efficient way to compute the desired values in some plot-grid is via the fit functions. The fit set is usually fairly accurate, but naturally the principal disadvantage remains that it is not exact and some particular features of the distribution may be missed in this way.

A general drawback of density plots is that the information is only two-dimensional. A three-dimensional numerical analysis is useful as support and may also be valuable in itself. We may for instance expand the density inside the atomic spheres in spherical harmonics. The numerical integration grid is well suited for this: for a sequence of radial distances an angular integration mesh is present. The maximum possible angular momentum value for the expansion depends on this mesh, varies with the radial value, but is usually substantially higher than the values occurring in the fit set. The total density as well as the deformation charge can then be described in detail. The data may be used to generate one-dimensional radial plots of the densities for each (lm)-component separately.

6 Control

BAND contains a general control mechanism, which we will refer to as the *controller*. The purpose of it is twofold. In the first place many tasks performed by the program are relatively independent. The controller may be used to manipulate the execution of various operations, e.g. via instructions on the input file. This will especially be useful when, in the future, restart possibilities are incorporated in a general way.
For instance we may have obtained the self consistent solution of some system and wish to know in what respects the outcome is influenced when plane waves are added to the valence basis, what happens when the temperature is increased, whether a spin-*un*restricted calculation would make some difference, or we may be interested in a few particular partial densities of states that have not been computed in the original run. In all such cases we do not want to duplicate work. The controller should take care of that when supplied with the appropriate data on file and a few instructions.
In the second place the control mechanism may be of some help to uncover and analyze errors, by checking relevant data and printing information when a problem is detected.

Many of the data related to the control structure are stored in the common blocks CNTRLC (character-type data) and CNTRLV (other variables). The controller in its present form is only a first, rather primitive structure which might be embedded in a more sophisticated set-up.
A summary of the current implementation:

The execution of all major subroutines and sections in BAND is enclosed between calls to two special routines, START and ENDOF.

```
call start('calc',iopt,execut)
if (execut) then
  ..
   call calc(....)                                          (6.1)
   ..
   call endof('calc',jopt)
endif
```

START determines whether CALC is to be executed. The decision is output in the logical execut. If execut is true the name 'calc' is added by START to an 'execution stack', the list of names exstck(). Routine ENDOF removes 'calc' again from the stack and checks whether the run should be stopped at this point.

Execut is computed in START by calling SKIP('calc'). The logical function SKIP compares its argument with a list of sections to be skipped, skplst(). The skip-list is empty at the start-up of the program; entries can be added via input, with the key `skip` (see SS^input).

ENDOF compares its first argument with the character variable exlast in common CNTRLC, which stores the name of the last section or routine to be executed; if they are equal ENDOF calls STOPIT to terminate the run. By default exlast equals 'band', implying that the whole program is to be executed. Via input, with the key `execute` (SS^input), another last-to-be-executed routine or section may be specified. Another variable, exfrst, states with what section the program should start. As no restart possibilities have yet been implemented this variable can effectively not be used and must have the default value 'band'.

The admissible arguments for the input commands `skip` and `execute` must be names that are recognized by the controller: they must correspond to the arguments of START and ENDOF as implemented in the program. See SS^input for more comments on these keys.

The integer option-arguments of START and ENDOF, iopt and jopt in example (6.1) above, specify additional actions to be undertaken by these routines. Currently this aspect is hardly used. START optionally (iopt=1) copies the name of the section to the day-file. This provides information as to in which stage the calculation is at a certain moment. ENDOF optionally calls the timing routine SECTIM. (SECTIM keeps track of how many times a certain section is executed and the amount of CP-time used there (i.e. the time lapse since the previous call of SECTIM). The data structure of SECTIM is initialized by ITIMER (called from INIT) and all information is output by TSTAT, called from STOPIT).

STOPIT is the routine which terminates the program after executing a few final tasks. STOPIT is called whenever a fatal problem is detected in some way, and also when the controller decides (ENDOF) that the calculation has to be finished. STOPIT outputs

- the state of affairs in the file manager: the files currently in use by BAND, the pointer structure that defines the relation between the files and the amount of data on them (SS^files),

- the situation in the workspace manager: the currently allocated arrays and the markers around them (SS^workspace),
- the contents of several common blocks; the variables stored in them represent important aspects of the calculation and may give a clue to the cause of some problems,
- the execution stack, telling us in what stage of execution the calculation was terminated.

Since STOPIT calls other subroutines, in particular those of the file manager, which may themselves call STOPIT in case of errors, we have a potential recurrency. The ensuing problems are circumvented with a variable istop in common CNTRLV. Istop is set to zero in INIT. The first actions in STOPIT are

```
if (istop.ne.0) stop'recurrency'                    (6.2)
istop = istop + 1
```

This assures that (infinite) recurrency loops are cut short.

A few important aspects of BAND's control mechanisms have been organized in specific structures that are discussed in other sections. These are

- the use of files: SS^files.
- the utilization of work space: SS^workspace.

7 Coulomb potential and lattice sums

The charge density in the crystal is given by a sum over occupied orbitals, expressed in basis functions ϕ:

$$\rho = \sum_i n_i \, |\psi_i|^2 = \sum_{ij} P_{ij} \, \phi_i^* \, \phi_j \qquad (7.1)$$

n_i are occupation numbers and the sum includes integration over the Brillouin Zone. P_{ij} is the density matrix in the representation of the basis functions. The coulomb potential is

$$V(\boldsymbol{r}) = \int \rho(\boldsymbol{r}') \, |\boldsymbol{r}-\boldsymbol{r}'|^{-1} \, d\boldsymbol{r}' = \sum_{ij} P_{ij} \int \phi_i^*(\boldsymbol{r}') \, \phi_j(\boldsymbol{r}') \, |\boldsymbol{r}-\boldsymbol{r}'|^{-1} \, d\boldsymbol{r}' \qquad (7.2)$$

The basis functions are (mostly) one-center functions: numerical orbitals from the free atom subprogram and/or Slater type orbitals. The integrals in the r.h.s. of (7.2) are hard to evaluate when ϕ_i and ϕ_j are located on different atoms. Therefore a set of fit functions f_i is introduced such that *a)* the true density is accurately approximated by a linear combination of them and *b)* their coulomb potentials f_i^c can be computed.
The crystal density is split in the superposition of atomic densities and the deformation density

$$\rho_{cry} = \sum_\alpha \rho_\alpha + \rho_{def} \qquad (7.3)$$

with coulomb potential (including the contribution from the nuclei)

$$V = \sum_{\alpha} V_{\alpha} + V_{def} \tag{7.4}$$

The atomic functions ρ_{α} and V_{α} are produced in tabular form by DIRAC and interpolated in ATMFNC to obtain their values in the crystal integration points. The fit functions are used for the deformation part of the density

$$\rho_{def} \approx \sum_{i} c_i f_i \tag{7.5a}$$

$$V_{def} \approx \sum_{i} c_i f_i^c \tag{7.5b}$$

The coefficients c_i are the least squares solution of (7.5a) with the constraint that the fitted density contain zero charge. Neglect of this condition, though leading to a more accurate description of the density, generates a potential corresponding to an incorrect amount of charge. The resulting error in the potential is much larger than the gain from the better density approximation.

The crystal fit functions constitute an orthonormal set (see below). The constraint is applied by the Lagrange multiplier technique. Define

$$v_i = \int \rho_{def} f_i \, d\mathbf{r} \tag{7.6a}$$

$$n_i = \int f_i \, d\mathbf{r} \tag{7.6b}$$

v_i would be the fit coefficient in absence of the constraint; n_i is the charge content of the fit function. The coefficients c_i are obtained from the usual variational equation, with $\sum c_i n_i = 0$

$$\partial/\partial c_j \left\{ \int (\rho - \sum c_i n_i)^2 \, d\mathbf{r} - \lambda \sum c_i n_i \right\} = 0 \qquad \lambda n_j = 2(v_j - c_j) \tag{7.7}$$

Multiply by n_j and sum over all j

$$\lambda = 2 \frac{\sum n_j v_j - \sum c_j n_j}{\sum n_j^2} = 2 \frac{\sum n_j v_j}{\sum n_j^2} \tag{7.8}$$

Reinsert into (7.7):

$$c_i = v_i - \frac{\lambda n_i}{2} = v_i - n_i \frac{\sum n_j v_j}{\sum n_j^2} \tag{7.9}$$

The fit coefficients c_i are computed in RHOFIT at every cycle of the self-consistency procedure. The fit functions, potentials and charge contents are on file itfit.

fit functions and fit potentials

Two types of fit functions may be useful. In the first place atomic one-center functions $\xi(\boldsymbol{r})=Z_{lm}(\Omega)\,P(r)$. Since the density is totally symmetric the crystal fit functions are symmetric combinations of the one-center functions (SS^symmetry); the one-center functions themselves are the *generating* (fit) functions. The options available in the program for $P(r)$ are Slater type functions $P(r)=r^{l+n}\,e^{-\alpha r}$ and numerical functions. The latter are the squares of the free atom (numerical) orbitals; the angular quantum number for the corresponding fit functions is set to l=0, so that the $l\neq 0$ components in the charge density have to be represented by the Slater type fit functions. See also SS^input.

To evaluate the potential $\xi^c(\boldsymbol{r})$ corresponding to a one-center function $\xi(\boldsymbol{r})$ the expansion of $\hat{o}\,\boldsymbol{r}-\boldsymbol{r}'\hat{o}^{-1}$ in spherical harmonics is applied

$$\xi^c(\boldsymbol{r}) = \int d\boldsymbol{r}'\, Z_{lm}(\Omega')\; P(r') \sum_{l'm'} \frac{4\pi}{2l'+1}\; Z^*_{l'm'}(\Omega')\, Z_{l'm'}(\Omega)\, \frac{r_<^l}{r_>^{l+1}} =$$

$$= \frac{4\pi}{2l+1}\; Z_{lm}(\Omega) \int_0^\infty \frac{r_<^l}{r_>^{l+1}}\,(r')^2\; P(r')\,dr' \equiv \frac{4\pi}{2l+1}\; Z_{lm}(\Omega)\; I(r) \tag{7.10}$$

For the numerical functions $P(r)$, given as a table $\{P(r_i)\}$ $I(r)$ is evaluated in routine COULOM, by numerical integration.
For an analytical function $P(r)=r^n\,e^{-\alpha r}$ we use the incomplete gamma function. Define

$$x = \frac{r'}{r} \qquad\qquad \beta = \alpha r \tag{7.11}$$

and apply

$$\int_0^1 x^k\; e^{-\beta x}\; dx = \frac{k!}{\beta^{k+1}} \left(1 - \sum_{i=0}^{k} \frac{\beta^i}{i!}\right) \tag{7.12}$$

to write

$$I(r) = \int_0^\infty \frac{r_<^l}{r_>^{l+1}} (r')^{n+2}\ e^{-\alpha r'}\ dr' = r^{n+2} \left(\int_0^1 x^{n+l+2}\ e^{-\beta x}\ dx + \int_1^\infty x^{n-l+1}\ e^{-\beta x}\ dx \right) =$$

$$= r^{n+2} \left(\frac{(n+l+2)!}{\beta^{n+l+3}} \left(1 - e^{-\beta} \sum_{i=0}^{n+l+2} \frac{\beta^i}{i!} \right) + \frac{(n-l+1)!}{\beta^{n-l+2}}\ e^{-\beta} \sum_{i=0}^{n-l+1} \frac{\beta^i}{i!} \right) =$$

$$= \frac{1}{r^{l+1}} \frac{(n+l+2)!}{\alpha^{n+l+3}} \left(1 - e^{-\beta} \sum_{i=0}^{n+l+2} \frac{\beta^i}{i!} \right) + \frac{(n-l+1)!}{\alpha^{n-l+2}}\ r^l e^{-\beta} \sum_{i=0}^{n-l+1} \frac{\beta^i}{i!} \qquad (7.13)$$

Some care is necessary in the numerical evaluation when αr is small. $e^{-\beta} \sum(\beta^i / i!)$ equals almost 1.0 then and subtraction from unity in the first term of the r.h.s. of (7.13) may lead to a loss of significant digits. This is remedied by using in such a case

$$1 - e^{-\beta} \sum_{i=0}^{m} \frac{\beta^i}{i!} = e^{-\beta} \sum_{i=m+1}^{\infty} \frac{\beta^i}{i!} \qquad (7.14)$$

The infinite series in the r.h.s. converges rapidly ($\beta << 1$) and can be truncated after a few terms.
The generating fit functions and the potentials are stored on file as tables of function values in a radial logarithmic mesh, like all other one-center functions in BAND. The radial values are computed in FITRAD, with auxiliary routines FITRA1 and FITRA2.

In the second place plane waves might be useful as fit functions (in 3D crystals). The potential functions are straightforward from the Poisson equation $-\Delta V = 4\pi\rho$:

$$f(\mathbf{r}) = e^{i\boldsymbol{K}\cdot\boldsymbol{r}}$$
$$f^c(\mathbf{r}) = \frac{4\pi}{K^2}\ e^{i\boldsymbol{K}\cdot\boldsymbol{r}} \qquad (7.15)$$

Again these have to be combined into symmetric combinations.
Plane wave fit functions have not (yet) been implemented in BAND and are in fact not necessary. The atomic one-center functions have proven to be very adequate in practice (see SS^energy for a note on the errors involved).

FITORT is the master routine for the crystal fit functions and potentials. FITPNT interpolates the radial tables, constructs the bloch sums and combines them into symmetric functions.
FITOVL computes (from the overlap matrix) the transformation to an orthonormal set. FITTRA performs the transformation and FITQ determines the charge contents n_i (7.6b) of the final orthonormal fit functions. All data are written to file itfit.

The eigenvalues of the overlap matrix are used to check linear dependency of the fit set. Eigenvectors with too small eigenvalues are discarded. The smallest eigenvalue allowed is scrfit. Scrfit is input with key `scrfit`, `fit overlap` or `fit dependency`; first and second defaults are 10^{-5} , 10^{-6} .
Dependency coefficients, analogous to the valence basis (SS^basis) are computed and printed, depending on the print option iprntp.

lattice sums

A crystal fit function is a (symmetrized) bloch sum of one-center functions. The potential function contains the bloch sum of a multipole potential $1/r^{l+1}$. This is exposed by rewriting (7.13) as

$$I(r) = \frac{(n+l+2)!}{\alpha^{n+l+3}} \frac{1}{r^{l+1}} + e^{-\alpha r} \left[\frac{(n-l+1)!}{\alpha^{n-l+2}} r^l \sum_{i=0}^{n-l+1} \frac{(\alpha r)^i}{i!} - \frac{(n+l+2)!}{\alpha^{n+l+3}} \frac{1}{r^{l+1}} \sum_{i=0}^{n+l+2} \frac{(\alpha r)^i}{i!} \right] \qquad (7.16)$$

(The numerical fit functions have a similar behaviour).

The second term decays exponentially with the distance and hence its bloch sum can be computed efficiently in a limited loop over lattice points. The first term gives rise to the familiar lattice summation problem of solid state theory: we have to calculate sums like

$$V_{lm}(\boldsymbol{r}) = \sum_{\boldsymbol{R}} \frac{Z_{lm}(\Omega_{\boldsymbol{R}})}{|\boldsymbol{r}-\boldsymbol{R}|^{l+1}} \qquad (7.17)$$

In a three-dimensional crystal the sum diverges for l=0, is conditionally convergent for l=1,2 and it is properly defined for higher l-values. But even then convergence is so slow, that a straightforward summation would be unpractical. We will concentrate on the l=0 lattice sum (for 3D crystals), since that is the most difficult case.

Let a charge q_i be given at position $\boldsymbol{s}_i$ in the unit cell. The potential due to the associated lattice of charges is

$$V(\boldsymbol{r}) = q_i \sum_{\boldsymbol{R}} \frac{1}{|\boldsymbol{r}-\boldsymbol{R}-\boldsymbol{s}_i|} \qquad (7.18)$$

The divergence of this sum is not a fundamental physical problem because the net charge in the unit cell is zero. The addition of the sums (7.18) due to all charges q_i leads to a cancelling of the 'infinities', but of course only when properly combined. Hence the expression for the total Madelung potential

$$V_M(\boldsymbol{r}) = \sum_j q_j \sum_{\boldsymbol{R}} \frac{1}{|\boldsymbol{r}-\boldsymbol{R}-\boldsymbol{s}_j|} \qquad (7.19a)$$

with the charge neutrality condition

$$\sum_j q_j = 0 \tag{7.19b}$$

is *conditionally* convergent. The same holds for the related Madelung *energy* (per unit cell)

$$E_M = \frac{1}{2} \sum_i q_i \; V'_M(\boldsymbol{s}_i) \tag{7.20}$$

where $V'_M(\boldsymbol{s}_i)$ is the Madelung potential at position $\boldsymbol{s}_i$ given by (7.19), but without the singular term ($\boldsymbol{R}=0$, $j=i$):

$$V'_M(\boldsymbol{s}_i) = \sum_{\boldsymbol{R}\neq 0} \sum_j q_j \frac{1}{\hat{0}\boldsymbol{s}_i - \boldsymbol{R} - \boldsymbol{s}_j\hat{0}} + \sum_{j\neq i} q_j \frac{1}{\hat{0}\boldsymbol{s}_i - \boldsymbol{s}_j\hat{0}} \tag{7.21}$$

Conditional convergence implies that we can obtain any answer from the summation, depending on the order in which the terms are taken. Since the Madelung energy is a well defined physical quantity the conditional convergence of the sums (7.19a) and (7.21) is a little puzzling at first sight. This is not a purely mathematical inconvenience here, but it is related to the physical aspect that we cannot have a truly infinite crystal. It is not difficult to show that for any large, but finite crystal the (Madelung) potential in the interior does not only depend on the charge distribution $\{q_i\,,\ \boldsymbol{s}_i\}$ in the unit cell and on the lattice structure $\{\boldsymbol{R}\}$, but also on the boundary of the macroscopic crystal. In the limit of a very large crystal it depends on the average charge density on the surface [Young 1987]. The lattice sum may thus be interpreted as follows: we take a large but finite crystal of a specific shape, evaluate the finite (and hence well defined) sum and let then increase the crystal *size* to infinity preserving the *shape*. The conditional convergence is now translated in that the limiting result depends on the assumed shape of the crystal. The *physically* correct value is obtained only if the shape has been chosen such that the average surface (and bulk) charge density is zero [Young 1987].

The evaluation of lattice sums has attracted much attention in the literature (see Tosi [1964] and Glasser and Zucker [1980] for a discussion; a more recent, but concise survey with references is given by Bhowmick *et al.* [1988]). The numerous proposed methods fall in two classes. The first uses some kind of integral transform to replace the slowly or even conditionally convergent sum by (usually) two series which converge both rapidly. The most famous exponent of this class is Ewald's method [Ewald 1921]. Part of the sum (7.18) is transformed then into a fourier series, i.e. a sum over lattice points in reciprocal space, and the remaining real space part is a sum over rapidly decaying error functions. The above mentioned boundary condition for expanding shapes is in this case translated in the implicit assumption (fourier series) that the potential has the same periodicity as the Bravais lattice; a surface charge density in a macroscopic crystal would invalidate this, also in the interior far from the crystal boundary.

In the second class of methods a direct real space summation is performed. Often some kind of expanding-shapes procedure is utilized. The intuitively attractive method of expanding *spheres* is inappropriate since charge neutrality is not preserved and the average charge density on a spherical surface is not zero. In Evjen's method, using expanding cubes, charge neutrality is imposed as an auxiliary condition [Evjen 1932]. For the NaCl structure this gives indeed the correct value for the Madelung constant. Expanding cubes fail however for the CsCl structure [Evjen 1932, Bhowmick *et al.* 1988] because in that case the average charge density on a crystal boundary plane is not zero.

In the straightforward real-space summation techniques the assumed shape, i.e. the way in which the limit of an infinite crystal is approached, is thus of crucial importance. The appropriateness of a choice depends of course on the crystal and has to be determined separately for every new situation. When this is taken care of, one may have obtained a sequence that converges to the correct limit but only slowly. Various of the well known convergence accelerating methods for sequences may then be applied to improve the efficiency. Fortunately these sequence transforms may effectively also alleviate the mentioned shape dependence, as we will see, and thus they make the (transformed) real space summation much more convenient for practical use [Bhowmick *et al.* 1988]. To understand this we return to the definition of the Madelung potential.

A mathematically satisfactory and robust way to solve the apparent arbitrariness of the Madelung lattice sum is provided by the technique of analytic continuation. The coulomb potential r^{-1} is replaced by r^{-t} with a complex parameter t. The corresponding lattice sum is well defined for $\mathrm{Re}(t)>3$. By analytic continuation its value for other t, and in particular for $t=1$ can be derived; the outcome coincides with the physically correct value and with the value obtained by expanding shapes with the appropriate condition on the surface charge [Young 1987, Borwein *et al.* 1985, 1988, Crandall and Buhler 1987].

The situation bears a close resemblence to other, more familiar sequences with problematic convergence. Consider the power series representation of the logarithm

$$\ln(1+z) = z - \frac{z^2}{2} + \frac{z^3}{3} - \frac{z^4}{4} + \ldots = -\sum_{m=1}^{\infty} \frac{(-z)^m}{m} \tag{7.22}$$

The sum converges only for $|z|\leq 1$, $z\neq -1$ but the l.h.s. is defined for all complex z not on the cut (conventionally: z real and $z\leq -1$). So, even where the sequence of partial sums

$$A_n = -\sum_{m=1}^{n} \frac{(-z)^m}{m} \tag{7.23}$$

does not converge we may assign the value $\ln(1+z)$ to the *infinite* sum, that is, to the limit $\{A_n\}_{n\to\infty}$. Putting it another way: the infinite, non-convergent sum is a correct, though somewhat inconvenient *representation* of $\ln(1+z)$.

Following a discussion by Shanks [1955], many sequences occurring in physics and mathematics can be written as 'transients'

$$A_n = B + \sum_{i=1}^{k} \alpha_i \; q_i^n \qquad (q_i \neq 0,1) \tag{7.24}$$

The constant B is called the *base* and the (complex) ratios q_i and amplitudes α_i constitute the *spectrum* of the sequence. The number of transient terms, k, is the *order* of the sequence. The order may be infinite or even continuous so that the sum is to be replaced by an integral. (Levin [1973] treats a more general form than (7.24), which covers more types of sequences and is in fact more related to our lattice sums [Weniger *et al.* 1986], but the simpler case of Shanks suffices for the discussion here).

Depending on the q_i the sequence may have monotonic and oscillatory, converging and diverging components. If all $|q_i|<1$ the sequence converges to B and hence B is the limit of the sequence. If at least one of the $|q_i|>1$ the sequence diverges and B is then called the *antilimit*: the sequence diverges away from B. In general B is the 'intrinsic part'.
An example. Consider the power series representation of $(1-z)^{-1}$

$$\frac{1}{1-z} = 1 + z + z^2 + z^3 + \ldots \tag{7.25}$$

The partial sum can be written in the transient form (7.24):

$$A_n = \sum_{m=0}^{n} z^m = \frac{1}{1-z} - \left(\frac{1}{1-z}\right) z^n \tag{7.26}$$

so that it turns out to be a first order transient with base $1/(1-z)$ and only one ratio, z, with amplitude $-1/(1-z)$. For $|z|>1$ the sequence A_n diverges, but the infinite sum is the base $1/(1-z)$. This is an example of the general phenomenon that the base of a diverging sequence is the 'correct' value for the 'limit', that is, the intrinsic part equals the analytic continuation of the sums of the *convergent* series [Shanks 1955].
The same applies to our lattice sum. Replacing the coulomb potential r^{-1} by r^{-t} the corresponding lattice sum

$$V(\boldsymbol{r}) = \sum_{\boldsymbol{R}} \frac{1}{|\boldsymbol{r}-\boldsymbol{R}-\boldsymbol{s}_i|^t} \tag{7.27}$$

is a series representation of a generalized multidimensional Riemann ζ-function [Glasser and Zucker 1980, Crandall and Buhler 1987, Borwein *et al.* 1988]. It has poles for t=0 and t=3 (in the 3-dimensional case) and it is properly defined everywhere else in the complex plane. Its representation by the lattice sum yields a sequence (when we take all terms inside a sphere of radius R and let R increase) which converges for

Re(t)>3. For other t, and in particular for the 'coulomb value' t=1 the sequence diverges but the physically correct value is the intrinsic part.

Obviously it is not possible to compute the intrinsic part of a diverging sequence by just monitoring the sequence of partial sums. As noted already: the evaluation of lattice sums by the method of 'expanding spheres' fails. It would be desirable therefore to transform the diverging transient part into a converging sequence (without affecting the intrinsic part of course). Two closely related methods are often applied in such cases: 'screening' and 'sequence transformations'. We will shortly discuss both and indicate the connection between them, because BAND employs a screening form which is heuristically motivated by a particular type of sequence transformations.

screening

Screening functions serve to suppress long-range tails of functions (such as the coulomb potential here) that cause problems by the slowness of their decay. A screening function which is often used because of its mathematical simplicity is the exponential function. Replace the coulomb potential $1/r$ by $e^{-\alpha r}/r$ with real and positive α; finally we will then take the limit $\alpha \downarrow 0$.
Although for finite α the lattice sum due to one single lattice of charges q_i

$$V_i^{\alpha}(\boldsymbol{r}) = q_i \sum_{\boldsymbol{R}} \frac{e^{-\alpha r}}{\hat{\mathrm{o}}\,\boldsymbol{r}-\boldsymbol{R}-\boldsymbol{s}_i\hat{\mathrm{o}}} \tag{7.28}$$

is defined, its value diverges smoothly as we diminish α to zero. If we apply the screening however to the total Madelung potential due to the combination of all charges, which is in fact the quantity of interest,

$$V^{\alpha}(\boldsymbol{r}) = \sum_i q_i \sum_{\boldsymbol{R}} \frac{e^{-\alpha r}}{\hat{\mathrm{o}}\,\boldsymbol{r}-\boldsymbol{R}-\boldsymbol{s}_i\hat{\mathrm{o}}} \tag{7.29}$$

then the result stays bounded and converges to the correct value in the limit $\alpha \downarrow 0$. This is caused by the 'cancelling of infinities' from the compensating charges q_i . Compare the 1D analogues

$$\lim_{\alpha \downarrow 0} \sum_{n=0}^{\infty} e^{-\alpha n} = \infty \tag{7.30a}$$

and

$$\lim_{\alpha \downarrow 0} \sum_{n=0}^{\infty} (-)^n \, e^{-\alpha n} = 1/2 \tag{7.30b}$$

This confirms what may be intuitively obvious: screening may work for conditionally converging sums, but it fails for diverging sums.

The coulomb screening factor $e^{-\alpha r}$ has been used for instance by Born [1921] in an analysis of the electrostatic energy of cubic lattices. In BAND we employ a somewhat different screening function, namely a fermi-dirac function $h(r)$

$$r^{-1} \qquad \frac{r^{-1}}{1+e^{(r-r_0)/d}} \equiv h(r)\, r^{-1} \tag{7.31}$$

Like $e^{-\alpha r}$ the function $h(r)$ decays exponentially for large r. If we choose r_0 large enough however the nearest terms in the lattice sum are only negligibly affected by the screening function $h(r)$ and this turns out to be an advantage in the convergence behaviour. The limiting case is obtained by taking $d \to \infty$ and simultaneously $r_0 / d \to \infty$, so that $h(r)$ converges uniformly to unity for all r as the limit is approached. The limit is not actually reached in the computational application; instead the calculations are performed with a finite screening; the results are therefore only (good) approximations of the limiting values.
The motive for this form of screening function originates from a particular type of sequence transformations.

sequence transformations

A large number of transformations are known that accelerate the convergence for various types of sequences [Brezinski 1977]. Among the most widely used are the ε-transforms of Shanks [Shanks 1955] of which Aitken's δ^2 -procedure [Aitken 1926] is a special case, Levin's u-transformation [Levin 1973], the Padé approximations [Sarkar and Bhattacharyya 1988] and Euler's transformation. The latter will have our special attention and is treated in detail by Hardy [1949].

Sequence to sequence transformations are not only used to speed up convergence but also to induce absolute convergence in diverging or conditionally converging sequences [Shanks 1955, Weniger *et al.* 1986, Bhowmick *et al.* 1988]. The motive to do so is of course that we are interested in the intrinsic part, or the base in Shank's terminology, of an ill-converging sequence. If the transformation leaves the base unaffected the transformation is justified and may help us to handle the diverging extrinsic part by transforming it into a *converging* sequence.

That sequence transformations are capable to transform diverging into converging sequences (with the correct limit) may be illustrated by applying Shank's ε_2 -transform (i.e. Aitken's method) to the sequence (7.26)

$$A_n^{\varepsilon_2} \equiv \frac{A_{n-1}A_{n+1} - (A_n)^2}{A_{n-1} + A_{n+1} - 2A_n} = \frac{\left(\frac{1}{1-z}\right)^2 \{ (1-z^{n-1})(1-z^{n+1}) - (1-z^n)^2 \}}{\left(\frac{1}{1-z}\right) \{ (1-z^{n-1}) + (1-z^{n+1}) - 2(1-z^n) \}} = \frac{1}{1-z} \tag{7.32}$$

This transformation turns thus out to be extremely well suited for the sequence at hand. In other cases the transformed sequence may of course not be converged immediately, as it is here, but at least it may be much

more pleasant than the original one. See Shanks [1955] and Weniger *et al.* [1986] for a more general discussion of this subject.

A transformation of interest for us is Euler's transformation, defined by simple averaging

$$A_n \qquad A_n^1 = \frac{A_n + A_{n-1}}{2} \tag{7.33}$$

Euler's transformation is well suited to accelerate convergence in summations of alternating terms. The reason is easily understood: due to the alternation in the terms the sequence of partial sums $\{A_n\}$ oscillates and averaging two consecutive values will 'damp' the oscillations.
One may repeat the proces by applying the transformation also to the transformed sequence. In this case we obtain the second-order Euler-transformed sequence

$$A_n^2 = \frac{A_n^1 + A_{n-1}^1}{2} = \frac{A_n + 2A_{n-1} + A_{n-2}}{4} \tag{7.34}$$

and in general

$$A_n^m = \frac{1}{2^m}\left\{\sum_{k=0}^{m} \binom{m}{k} A_{n-k}\right\} \qquad n=m,m+1,\ldots \tag{7.35}$$

The effect of this simple transformation and its iterates is exemplified by the following. Let the terms be

$$a_k = \frac{(-)^k}{k+1} = \{1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \ldots\} \tag{7.36a}$$

The partial sums and the transformed sequences are then

$$\begin{aligned}
A_n^0 &= \{\ 1.0000, \quad 0.5000, \quad 0.8333, \quad 0.5833, \quad 0.7833, \quad 0.6167,\ldots\} \\
A_n^1 &= \{\qquad\quad 0.7500, \quad 0.6667, \quad 0.7083, \quad 0.6833, \quad 0.7000,\ldots\} \\
A_n^2 &= \{\qquad\qquad\quad 0.7083, \quad 0.6875, \quad 0.6958, \quad 0.6917,\ldots\} \\
A_n^3 &= \{\qquad\qquad\qquad\quad 0.6979, \quad 0.6917, \quad 0.6938,\ldots\}
\end{aligned} \tag{7.36b}$$

The exact limit is log(2)=0.693147... and each successive transformation improves the convergence considerably.

If N+1 terms a_k are known, the transformation can be repeated at most N times to yield the number A_N^N; the expression 'Euler transformation' applies in fact to this ultimate value. For convenience in the discussion we use the phrase '*repeated* (Euler) transformation' and we denote the sequence A_N^N, N=0,1,.. as the *diagonal* (Euler) transformation. Assuming that each successive Euler transformation yields a faster convergence, the diagonal sequence represents the best approximation.

An attractive aspect of Euler's transform is that it is a *linear* transformation. Whereas non-linear transforms, such as the ε-transforms, may in comparison be spectacularly more effective for some sequences, they do not give the *correct* limit for *all* sequences. Linear transformations are more reliable in this respect [Hardy 1949, Shanks 1955, Weniger *et al.* 1986]. So one may be willing to accept a somewhat reduced efficiency as an insurance premium for correctness of the results.

historical note

Already before the analysis of divergent sequences was given a solid mathematical foundation [Stieltjes 1886, Poincaré 1886] Euler assumed and used in his work that 'summa cujusque seriei est valor expressionis illius finitae, ex cujus evolutione illa series oritur'[1] . Whereas this is strictly speaking not true, in that distinct expressions with different numerical values may yield the same diverging series, it is essentially correct, with a somewhat stricter definition and his 'repeated averaging' procedure is a very natural method for the numerical evaluation of the sum of such (alternating) non-convergent series [Hardy 1949].

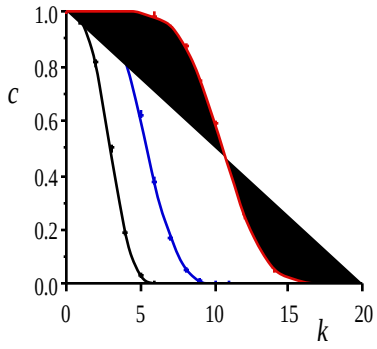

Fig.1. Coefficients c_k^N in the diagonal Euler sequences, with interpolating curve, for N=5,10 and 20.

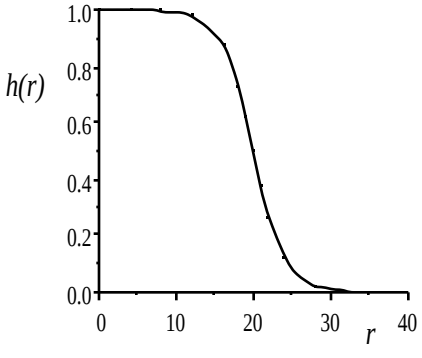

Fig.2. The fermi dirac screening function $h(r)=\{1+\exp((r-r_0)/d)\}^{-1}$; the parameters are here arbitrarily taken as: r_0 =20, d=2.

The contribution of the primitive terms a_k in the Euler sequences can be expressed as sums of binomial coefficients. Let

[1] the sum of the series is the finite numerical value of the arithmetic expression from which the series is derived [Hardy 1949]

$$A_n^m = \sum_{k=0}^{n} c_k^{(mn)}\, a_k \qquad (7.37)$$

All terms $k \le n-m$ are fully present ($c_k = 1$), the next coefficients decrease and $c_k = 0$ for all $k > n$. The fractional coefficients are, from (7.35)

$$c_k^{(mn)} = \frac{1}{2^m} \sum_{i=0}^{n-k} \binom{m}{i} \qquad k=n-m,\ n-m+1\ldots..,n \qquad (7.38)$$

The coefficients in the diagonal sequence are obtained by putting $m=n\equiv N$:

$$c_k^N = \frac{1}{2^N} \sum_{i=0}^{N-k} \binom{N}{i} \qquad k=0,1,...,N \qquad N=0,1,... \qquad (7.39)$$

Fig.1 displays these coefficients, with an interpolating curve, for a few values of N.

As shown by the interpolating curves, the coefficients may also be described by continuous functions $c^N(x)$ which reproduce the values (7.39) for the integer arguments x=0,1,2,..., or more generally for a set of equidistant points x_i , i=1,2,.... This suggests already the correspondence between screening and sequence transformations: we might have started with the (screening) function $c(x)$; then, if the members of the original sequence A_n , n=0,1,... correspond to equidistant values of x, we have effectively performed a sequence transformation by application of the screening.

A second aspect suggested by the curves in fig.1 is that the functions $c^N(x)$, N=0,1,2,.. can also be interpolated in the variable N. We may thus define a general function $c(p;x)$ which coincides with $c^N(x)$ for $p=N$ and interpolates between these curves for non-integer p. Now we have arrived at a general screening with a continuous screening parameter p, similar to the traditional exponential screening parameter α as in (7.29).

The application of the (diagonal) Euler transformation to the lattice summation poses some problems. First: how do we define the zero-th order sequence to be transformed? It is natural and intuitively attractive to order the contributions according to their distance from the evaluation point $\boldsymbol{r}$, and to group together all terms with the same distance: an 'expanding spheres' picture. For the 'special' evaluation points $\boldsymbol{s}_i$, where the charges q_i in the unit cell are located, we obtain then 'stars' of terms. For a general off-center evaluation point $\boldsymbol{r}$ these stars are scattered and fall apart in distinct terms. In all cases the discrete distance-values for which we find contributions are not equidistant and, although on the average the terms will be alternating, this may not be true on the smallest scale and their sizes may be rather irregular. So, whereas we have the general structure of an alternating sequence, on closer inspection the development of the sequence is far from smooth and regular. This makes it a little unclear how to apply exactly the Euler transformations.

Secondly, and this is possibly even more problematic from a practical point of view, the grouping of the terms has to be determined anew for every evaluation point, because the sequence looks different, with possibly even a different *number* of terms inside a sphere with fixed radius, and the combination coefficients (7.39) will have to be computed many times. Considering the large number of evaluation points (all points of the crystal integration grid) this is computationally unwieldy.

Moreover the definition of the zero-th order sequence is to some extent arbitrary. We may for instance introduce formally a very dense radial mesh with equidistant points R_i , i=0,1,... and define the sequence as the potential due to all terms inside the corresponding spheres. Of course, in the limit of a continuous mesh we will find large subsequences of constant values in the sequence and sudden jumps at a few radial values. The limiting long-range development of the sequence is not essentially changed by this however and we may still expect that screening, or equivalently a suitable sequence transformation, can be applied to remove the divergence and to extract the desired intrinsic part of the sequence. We will use this approach (i.e. conceptually introducing the dense and equidistant radial mesh) and use thus a continous screening function.

The next problem is then how to define that continuous function. Ideally we would like to reproduce the Euler transformation by it, but we have not succeeded in finding an analytical form $c(x)$ to compute efficiently the binomial expressions for equidistant arguments x_i . Fortunately the precise form of the screening function is not essential for our final purpose, the calculation of the intrinsic part of the sequence. Only we expect that, the more it resembles the 'Euler screening', the more effective it will be as regards the convergence with ever smoother and 'weaker' screening. The fermi-dirac function which we employ in BAND resembles the Euler function reasonably well and has shown to give satisfactory results; fig.2 displays this function $h(r)$ (7.31) for a comparison with fig.1.

remarks

We have treated screening and sequence transformations as essentially different and only accidentally similar. In fact both can be analyzed more rigorously as methods to 'sum' certain types of non-convergent series. The screening as presented here is a special case of the Abelian method. In the chemical and physical literature the phrase 'screening' is often used, which is the reason to denote it here as such.
The interpolation in the variable N (7.39), i.e. the definition of a continuous-order Euler transformation can also be given a strict meaning by a generalization of (7.39).
These and many more fascinating aspects of the summation of non-convergent series are discussed extensively by Hardy [1949] and the reader is referred to his work for mathematical rigor and more background information. The intention of our presentation here has been to explain in an intuitively appealing way (we hope) why and how our screening method works.

the screening function h(r)

Finally we have to determine the two fermi-dirac screening parameters r_0 and d. Sticking to the analogy with the Euler transformations we may say that d corresponds to the *order* of the transformation (the parameter m in (7.35)), but now as a continuous parameter, and r_0 is similarly the member-index n of the (transformed) sequence. The *diagonal* transformation corresponds to tuning d and r_0 relative to each other in

such a way that $h(r)$ starts to fall off from unity already at small r (compare (7.38) and (7.39): *all* coefficients in the diagonal transform are fractional). The diagonal *sequence*, that is, increasing N in (7.39) finally corresponds to increasing d and simultaneously adapting r_0 in the manner just discussed.
From our test calculations we learned the following:

The 'order of the transformation' d should at least be of the same size as the nearest neighbour distance; otherwise the oscillations in the sequence, i.e. the oscillations as a function of r_0, are too large to make the results reliable.
The complete interval over which $h(r)$ changes should be used; in particular all terms at large r must be taken into account until $h(r)$ has become truly negligible; neglect of this condition leads quickly to unacceptable errors.
Since the actual summation over lattice points must be limited as much as possible for computational reasons, the extension of $h(r)$ has to be restricted, so that d and hence r_0 cannot be chosen arbitrarily large. A good practical compromise is expressed by the following conditions:
   a) $d \approx 0.65 \times D_{NN}$
   b) $r_0 \approx 13 \times d$

where D_{NN} is the nearest neighbour distance in the crystal.
In the implementation these figures are adapted to the general (integration) accuracy parameter accint (see SS^integration); higher accuracy induces larger values for d and r_0.

The association of the screening function $h(r)$ with repeated Euler transformations is confirmed by the test results for the Madelung energy. Let $E(r_0, d)$ be the computed Madelung energy as a function of the screening parameters. Then, for fixed d, E oscillates as a function of r_0. The amplitude of these oscillations depends on d and diminishes if we increase d. The energy values oscillate around the limit, provided r_0 is large enough. This corresponds to the general phenomenon in Euler transformations that each of the series oscillates around the limit and the higer order transformations display smaller oscillations.

integration of the oscillations
Except for very small d, the oscillations as a function of r_0 are found to be fairly regular over a considerable range in r_0, so that $E(r_0)$ is reasonably well described by a simple cosine function around the limit. The periodicity of the cosine depends on the crystal structure, is in the order of the nearest neighbour distance (slightly larger), but we have not been able to detect a strict relation.

Nevertheless the regularity of the oscillation suggests that we may determine the limit more precisely by averaging the results for a sequence of different values r_0 distributed over a cycle of the oscillation. We do so in BAND by taking five equidistant points; this corresponds to a numerical integration for periodic functions if the spacing between the points is one fifth of the period. BAND assumes that the period equals $1.15 \times D_{NN}$; this will usually not be exact and consequently the 'integration' is not optimal. It should be noted however that some improvement results anyway from the averaging.

remark

It would be possible to set up a standard pilot computation in BAND to determine a priori the periodicity of the oscillation in the Madelung potential as a function of the parameter r_0 . This could then be used to replace the assumed periodicity, thereby raising the efficiency of the averaging. At first sight this does not seem very important because the results are fairly accurate anyway. However with an improved averaging the conditions on the parameters r_0 and d may be weakened, both can be reduced and this leads to a smaller lattice summation loop: increased efficiency. It may be noticed here that in 3-D crystals the construction of the multipole potentials, though not a bottleneck, takes a significant amount of time (a few percents of the total calculation).

implementation

VMULTI computes the multipole potentials

$$V_{lm}^{\alpha}(\boldsymbol{r}) = \sum_{\boldsymbol{R}} \frac{Z_{lm}(\Omega_{\boldsymbol{R}+\boldsymbol{s}_\alpha})}{|\boldsymbol{R}+\boldsymbol{s}_\alpha-\boldsymbol{r}|^{l+1}} \; h(|\boldsymbol{R}+\boldsymbol{s}_\alpha-\boldsymbol{r}|) \tag{7.40}$$

for all l-values occurring in the fit set. $\boldsymbol{s}_\alpha$ is the position of an atom in the unit cell; $h(r)$ is the screening function.

The parameters r_0 and d are in BAND the variables rmadel and dmadel. One may specify them in input with the keys `r madelung` and `d madelung`. By default they are derived (in RADIAL) from the nearest neighbour distance dneigh (computed by NEIGHB) and the accuracy parameter accint:

$$\text{dmadel} = (0.50 + 0.05\times\text{accint}) \times \text{dneigh} \tag{7.41a}$$

$$\text{rmadel} = (10+\text{accint}) \times \text{dmadel} \tag{7.41b}$$

The loop over the cells terminates when the tail of $h(r)$ has become negligible. Experience shows that cutting off the tail of $h(r)$ at large r too quickly leads to unacceptable errors. The maximum cell distance, rcelx is set at

$$\text{rcelx} = \text{rmadel} + (11+\text{accint}) \times \text{dmadel} \tag{7.42c}$$

The default for rcelx may be overruled via input (key `cell distance`).

If one of the variables dmadel, rmadel and rcelx is specified via input, the default determination of the others is adapted to satisfy as well as possible all conditions discussed here (routine RADIAL).

modifications of the screened lattice sums

1. The evaluation of (7.29) in a point $\boldsymbol{r}$ takes place in a simple loop over lattice points; no interpolation is needed, in contrast with the treatment of other one-center functions in BAND. The 'exact' evaluation, without interpolation, gives rise to an awkward problem in the core regions. The potential of an l=0 fit function has the long range behaviour $1/r$, but for small r it goes to zero: the $1/r$ multipole term is cancelled by another term, as can be verified by inspection of (7.13). FITPNT constructs the fit potential, by addition of the multipole lattice sum (from VMULTI) at one hand and the 'normal' bloch sum of the exponentially

decaying part at the other hand. The second part is obtained by interpolation from one-center function tables and contains thus small inaccuracies which are not present in the multipole part. Consequently the cancelling for $r \downarrow 0$ is not exact and for very small r these numerical errors blow up.
Therefore the multipole term is modified such that for small r it is suppressed: that part is incorporated in the other term, i.e. in the one-center function table (in routine FITRA2). The modified multipole potential becomes

$$M_{lm}(r) \qquad Z_{lm}(\Omega)\,\frac{r^2}{r^{l+3}+e^{-r}} \tag{7.43}$$

For large r this equals the pure multipole potential $\sim 1/r^{l+1}$. For $r \downarrow 0$ it behaves as r^2 . Consequently the multipole lattice potentials as computed in VMULTI have 'core-holes'. This is compensated in BAND in the one-center tables (FITRA2), but it must be kept in mind when the results of VMULTI are to be applied in another situation.

	Rock-salt	*Cesium Chloride*	*Zincblende*	*Fluorite*
A=2	1.747 51 41 ..	1.017 65 41 ..	1.637 76 12 ..	2.519 08 40 ..
A=4	1.747 56 19 ..	1.017 67 95 ..	1.638 06 34 ..	2.519 40 07 ..
exact	1.747 56 45 9.	1.017 68 07 5.	1.638 05 ..	2.519 39 ..

Table I. Computed Madelung constants for different accuracies A. The exact values have been taken from Sarkar and Bhattacharyya [1988] (rock-salt and cesiumchloride) and Atkins [1987] (zincblende and fluorite).

2. The numerical integration (over the parameter r_0) of the lattice sums, as discussed above, is carried out by a modification of $h(r)$. Let the spacing between the equidistant integration points be δ and the five points be $r_0 - 2\delta$, $r_0 - \delta$, r_0 , $r_0 + \delta$, $r_0 + 2\delta$. Then the multipole potential is computed as (we leave the first modification aside for the moment)

$$V_{lm}(\boldsymbol{r}) = \frac{1}{5}\sum_{k=-2}^{2}\left\{\sum_{\boldsymbol{R}}\frac{Z_{lm}(\Omega_{\boldsymbol{R}})}{|\boldsymbol{R}+\boldsymbol{r}|^{l+1}}\,\frac{1}{1+e^{(|\boldsymbol{R}-\boldsymbol{r}|-(r_0+k\delta))/d}}\right\} =$$

$$= \sum_{\boldsymbol{R}}\left\{\frac{Z_{lm}(\Omega_{\boldsymbol{R}})}{|\boldsymbol{R}+\boldsymbol{r}|^{l+1}} \times \frac{1}{5}\sum_{k=-2}^{2}\frac{1}{1+f^k\, e^{(|\boldsymbol{R}-\boldsymbol{r}|-r_0)/d}}\right\} \tag{7.44}$$

with $f = e^{-\delta/d}$. f is the variable fermfc in BAND; fermfc is computed in RADIAL. The screening function actually employed (FITPNT, ATMFNC, VMULTI) is thus

$$h(r) = \frac{1}{5} \sum_{k=-2}^{2} \frac{1}{1+f^k \; e^{(r-r_0)/d}} \tag{7.45}$$

In table I we display the computed Madelung constants of a few standard crystal types, for different values of the accuracy parameter.

potential and density, another relation

The following equation connects the integral over the potential with an integral over the density

$$\int V(\boldsymbol{r})\, d\boldsymbol{r} = \frac{-2\pi}{3} \int r^2 \;\; \rho(\boldsymbol{r})\, d\boldsymbol{r} \tag{7.46}$$

In a crystal the region of integration is an 'element of periodicity', such as the crystal unit cell. When divided by the cell volume (7.46) gives the average potential, sometimes called the 'constant term (in the potential)'. A discussion of the average potential and/or relation (7.46) turns up from time to time in the literature, without bearing much relevance, since the zero level of the potential in a 3D crystal is physically arbitrary.
The reason to mention it here is not that we found an interesting application, but rather that in the derivation of (7.46) I overlooked in first instance an instructive mathematical aspect, which is fairly general and which may easily be missed.

First we examine the relation in a finite system, where the integrations extend over all space.
The potential due to a charge distribution $\rho(\boldsymbol{r})$ is defined by

$$V(\boldsymbol{r}) = \int \frac{\rho(\boldsymbol{r})d\boldsymbol{r}'}{\hat{o}\, \boldsymbol{r}-\boldsymbol{r}'\hat{o}} \tag{7.47}$$

alternating sequences again

We notice first that (7.46) does not hold for an arbitrary charge distribution. Two conditions (at least) must be fulfilled: the monopole and dipole moments of $\rho(\boldsymbol{r})$ have to vanish both. The first condition is intuitively obvious since otherwise the integral over the potential would be unbounded. Both requirements are clearly exposed when we choose another coordinate frame by displacing the origin, i.e. set $\boldsymbol{r} \rightarrow \boldsymbol{r}+\boldsymbol{r}_0$. The r.h.s. of (7.46) becomes (apart from the factor $-2\pi/3$)

$$\int \hat{o}\, \boldsymbol{r}+\boldsymbol{r}_0 \; \hat{o}^2 \;\; \rho(\boldsymbol{r})\, d\boldsymbol{r} = \int r^2 \;\; \rho(\boldsymbol{r})\, d\boldsymbol{r} + r_0^2 \int \rho(\boldsymbol{r})\, d\boldsymbol{r} + 2\boldsymbol{r}_0 \cdot \int \boldsymbol{r}\, \rho(\boldsymbol{r})\, d\boldsymbol{r} =$$

$$= \int r^2 \;\; \rho(\boldsymbol{r})\, d\boldsymbol{r} + r_0^2 \;\; Q + 2\boldsymbol{r}_0 \cdot \boldsymbol{D} \tag{7.48}$$

giving the original result plus a monopole and a dipole term; Q is the total amount of charge, i.e. the monopole moment of $\rho(\boldsymbol{r})$, $\boldsymbol{D}$ is the dipole moment. Since the l.h.s. of (7.46) is not affected by the diplacement of the origin, the integral over the potential is apparently not well defined unless both Q and $\boldsymbol{D}$ are zero.

Now we will give a derivation of (7.46) in which the dipole condition is erronously juggled away; so we assume only a vanishing monopole moment and proceed to 'prove' (7.46). Consider a finite system, a molecule or cluster, such that all charge is confined within a huge but finite sphere with radius R. The potential outside the sphere has an expansion in spherical harmonics

$$V(\boldsymbol{r}) = \sum_{lm} V_{lm}(r)\ Z_{lm}(\Omega) \qquad |\boldsymbol{r}| > R \tag{7.49}$$

where the components $V_{lm}(r)$ are related to the multipole moments of the charge distribution. For a neutral system the monopole term is zero. Hence, due to the angular integration of $Z_{lm}(\Omega)$, $l \geq 1$

$$\int_{\substack{outside\\ sphere}} V(\boldsymbol{r})\, d\boldsymbol{r} = 0 \tag{7.50}$$

So for the potential (as well as for the density) the integration over all space in (7.46) can be restricted to the integral inside the sphere.

To evaluate the integral of the potential (the l.h.s. of 7.46) we use a formal expression for the density analogous to (7.49), and the expansion of $|\boldsymbol{r}-\boldsymbol{r}'|^{-1}$ in spherical harmonics

$$\frac{1}{|\boldsymbol{r}-\boldsymbol{r}'|} = \sum_{lm} \frac{4\pi}{2l+1} \frac{r_<^l}{r_>^{l+1}}\ Z^*_{lm}(\Omega')\ Z_{lm}(\Omega) \tag{7.51}$$

$r_<$ and $r_>$ refer to the smaller and larger respectively of $|\boldsymbol{r}|$ and $|\boldsymbol{r}'|$. This gives

$$\int_{\substack{all\\ space}} V(\boldsymbol{r})\, d\boldsymbol{r} = \int_0^R r^2\, dr \int d\Omega \int \rho(\boldsymbol{r}')\, d\boldsymbol{r}' \sum_{l'm'} \frac{4\pi}{2l'+1} \frac{r_<^{l'}}{r_>^{l'+1}}\ Z^*_{l'm'}(\Omega')\ Z_{l'm'}(\Omega) =$$

$$= \int_0^R r^2\, dr \int d\Omega \int_0^R (r')^2\, dr' \int d\Omega' \sum_{lm} \rho_{lm}(r')\, Z_{lm}(\Omega') \sum_{l'm'} \frac{4\pi}{2l'+1} \frac{r_<^{l'}}{r_>^{l'+1}}\ Z^*_{l'm'}(\Omega')$$

$$Z_{l'm'}(\Omega) =$$

$$= \int_0^R r^2\, dr \int_0^R (r')^2\, dr' \sum_{lm} \rho_{lm}(r') \frac{4\pi}{2l+1} \frac{r_<^{l}}{r_>^{l+1}} \int d\Omega\ Z_{lm}(\Omega) =$$

$$= 4\pi \int_0^R (r')^2\, dr'\, \rho_{00}(r') \int_0^R r^2\, dr\, \frac{(4\pi)^{1/2}}{r_>} = (4\pi)^{3/2} \int_0^R (r')^2\, dr'\, \rho_{00}(r') \left(\frac{R^2}{2} - \frac{(r')^2}{6} \right) =$$

$$= 2\pi R^2 \int_0^R r^2\, dr\, \rho_{00}(r) \int d\Omega\, Z_{00}(\Omega) - \frac{2\pi}{3} \int_0^R r^4\, dr\, \rho_{00}(r) \int d\Omega\, Z_{00}(\Omega) =$$

$$= \sum_{lm} \left[2\pi R^2 \int_0^R r^2\, dr\, \rho_{lm}(r) \int d\Omega\, Z_{lm}(\Omega) - \frac{2\pi}{3} \int_0^R r^4\, dr\, \rho_{lm}(r) \int d\Omega\, Z_{lm}(\Omega) \right] =$$

$$= 2\pi R^2 \int \rho(\boldsymbol{r})\, d\boldsymbol{r} - \frac{2\pi}{3} \int r^2\, \rho(\boldsymbol{r})\, d\boldsymbol{r} = -\frac{2\pi}{3} \int r^2\, \rho(\boldsymbol{r})\, d\boldsymbol{r} \qquad (7.52)$$

The charge neutrality of the system has been used in the last line of (7.52) and in (7.50), but what happened to the *dipole* condition?

The clue is the dipole term in equation (7.50). A dipole moment $\boldsymbol{D}$ gives a potential at a position $\boldsymbol{r}$ relative to the dipole

$$V_{dipole}(\boldsymbol{r}) = \frac{D\, \cos\theta}{r^2} \qquad (7.53)$$

where θ is the angle between $\boldsymbol{D}$ and $\boldsymbol{r}$. The integral of this potential over the region outside the sphere is

$$\int_{\substack{outside \\ sphere}} V_{dipole}(\boldsymbol{r})\, d\boldsymbol{r} = D \times \left(\int_R^\infty \frac{dr}{r} \right) \times \left(\int d\Omega\, \cos\theta \right) \qquad (7.54)$$

This was taken to be zero (7.50) because of the last factor, but the unboundedness of the radial integral makes such an assessment hazardous: infinity×zero=??.

Integral (7.54) may be associated with the summation of an alternating sequence, of which the terms do not go to zero. If we start at the sphere boundary and increase R by small discrete steps, each shell may be thought to contribute two terms to the integral (7.54), one positive and one negative, corresponding to the half shells ($0 \le \theta \le \pi/2$) and ($\pi/2 \le \theta \le \pi$). They cancel each other, but the individual terms do not go to zero as R increases.

The 'outside' integral (7.54) resembles thus a summation like $S = 1-1+1-1+1-\ldots$. Putting it another way: assume that the dipole moment is oriented along the z-axis and imagine a displacement of the sphere in the positive z-direction (a different choice of origin): as regards the integral over the dipole potential a *positive* half-shell is drawn into the sphere from the outside region and a similar *negative* half shell is expelled. The integral inside the sphere is thus changed by a term which is not negligible however large we took R. An equal, but opposite change has to take place in the 'outside' integral if the integral over all space (7.46) is

properly defined. Relation (7.50) cannot be true then in both situations, although the monopole condition, presumably the only requirement for (7.50), may be satisfied.
Conclusion: the integral of the dipole potential over all space, and in particular over the outside region, is *not* well defined. In analogy with a discrete series we may call (7.50) *conditionally* defined.

average potential in a crystal

In a 3D crystal the charge and potential are periodic functions. Divide $\rho(\boldsymbol{r})$ in localized terms $\rho_i(\boldsymbol{r})$

$$\rho(\boldsymbol{r}) = \sum \rho_i(\boldsymbol{r}) \tag{7.55}$$

such that the $\rho_i(\boldsymbol{r})$ transform into each other under the translation operators of the crystal. This does not uniquely define $\rho_i(\boldsymbol{r})$. We might take for example $\rho_i(\boldsymbol{r})$ equal to the total charge $\rho(\boldsymbol{r})$ in cell i, and $\rho_i(\boldsymbol{r})$ =0 otherwise; that is we may view $\rho(\boldsymbol{r})$ as being build up from 'cells'. We may however also choose $\rho_i(\boldsymbol{r})$ as an atomic-like distribution so that $\rho(\boldsymbol{r})$ is described as a 'sum-of-overlapping-atoms'. We demand only that $\rho_i(\boldsymbol{r})$ is of finite extension so that it can be contained in a finite sphere. Furthermore we require of course that its monopole and dipole moments are zero (i.e. the following remarks do not apply if these conditions cannot be met). Then, following the lines of the derivation (7.40), it is easily found that the average potential is

$$\frac{1}{\Omega} \int_{\substack{unit\\cell}} V(\boldsymbol{r})\, d\boldsymbol{r} = \frac{-2\pi}{3\Omega} \int_{\substack{all\\space}} r^2\ \rho_i(\boldsymbol{r})\ d\boldsymbol{r} \tag{7.56}$$

where Ω is the volume of the unit cell.

A strange aspect of this result is that a different choice of the localized 'element' $\rho_i(\boldsymbol{r})$, e.g. a 'cell' instead of an atomic-like distribution, may yield a different value for the average potential (we assume of course that for all choices of $\rho_i(\boldsymbol{r})$ the dipole (and monopole) moments are zero). Indeed, in a truly infinite 3D crystal the average potential has no physical meaning (we cannot take an electron out of the system) and mathematically it is not defined.
For this reason BAND prints in a 3D calculation the final one-electron energies not as absolute numbers, but gives their values relative to the fermi energy.

In a very huge, but finite system, we may also define elements $\rho_i(\boldsymbol{r})$ to build up the system, but the boundaries of the system now determine the $\rho_i(\boldsymbol{r})$ and the result (7.56) is well defined and gives the average potential in the interior.

8 DOS: density of states and population analysis

DOS is the master routine for the analysis of the eigenstates and energy bands after the SCF procedure has been finished. A Mulliken population analysis is performed and optionally the density of states (DOS) is computed. The evaluation of the necessary integrals over the Brillouin Zone is discussed in SS/BZ-integration and will not be repeated here.

Apart from the total density of states various *partial* densities of states may be computed. They are analogous to the Mulliken populations [Mulliken 1955]. We write the total DOS as

$$n(E) = \sum_{ij} q_{ij}(E) \tag{8.1}$$

where i and j run over the primitive basis functions: atomic one center functions and/or plane waves in the current implementation; one might of course define other basic quantities such as fragment orbitals. An equation for $q_{ij}(E)$ is derived from the usual expression for the DOS:

$$n(E) = \sum_{n} \int_{BZ} \mathrm{d}\boldsymbol{k}\ \delta(E - e_n(\boldsymbol{k})) \tag{8.2}$$

We insert the identity (normalization of the one-particle states)

$$1 = \int \left| \psi_n(\boldsymbol{k};\boldsymbol{r}) \right|^2 \mathrm{d}\boldsymbol{r} \tag{8.3}$$

and use the expansion of $\psi_n(\boldsymbol{k};\boldsymbol{r})$ in the basis functions

$$\psi_n(\boldsymbol{k};\boldsymbol{r}) = \sum_{i} c_{ni}(\boldsymbol{k})\ \phi_i(\boldsymbol{k};\boldsymbol{r}) \tag{8.4}$$

to write for (8.2)

$$n(E) = \sum_{ijn} \iint \mathrm{d}\boldsymbol{k}\ \mathrm{d}\boldsymbol{r}\ \delta(E - e_n(\boldsymbol{k}))\ c^{*}_{ni}(\boldsymbol{k})\ c_{nj}(\boldsymbol{k})\ \phi^{*}_{ni}(\boldsymbol{k};\boldsymbol{r})\ \phi_{nj}(\boldsymbol{k};\boldsymbol{r}) =$$

$$= \sum_{ijn} \int \mathrm{d}\boldsymbol{k}\ \delta(E - e_n(\boldsymbol{k}))\ c^{*}_{ni}(\boldsymbol{k})\ c_{nj}(\boldsymbol{k})\ S_{ij}(\boldsymbol{k}) \tag{8.5}$$

so that

$$q_{ij}(E) = \sum_n \int \mathrm{d}\boldsymbol{k}\ \delta(E-e_n(\boldsymbol{k}))\ c^*_{ni}(\boldsymbol{k})\ c_{nj}(\boldsymbol{k})\ S_{ij}(\boldsymbol{k}) \tag{8.6}$$

The (surface) BZ integral is evaluated numerically:

$$\int \mathrm{d}\boldsymbol{k}\ \delta(E-e_n(\boldsymbol{k})) \quad \sum_{\boldsymbol{k}}\ w_n(\boldsymbol{k}) \tag{8.7}$$

The weights $w_n(\boldsymbol{k})$ depend on the energy E, giving

$$q_{ij}(E) = \sum_n\ w_n(\boldsymbol{k};E)\ c^*_{ni}(\boldsymbol{k})\ c_{nj}(\boldsymbol{k})\ S_{ij}(\boldsymbol{k}) \tag{8.8}$$

The off-diagonal *overlap density of states* is then

$$n_{ij}(E) \equiv q_{ij}(E) + q_{ji}(E) = \sum_{n\boldsymbol{k}}\ w_n(\boldsymbol{k};E)\ \left[c^*_{ni}(\boldsymbol{k})\ c_{nj}(\boldsymbol{k})\ S_{ij}(\boldsymbol{k}) + c^*_{nj}(\boldsymbol{k})\ c_{ni}(\boldsymbol{k})\ S_{ji}(\boldsymbol{k})\right] =$$

$$= \sum_{n\boldsymbol{k}}\ w_n(\boldsymbol{k};E)\ 2\,\mathrm{Re}\left\{c^*_{ni}(\boldsymbol{k})\ c_{nj}(\boldsymbol{k})\ S_{ij}(\boldsymbol{k})\right\} \equiv \sum_{n\boldsymbol{k}}\ w_n(\boldsymbol{k};E)\ u^{\boldsymbol{k}n}_{ij} \qquad (i\neq j) \tag{8.9}$$

where we defined $u^{\boldsymbol{k}n}_{ij}$ by

$$u^{\boldsymbol{k}n}_{ij} = u^{\boldsymbol{k}n}_{ji} = 2\,\mathrm{Re}\left\{c^*_{ni}(\boldsymbol{k})\ c_{nj}(\boldsymbol{k})\ S_{ij}(\boldsymbol{k})\right\} \tag{8.10}$$

The *gross density of states* is

$$n_i(E) = q_{ii}(E) + \frac{1}{2}\sum_{i\neq j}\ [q_{ij}(E) + q_{ji}(E)] = \sum_{n\boldsymbol{k}}\ o_n(\boldsymbol{k};E)\,\frac{1}{2}\sum_j\ u^{\boldsymbol{k}n}_{ij} \equiv \sum_{n\boldsymbol{k}}\ w_n(\boldsymbol{k};E)\ v^{\boldsymbol{k}n}_i \tag{8.11}$$

The diagonal elements u_{ii} in this expression are defined by (8.10) (i=j) and are twice the *net* density of states.

implementation

DOS is the master routine of this section. The total and partial (gross and overlap) densities of states are computed for a number of equidistant energy values in the range (edosmn,edosmx). The number of energy values, nedos is input by the key `nedos` or `dos energies`; first and second defaults are 0 and 100. The value 0 implies that no DOS is computed. The DOS section performs then only the Mulliken population analysis (see below).

The minimum and maximum energy values can be specified in the same input record as nedos (with ordering: nedos, edosmn, edosmx), but they may be omitted; defaults are edosmn=−4 and edosmx=+1 (a.u.). These values are relative to the fermi energy.

By default only the total DOS is evaluated and output. Gross and overlap densities of states are generated if the keys `gross pop`, respectively `overlap pop` are supplied. Each data record in the corresponding key-input block (see SS^input) specifies the number of the basis function, (respectively the numbers for a pair of functions) for which the partial DOS is to be computed. This information is retrieved from the input file by DOSINP and stored in the arrays idgros() and idover().

The eigenstates $\psi_n(\boldsymbol{k};\boldsymbol{r})$ and eigenvalues $e_n(\boldsymbol{k})$ are on file iteig. The stored expansion coefficients relate however to the orthonormal basis used in the SCF procedure. The first step is therefore the back transformation to the original basis, the plane waves and bloch sums of one-center functions. The transformation matrix has been written on file itdata by BASOVL, together with the overlap matrix of the original basis (S in (8.8)). DOSTRA performs the back transformation to obtain the required expansion coefficients and calls then DOSCON. DOSCON computes the quantities $u_{ij}^{\boldsymbol{k}n}$ and $v_i^{\boldsymbol{k}n}$ ((8.10) and (8.11)) and stores them on file itcon.
DOSCAL finally organizes the calculation of the total and partial DOS The energy bands $e_n(\boldsymbol{k})$ are read from file iteig and the surface integral occupation numbers $w_n(\boldsymbol{k};e_i)$ are computed (OCCUPA) for all required energies e_i . The summation over the bands and k-points yields directly the total DOS (DOSTOT). The gross and overlap DOS are evaluated in DOSPOP by straightforward numerical integration of $u_{ij}^{\boldsymbol{k}n}$ and $v_i^{\boldsymbol{k}n}$.

The DOS output (DOSOUT) is either printed (default) or written to a plotfile. The plot file is generated if the key `dos plot` is found in input. The same input record may also contain the unit number for the plot file itplot (default: 7). The plot file is a formatted file.

POPANA, called by DOS, computes the Mulliken populations. The procedure is completely analogous to that in DOSPOP: the functions $u_{ij}^{\boldsymbol{k}n}$ and $v_i^{\boldsymbol{k}n}$ are summed over the bands and numerically integrated over the BZ. The integration weights refer now not to the surface integral for some energy, but to the volume integration up to the fermi energy: they are the occupation numbers of the one-particle states.

9 Energy: total and cohesive

In the DF formalism, with the customary approximation of motionless point nuclei, the total energy of a system is

$$E = E_T + E_{XC} + E_C \tag{9.1}$$

E_T is the kinetic energy, E_{XC} the XC energy and E_C the coulomb energy.

$$E_T = \sum n_i \quad \psi_i \quad T\psi_i \tag{9.2}$$

The summation includes integration over the BZ; n_i are occupation numbers of the one-particle states ψ_i ; T is the kinetic energy operator, $-\Delta/2$ in atomic units.

$$E_{XC} = \int \varepsilon_{XC}(\boldsymbol{r})\ d\boldsymbol{r} \tag{9.3}$$

$\varepsilon_{XC}(\boldsymbol{r})$ is the XC energy density. It is a functional of the electronic charge density $\rho(\boldsymbol{r})$. The precise form depends on the DF adopted, for example X_α or Vosko-Wilk-Nusair (see SS^XC).

$$E_C = \frac{1}{2}\iint d\boldsymbol{r}_1\ d\boldsymbol{r}_2\ \{\rho(\boldsymbol{r}_1) + \sum Z_\alpha\, \delta(\boldsymbol{r}_1 - \boldsymbol{R}_\alpha)\}\ \frac{1}{r_{12}}\ \{\rho(\boldsymbol{r}_2) + \sum Z_\alpha\, \delta(\boldsymbol{r}_2 - \boldsymbol{R}_\alpha)\} \tag{9.4}$$

The summation runs over all atoms α. Z_α and $\boldsymbol{R}_\alpha$ are the nuclear charge and position.

There are two sources of error in the computation of the energy. The first is from the evaluation of various integrals. The applied numerical integration procedure [chapter III], though fairly accurate, is not exact. The second source of error is the coulomb potential, implicit in the Coulomb energy (9.4). To calculate the coulomb potential from the charge density BAND employs fit functions $\{f_i\}$ for which the corresponding coulomb potentials are known (SS^coulomb potential).

$$\rho = \sum c_i f_i + \delta \tag{9.5}$$

$$V_{Coul}[\rho] \approx V_{Coul}[\sum c_i f_i] = \sum c_i\, V_{Coul}[f_i] \tag{9.6}$$

The coefficients c_i are the least squares solution of the fitting problem, with the constraint that the fit density $\rho_{fit} \equiv \sum c_i f_i$ represent the same total charge as the exact density ρ. δ is the difference between the exact density and the fit density. The presence of δ implicates an error in the computed coulomb energy term.

The cohesive energy is the difference between the total energies of the crystal and the constituting atoms respectively.

$$E_{coh} = \sum E_\alpha - E_{crystal} \tag{9.7}$$

The free atom subprogram DIRAC determines the first term in the r.h.s. of (9.7) almost exactly. The crystal energy could be calculated by numerical integration in the crystal integration grid. Since E_{coh} is often very small compared with the two terms defining it, care is needed lest a meaningless value for E_{coh} is obtained from (9.7), even when $E_{crystal}$ is computed with a relative accuracy of 10^{-4} or 10^{-5}. Therefore we rewrite expression (9.7) in a form that allows the calculation of E_{coh} *directly* by numerical integration, with corresponding precision. The crystal total energy is then defined and computed as

$$E_{crystal} = \sum E_\alpha - E_{coh} \tag{9.8}$$

By interpolation from the tables produced in DIRAC, the free atom functions (the charge density, the coulomb potential and the orbital functions ψ_i and $T\psi_i$) are evaluated in the crystal integration points. The various energy terms in E_{coh} are conceptually the difference of the corresponding terms in the crystal

and atomic total energies respectively. Since these are now computed by integration of the energy densities in the same grid, the obtained difference term is identical to the integral of the difference function. Hence the precision of the integration is preserved in E_{coh} :

$$\sum w_i \; E_{crystal}(\boldsymbol{r}_i) - \sum w_i \; E_{atoms}(\boldsymbol{r}_i) \; = \sum w_i \left(E_{crystal}(\boldsymbol{r}_i) - E_{atoms}(\boldsymbol{r}_i)\right) \; = \sum w_i \; \Delta E(\boldsymbol{r}_i \;) \qquad (9.9)$$

The summation runs over all integration points.

kinetic energy

For each atom α the kinetic energy *E ?????*

$$E_{T\,crystal} \; = \sum T_{ij} \; P_{ij} \qquad (9.10)$$

This form is equivalent to (9.2).

XC energy

For each atom α the free atom charge density $\rho_\alpha(\boldsymbol{r})$, evaluted in the crystal points by interpolation, is used to compute the XC energy density $\varepsilon_{XC,\alpha}(\boldsymbol{r})$. This is then integrated to contribute to the energy term $\sum E_{XC,\alpha}$. The self consistent crystal charge density is used of course for $E_{XC,crystal}$.

coulomb energy

The coulomb energy difference is rewritten to split off the electrostatic coulomb interaction between the unrelaxed free atoms. This large term can be evaluated separately with high precision without much effort (see below). The rest, the relaxation part, is computed by 'normal' numerical integration.

Define

$$\rho_{atoms} \; = \sum \rho_\alpha \qquad (9.11a)$$

$$\rho_{crystal} \; = \rho_{atoms} \; + \rho_{def} \qquad (9.11b)$$

The deformation density ρ_{def} is approximated with fit functions for the solution of the Poisson equation (SS^coulomb potential):

$$\rho_{def} \; = \rho_{fit} \; + \delta \qquad (9.11c)$$

$$V_{fit} \; = \text{coulomb potential due to } \rho_{fit} \qquad (9.11d)$$

$$V_\alpha \; = \text{coulomb potential of atom } \alpha, \text{ including the nuclear potential } Z_\alpha / r \qquad (9.11e)$$

$$V_{atoms} \; = \sum V_\alpha \qquad (9.11f)$$

Then

$$2\;\Delta E_C \; = \iint \mathrm{d}\boldsymbol{r}_1 \; \mathrm{d}\boldsymbol{r}_2 \;\; \left\{\rho_{crystal}(\boldsymbol{r}_1) + \sum Z_\alpha \; \delta(\boldsymbol{r}_1 \; - \boldsymbol{R}_\alpha \;)\right\} \frac{1}{r_{12}} \;\; \left\{\rho_{crystal}(\boldsymbol{r}_2) + \sum Z_\alpha \; \delta(\boldsymbol{r}_2 \; - \boldsymbol{R}_\alpha \;)\right\} -$$

$$\ldots \sum_{\alpha} \iint d\boldsymbol{r}_1\ d\boldsymbol{r}_2\ \left\{\rho_\alpha(\boldsymbol{r}_1) + Z_\alpha\ \delta(\boldsymbol{r}_1 - \boldsymbol{R}_\alpha)\right\} \frac{1}{r_{12}}\ \left\{\rho_\alpha(\boldsymbol{r}_2) + Z_\alpha\ \delta(\boldsymbol{r}_2 - \boldsymbol{R}_\alpha)\right\} =$$

(omitting the spatial arguments)

$$= \iint d\boldsymbol{r}_1\ d\boldsymbol{r}_2\ \left\{\rho_{atoms} + \sum Z_\alpha + \rho_{def}\right\} \frac{1}{r_{12}} \left\{\rho_{atoms} + \sum Z_\alpha + \rho_{def}\right\} - \sum \int V_\alpha(\rho_\alpha + Z_\alpha)\ d\boldsymbol{r} =$$

$$= \int V_{atoms}\ \left\{\rho_{atoms} + \sum Z_\alpha + \rho_{def}\right\}\ d\boldsymbol{r} + \int \rho_{def}\ V_{atoms}\ d\boldsymbol{r} +$$

$$+ \iint d\boldsymbol{r}_1\ d\boldsymbol{r}_2\ \rho_{def} \frac{1}{r_{12}}\ \rho_{def} - \sum \int V_\alpha\ (\rho_\alpha + Z_\alpha)\ d\boldsymbol{r} =$$

$$= \sum_{\alpha\beta} \int V_\alpha\ (\rho_\beta + Z_\beta)\ d\boldsymbol{r} + 2\int V_{atoms}\ \rho_{def}\ d\boldsymbol{r} - \sum \int V_\alpha\ (\rho_\alpha + Z_\alpha)\ d\boldsymbol{r} + \iint d\boldsymbol{r}_1$$

$$d\boldsymbol{r}_2\ (\rho_{fit} + \delta) \frac{1}{r_{12}} (\rho_{fit} + \delta)\ =$$

$$= \sum_{\alpha \neq \beta} \int V_\alpha\ (\rho_\beta + Z_\beta)\ d\boldsymbol{r} + 2\int V_{atoms}\ \rho_{def}\ d\boldsymbol{r} + \int\ V_{fit}\ (\rho_{def} + \delta)\ d\boldsymbol{r} + \iint d\boldsymbol{r}_1\ d\boldsymbol{r}_2\ \delta \frac{1}{r_{12}}\ \delta$$

(9.12)

In the last line of (9.12) the first term is the electrostatic interaction between the free atoms, to be treated below. The second and third term are the relaxation terms, evaluated by numerical integration.

fit error

The last term in (9.12) cannot be computed. It is an error term resulting from the inadequacy of the fit functions to describe the (deformation) density exactly. An upperbound on this error term can be determined for three-dimensional crystals. Let δ have the fourier expansion

$$\delta(\boldsymbol{r}) = \sum \delta_{\boldsymbol{K}}\ e^{i\boldsymbol{K}\cdot\boldsymbol{r}} \tag{9.13}$$

$\boldsymbol{K}$ runs over the reciprocal lattice sites. ρ_{def} contains zero charge, as does ρ_{fit} (and hence δ) because of the constraint in the fit. Hence $\delta_0 = 0$ in (9.13).
The coulomb potential due to δ is obtained from the Poisson equation $-\Delta V = 4\pi\rho$:

$$V_\delta\ = 4\pi \sum_{\boldsymbol{K}\neq 0} \frac{\delta_{\boldsymbol{K}}}{K^2}\ e^{i\boldsymbol{K}\cdot\boldsymbol{r}} \tag{9.14}$$

This gives for the error term

$$\varepsilon = \iint d\boldsymbol{r}_1\ d\boldsymbol{r}_2\ \frac{1}{r_{12}}\ \delta\delta = \int V_\delta\ \delta\, d\boldsymbol{r} = 4\pi \int \left\{ \sum_{\boldsymbol{K}} \frac{\delta_{\boldsymbol{K}}}{K^2}\ e^{i\boldsymbol{K}\cdot\boldsymbol{r}} \sum_{\boldsymbol{K}'}\ \delta_{\boldsymbol{K}'}\, e^{i\boldsymbol{K}'\cdot\boldsymbol{r}} \right\} d\boldsymbol{r} \le \frac{4\pi}{K_{min}^2} \int \delta^2\ d\boldsymbol{r} \qquad (9.15)$$

K_{min} is the smallest nonzero vector of the reciprocal lattice. The integral $\int \delta^2\ d\boldsymbol{r}$ is easily computed and has shown in practice that the fit functions routinely employed are adequate.

This analysis does not include the effect of the inexactness of the fit on the self consistent solution itself. It seems reasonable however to suppose that this may be neglected as regards the energy: a small deviation in the ground state density from the true variational minimum has a quadratic and hence a *very* small effect on the resulting energy.

electrostatic interaction between neutral atoms

The electrostatic interaction between two spherically symmetric atoms A and B at positions $\boldsymbol{R}_A$ and $\boldsymbol{R}_B$ is

$$E_{elstat} = \iint d\boldsymbol{r}_1\ d\boldsymbol{r}_2\ \{\rho_A + Z_A\}\ \frac{1}{r_{12}}\ \{\rho_B + Z_B\} = \int V_A\ \{\rho_B + Z_B\}\ d\boldsymbol{r} = Z_B\, V_A(\boldsymbol{R}_B) + \int V_A\ \rho_B\ d\boldsymbol{r} \qquad (9.16)$$

The last integral is evaluated numerically in prolate spheroidal coordinates. Several other types of elliptic coordinates [Arfken 1970] have been tried but yielded inferior precision.
Let A and B be located along the z-axis, at positions $z=\pm a$. Define coordinates u,v,ϕ by

$$\begin{aligned} x &= a \sinh(u) \sin(v) \cos(\phi) \\ y &= a \sinh(u) \sin(v) \sin(\phi) \\ z &= a \cosh(u) \cos(v) \end{aligned} \qquad (9.17)$$

Numerical integration can be set up as a product formula in the variables (u,p,ϕ) with $p \equiv \cos(v)$. The ϕ-integration yields a simple factor 2π because the functions $V_A(\boldsymbol{r})$ and $\rho_B(\boldsymbol{r})$ are invariant for rotation around the z-axis. So

$$\int V_A\ \rho_B\ d\boldsymbol{r} = 2\pi \int_0^\infty du \int_{-1}^{1} dp\ J(u,p)\ V_A(u,p)\ \rho_B(u,p) \qquad (9.18)$$

The Jacobian is

$$J(u,p) = a^3\ \sinh(u) \left(\cosh^2(u) - p^2\right) \qquad (9.19)$$

For neutral atoms the functions fall off rapidly as u goes to infinity, so that the upper limit on the u-integration can be replaced by a suitable u_{max}. Numerical integration is then performed by a Gauss-Legendre product formula in u and p

$$\int V_A \ \rho_B \ d\mathbf{r} \approx \sum_{i=1}^{n_u} \sum_{j=1}^{n_p} J(u_i, p_j)\, w_i^u \ w_j^p \ V_A(u_i,p_j) \ \rho_B(u_i,p_j) \tag{9.20}$$

For light atoms up to the first series of transition metals n_u =30, n_p =20 give already accurate results. Heavier atoms require more points but still the electrostatic interaction energy can be calculated almost exactly without much effort. The precision of the cohesive energy is determined by the other terms.

electrostatic Madelung energy

In case the crystal calculation is started up with ions, the electrostatic energy has to be split in two terms: the Madelung energy due to the effective ionic point charges plus the interaction between the neutral atoms. The latter has been treated above. The former can be done in any of the standard ways, for instance the Ewald technique. In BAND it is evaluated by a finite lattice sum in real space. For point charges q_α at positions $\boldsymbol{d}_\alpha$ in the unit cell the Madelung energy per unit cell is computed as

$$E_M = \frac{1}{2} \sum_{\alpha\beta\boldsymbol{R}}{}' \frac{q_\alpha q_\beta}{|\boldsymbol{R}+\boldsymbol{d}_\alpha-\boldsymbol{d}_\beta|} \ h(|\boldsymbol{R}+\boldsymbol{d}_\alpha-\boldsymbol{d}_\beta|) \tag{9.21}$$

The prime on the summation signifies omission of the singular terms; $h(r)$ is a screening function. The formal expression for the Madelung energy has $h(r)\equiv 1$, making the sum conditionally convergent. In BAND $h(r)$ is a fermi-dirac distribution function

$$h(r) = \frac{1}{1+e^{(r-r_0)/d}} \tag{9.22}$$

so that the sum is absolutely convergent. The parameters r_0 and d determine the accuracy of the resulting sum. Typical values used are r_0 =40 a.u. and d=3 a.u. For a discussion see SS^coulomb potential.

implementation

ATMFNC interpolates and sums the atomic functions and writes them to file. The coulomb potential is stored on itvatm and the density (both the valence and the total density) on itdatm. ATMFNC calculates also the energy terms $E_{T,\ atoms}$ and $E_{XC,\ atoms}$ by integration over the crystal grid. The atomic total energies E_α are computed in DIRAC. All these energy terms are written to file itdatm after the density values.

Auxiliary routine ELSTAB evaluates the interaction between two neutral atoms (9.20); the elliptic integration parameters are nuelst and nvelst (in common FIXDAT).They can be assigned values via input (keys `u elstat` and `v elstat` respectively); first and second defaults are for nuelst: 40 and 60, for nvelst: 80 and 120.

ELSTAB is called by ELSTAT where all terms are added to the total electrostatic interaction energy (9.16); the Madelung part, a simple summation in ELSTAT, is kept separate. Both energy terms, elstt and emadel,

are passed to ATOMIC where they are written to itdatm, together with the sum of atomic total energies eatoms.
The variables defining the cut-off function (9.22) for the Madelung energy are rmadel (r_0) and dmadel (d); see SS^coulomb potential for the determination of their values.

The crystal kinetic energy matrix T_{ij} is constructed in HAMFIX (one matrix for each k-point in the BZ) and written to itdatm. The final self consistent density matrix P_{ij} is computed in PMATRX and written to file itpmat. The multiplication (9.10) is performed in ENERGY.

ENERGY integrates also the crystal XC energy (9.3), the coulomb terms vatdef and vdef (the 2nd and 3rd term in (9.12) respectively) and the fit error integral $\int \delta^2 \, d\mathbf{r}$.
The fit coefficients are computed in RHOFIT and, at the last cycle of the SCF procedure, written by RHOPOT to file itpot. They are used to compute the fit density and hence the deviation function δ (9.11c). The potential V_{fit} due to the fit is constructed from the fit coefficients (and the fit potentials on itfit) in RHOPOT and written to itpot, after the fit coefficients.

10 Files

BAND employs a large number of files to store data during the calculation. Some of these files can be fairly large. This is in particular so for the file that contains for each k-point the values of the basis functions in the integration points.
On some machines the total amount of data is not the problem, but there may be a severe limit on the amount per file. This has led us to devise a series of subroutines in which all operations with internal files, like reading, writing, open and close, are performed. We will denote this set of routines plus the related data structures as the *filemanager*. The file manager keeps track of the amount of data per file and switches to another file if a particular one is full.

Files are represented in the program as integer variables. The names of these variables start with the characters 'it'; itbas for instance is the file with the basis functions. The *value* of the variable is the unit number of the file. This value may be changed by the file manager during input/output (IO) operations. This happens when the file is full and more data have to be written; the file manager takes another unit to write on, but in the program we do not notice that: IO is still performed with file itbas.
The implementation of the file manager has not only solved the problem with the maximum file-sizes, but also it has increased the programming facilities in BAND. We can now freely open and use files, if some program extension requires so, without bothering which units are available: the file manager knows. One may for instance insert somewhere

```
call flnew (itnew,'unformatted')                    (10.1)
```

with the effect that a currently unused unit number is assigned to the variable itnew; the associated file is opened and rewound.

The file management system is structured as follows. File variables in the program, such as itbas, are not associated with one single unit, but with a sequence, or *string* of files. The value of the variable is the unit number of the currently active member of the string. The members of the string are connected by a pointer structure, implemented as a two-dimensional array indfil(0:maxfil,2); the constant maxfil is the maximum number of units that is in principle available to the file manager. Indfil(ii,1) is the successor of unit ii in the same string and indfil(ii,2) precedes ii. Begin and end of a string have the pointer pointing to itself: the first unit in a string has indfil(ii,2)=ii and the last unit has indfil(ii,1)=ii.

A special string is the string of free files: the units that are available but currently not in use. At the start-up of the program all units 1 through maxfil are members of the free string. Indfil(0,1) points to the first free file; initially indfil(0,1)=1, indfil(i,1)=i+1, i=1,2,... To detect when the last *free* file is reached, the forward pointer of that last free file, indfil(maxfil,1), is set to zero (instead of maxfil, as we would do at the end of any other string). Whenever a new file is requested by the program, the file manager takes the first free unit for it. This is removed from the free string to form a new string on its own; the new string consists of one member at that moment. The main part of subroutine FLNEW, mentioned above, thus reads

```
subroutine flnew (itnew,....)                                    (10.2)
itnew = indfil(0,1)                  pick up a free unit
indfil(0,1) = indfil(itnew,1)        first next free unit
indfil(itnew,1) = itnew              pointers of the new string
indifl(itnew,2) = itnew              ..
..
end
```

Apart from the index array indfil, the file manager employs an array lenfil. This is used to know when the end of a file has been reached. Initially lenfil(ii)=0 for all units *ii*. When *n* words are written to file *ii*, lenfil(ii) is increased by *n*.
Actually it is increased by: *n*×lstor+markio. The extra term markio is used to be on the safe side: depending on the computer the *system* file manager may write some extra information to file while executing a FORTRAN write instruction; consequently the file may already be more filled than the unwarned user expects.
The value of lenfil() is to be compared with the maximum file size, which is given by the constant mxflln in the program; the factor lstor converts the number of words, *n*, to the units in which mxflln is specified; usually this will be in words so that lstor=1.
When lenfil() reaches the maximum file length mxflln, the file manager switches to the next file in the same string. If necessary the string is extended with a unit from the free string.

The analogous procedure is followed when reading from a file. The counter lenfil() is reset at zero when a file is rewound. It is increased again with each read operation, so that we know when we have to switch to the next file to retrieve more data.

All this has the consequence that read and write operations are accompanied by several checks and updates of the file management information system. The same is true for file operations like rewind; rewind(itbas) has now to be interpreted as: *a)* associate with itbas the first unit in its string, *b)* rewind that unit, *c)* reset the file counter(s) lenfil().
To keep the program as transparant as possible all operations with files are performed in the file manager subroutines; the usual FORTRAN statements are replaced by calls to these routines. This makes it also easier to adapt the file manager to future demands. We list the involved routines below with a concise explanation.

Read and write are performed in three routines FLIOI, FLIOL and FLIOR. These are completely similar; the only difference is that they handle respectively integers, logicals and reals. A typical call is

```
call flior ('write',itbas,n,aa)                                    (10.3)
```

This is the analogue of the usual

```
write (itbas) (aa(i),i=1,n)                                        (10.4)
```

Aa may be a scalar if n equals unity. The two statements above are also equivalent in that they have to be interpreted as writing (or reading) exactly *one* record, whatever may happen 'on the background'. So the sequence

```
call flrwnd (itbas)                    rewind                      (10.5)
call flior ('write',itbas,10,aa)
call flior ('write',itbas,10,bb)
call flrwnd (itbas)
call flior ('read',itbas,5,cc)
call flior ('read',itbas,5,dd)
```

has the effect that the first five elements of bb are stored into dd(1:5) (i.e. not the last five of aa).
Furthermore the sequence

```
call flrwnd (itbas)                                                (10.6)
call flior ('write',itbas,10,aa)
call flior ('write',itbas,10,bb)
call flrwnd (itbas)
call flior ('read',itbas,15,cc)
```

results in an error, because it requests the reading of a larger record than has been written. On some machines the usual FORTRAN equivalent is possible and results in the reading of both written records. For reasons of simplicity and error checking we have chosen not to support this in BAND's file manager.
The read/write routines FLIO- can be used only for IO with *unformatted* files.

Opening of new files, deleting of superfluous files and rewinding are performed as follows.

- FLNEW(it,..) supplies a free unit number, opens and rewinds the file and assigns the unit number to the argument.
- FLFREE(it) closes, with status 'delete', unit it *and* all other units in its string. The unit numbers are inserted into the free string again.
- FLRWND(it) assigns to it the first unit in the string, rewinds that file and resets the counter lenfil(it).

The subroutines FLIOI, FLIOL, FLIOR, FLNEW, FLFREE and FLRWND are the only ones that are called in the 'normal' program. A few auxiliary routines are used in the file manager to isolate some specific aspects.

- FLNEXT(it) picks up and rewinds the next file in the string.
- FLADD(it) picks up a free unit and adds iut to the string it.
- FLCLOS(it,..) and FLOPEN(it,..) perform the usual FORTRAN open and close operations (i.e. for *one* unit)

Finally

- FLPROT(it) removes unit it from the string of free files, so that it is not available anymore to the file manager. FLPROT can be activated via input to protect specific units from use by the program; the key is `protect it`, where it is the unit number. Subroutine INIT calls FLPROT to protect the standard input and output files, units 5 and 6 respectively in the current implementation.
- FLDUMP(message,action) writes the state of affairs in the file manager to output. message is a string that will be printed before the information, action specifies what to do with the currently open files: action may be 'delete' or 'keep'. FLDUMP is called by STOPIT when the program is normally terminated, to check whether files are still not closed (action='delete'). For debugging purposes this information can be printed (action='keep') at many places in the program by giving the input instruction `trace files`.

remark
The read and write routines of the file manager are used only for *unformatted* files. A few specific files in BAND are *formatted*. These are the plotfile for the density of states data, the input file for the numerical integration package and the file to which all input for the program is copied. Furthermore the integration package POINTS delivers two files with information. These have not been written (and hence cannot be read) by the file manager. The unit numbers are controlled by the file manager but read and write is performed with the normal FORTRAN statements (in GEMTRY).

11 Form factors

X-ray factors, or form factors are (proportional to) the Fourier coefficients of the charge density. The form factors are denoted $F_{\boldsymbol{K}}$.

$$\rho(\boldsymbol{r}) = \sum F_{\boldsymbol{K}}\ e^{i\boldsymbol{K}\cdot\boldsymbol{r}} \tag{11.1}$$

The summation runs over all lattice points $\boldsymbol{K}$ of the reciprocal lattice.

$$F_{\boldsymbol{K}} = \frac{1}{\Omega}\int \rho(\boldsymbol{r})\, e^{-i\boldsymbol{K}\cdot\boldsymbol{r}}\, d\boldsymbol{r} \tag{11.2}$$

Integration is over the unit cell with volume Ω. The form factors corresponding to symmetry related $\boldsymbol{K}$-vectors are equal, or differ at most by a phase factor. Let $\{\boldsymbol{t},R\}$ be a space group operator, $\boldsymbol{r}'=\{\mathbf{t},\mathrm{R}\}\boldsymbol{r}=\boldsymbol{t}+R\boldsymbol{r}$, $\boldsymbol{K}'=R\boldsymbol{K}$, then

$$F_{\boldsymbol{K}'} = \frac{1}{\Omega}\int \rho(\boldsymbol{r}')\, e^{-i\boldsymbol{K}'\cdot\boldsymbol{r}'}\, d\boldsymbol{r}' = \frac{1}{\Omega}\int \rho(\{\boldsymbol{t},R\}\boldsymbol{r})\, e^{-i\,[\boldsymbol{K}'\cdot\boldsymbol{t}\ +\ \sum_{mn} R_{mn}\boldsymbol{K}_n \sum_p R_{mp}\boldsymbol{r}_p]}\, d\boldsymbol{r} =$$

$$= e^{-i\boldsymbol{K}'\cdot\mathbf{t}}\,\frac{1}{\Omega}\int \rho(\boldsymbol{r})\, e^{-i\boldsymbol{K}\cdot\boldsymbol{r}}\, d\boldsymbol{r} = e^{-i\boldsymbol{K}'\cdot\boldsymbol{t}}\ F_{\boldsymbol{K}} \tag{11.3}$$

where we used the symmetry of $\rho(\boldsymbol{r})$ and the unitarity of the operator R.

BAND calculates the form factors for the stars of $\boldsymbol{K}$-vectors 0..N, counting $\boldsymbol{K}$=0 as the zero-th star. N may be specified via input by the key `formf` or equivalently `xray` ; default N=3.
FORMFA, the master routine, calls first PLANEW to generate the coordinates of N stars of (reciprocal) lattice points, then CELRED to reduce this set to the subset of symmetry unique lattice points and finally FORMF1, where the form factors are actually calculated.
Integral (11.2) is evaluated numerically, by summation over all integration points

$$F_{\boldsymbol{K}} = \frac{1}{\sum w_j}\ \sum w_j\, e^{-i\boldsymbol{K}\cdot\boldsymbol{r}_j}\ \rho(\boldsymbol{r}_j\ ) \tag{11.4}$$

remarks

Equation (11.2) follows from (11.1) by orthogonality of the plane waves. Let $S_{\boldsymbol{K}\boldsymbol{K}'}$ be the overlap matrix of the employed plane waves, evaluated *numerically*

$$S_{\boldsymbol{K}\boldsymbol{K}'} = \frac{1}{\sum w_j}\ \sum w_j\ e^{-i\boldsymbol{K}\cdot\boldsymbol{r}_j}\ e^{i\boldsymbol{K}'\cdot\boldsymbol{r}_j} = \frac{1}{\sum w_j}\ \sum w_j\ e^{i(\boldsymbol{K}'-\boldsymbol{K})\cdot\boldsymbol{r}_j} \tag{11.5}$$

Starting from (11.1) we may define the form factors alternatively by

$$\bar{F}_{\boldsymbol{K}} = \sum_{\boldsymbol{K}'} S^{-1}_{\boldsymbol{K}\boldsymbol{K}'} F_{\boldsymbol{K}'} \tag{11.6}$$

where $F_{\boldsymbol{K}'}$ is given by (11.4). With a perfect numerical integration $S_{\boldsymbol{K}\boldsymbol{K}'}$ equals $\delta_{\boldsymbol{K}\boldsymbol{K}'}$, the analytical value, and $\bar{F}_{\boldsymbol{K}} = F_{\boldsymbol{K}}$. Both $\bar{F}_{\boldsymbol{K}}$ and $F_{\boldsymbol{K}}$ are calculated and printed. This gives some indication of the reliability of the results as regards the numerical integration.

We may expect that the imperfection of the integration shows up more strongly for larger K, so that the overlap matrix S will resemble the unit matrix better if its size is smaller, i.e. if less stars of $\boldsymbol{K}$-vectors are taken into account. Consequently the discrepancies $\bar{F}_{\boldsymbol{K}_0} - F_{\boldsymbol{K}_0}$ for a particular $\boldsymbol{K}_0$ may be due to the presence of larger K in the set and not necessarily to the difference between $F_{\boldsymbol{K}_0}$ as computed from (11.4) and its analytical value (11.2).

\# The form factors provide also a means to check the fit functions used for the calculation of the coulomb potential. Let the latter have a Fourier expansion

$$V(\boldsymbol{r}) = \sum V_{\boldsymbol{K}}\, e^{i\boldsymbol{K}\cdot\boldsymbol{r}} \tag{11.7}$$

From (11.1) and Poisson's equation $-\Delta V = 4\pi\rho$ we may compute the form factors as

$$F^{V}_{\boldsymbol{K}} = \frac{K^2}{4\pi} V_{\boldsymbol{K}} \tag{11.8}$$

$V_{\boldsymbol{K}}$ is determined from the potential values in the integration points, like $F_{\boldsymbol{K}}$ (11.4) . In the program $V(\boldsymbol{r})$ is computed via the approximate expansion of the density in fit functions. Comparison of $F_{\boldsymbol{K}}$ and $F^{V}_{\boldsymbol{K}}$ gives thus an indication of the adequacy of the fit set. See also the note on the fit error in SS^energy.

Relation (11.8) does not hold for $\boldsymbol{K}=0$: F^{V}_{0} is physically meaningless. The corresponding Fourier coefficient of the density relates to the total amount of charge; this particular coefficient is therefore not divided by the volume ($\sum w_j$) in FORMF1.

12 Geometry

The numerical integration schemes in BAND are closely related to geometric concepts: space is divided in *polyhedra*, low-dimensional crystals are envelopped in *boundary planes*, the points are generated in the *irreducible wedge*, etc. In this section we discuss the representation and processing of such data.

planes

A plane consists of all points $\boldsymbol{x}$ that satisfy

$$\boldsymbol{x}\cdot\boldsymbol{v} = d \qquad (12.1)$$

and is therefore represented in BAND by the normal vector $\boldsymbol{v}$ and the distance d; the variable names are usually plane(3) and dplane.
The same plane is obtained when we reverse the signs of both $\boldsymbol{v}$ and d. Often we will be interested in the part of space at a particular side of the plane. With this in mind we resolve the arbitrariness in the sign of $(\boldsymbol{v},d)$ by defining a point to be *inside* the plane if $\boldsymbol{x}\cdot\boldsymbol{v}<d$ and *outside* if $\boldsymbol{x}\cdot\boldsymbol{v}>d$. Reversing the signs means then that we interchange inside and outside of the plane.

orientation and rotation

Geometrical analysis in a plane is most convenient in the *xy*-plane, since we can then neglect the third coordinate. To achieve this we will usually need a rotation of the plane under consideration. The rotation matrix is computed by ROTMAT(v1,v2,rmat). Input are the vectors v1(3) and v2(3) and output is rmat(3,3), the matrix that rotates the direction v1 to the direction v2; the input vectors need not be normalized. In the mentioned application v1 would be the normal of the plane and v2 the *z*-axis, i.e. the vector (0,0,1).

The problem at hand does not uniquely define the rotation matrix: infinitely many unitary transformations v1 → v2 exist with determinant +1. ROTMAT takes the shortest possible arc of rotation; the fixed vector, the axis of rotation is the vector product v1×v2.
This particular convention is useful when some particular compounded rotations have to be constructed. Assume for instance that $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ are two vectors and we want to rotate them such that $\boldsymbol{s}_1$ becomes the x-axis and $\boldsymbol{s}_2$ lies in the *xy*-plane. Let then $\boldsymbol{u}_x$ and $\boldsymbol{u}_z$ be the *x*-axis (1,0,0) and *z*-axis (0,0,1) respectively. The rotation matrix rmat is calculated by

```
    call r3vecp (s1,s2,axis)              vectorproduct of two vectors
    call rotmat (s1,ux,rmat1)             rotate s1 to the x-axis
    call rotate (rmat1,axis,axis2)        rotate a vector
    call rotmat (axis2,uz,rmat2)          rotate (s1,s2) to xy-plane
    do 10 i=1,3
 10 call rotate (rmat2,rmat1(1,i),rmat(1,i))          product rotation
```
(12.2)

Since axis is orthogonal to $\boldsymbol{s}_1$ (and $\boldsymbol{s}_2$), axis2 is orthogonal to the *x*-axis (the rotated $\boldsymbol{s}_1$) and lies thus in the *yz*-plane. ROTMAT constructs then rmat2, which rotates axis2 to the *z*-axis, *using the x-axis as the axis of rotation.* So rmat2 leaves $\boldsymbol{u}_x$, the rotated $\boldsymbol{s}_1$, invariant, hence the compounded rotation transforms $\boldsymbol{s}_1$ into the *x*-axis, as required. This procedure is used for example in PYRPT4 where a pyramid is rotated (PYRROT) to some standard orientation.

lines

Lines in the *xy*-plane are defined by (12.1), where $\boldsymbol{x}$ and $\boldsymbol{v}$ have now only two components. In analogy with the planes we define the inside and outside of a line ($\boldsymbol{x}\cdot\boldsymbol{v}<d$ and $\boldsymbol{x}\cdot\boldsymbol{v}>d$).
By the *direction* of a line we will understand the angle ϕ that its normal makes with the positive *x*-axis ($0\leq\phi<2\pi$). Any set of lines to be processed is ordered in BAND such that their directions are in increasing order.

polygons

A polygon (in the *xy*-plane) is defined by its sides, i.e. by a set of lines with their orientations such that the polygon is inside each line.
To find the vertices of the polygon we order the sides according to their directions. The intersection point $\boldsymbol{x}$ of an adjacent pair of lines is the solution of a linear 2×2 system

$$\boldsymbol{v}_i \cdot\boldsymbol{x} = d_i \tag{12.3a}$$

$$\boldsymbol{v}_{i+1} \cdot\boldsymbol{x} = d_{i+1} \tag{12.3b}$$

Given a set of lines the polygon defined by them may have fewer sides if one or more of the lines are redundant; fig.3 shows (part of) a set of lines, of which the numbers 2 and 3 are redundant; they are excluded, or cut-off one might say, by the lines 1 and 4.

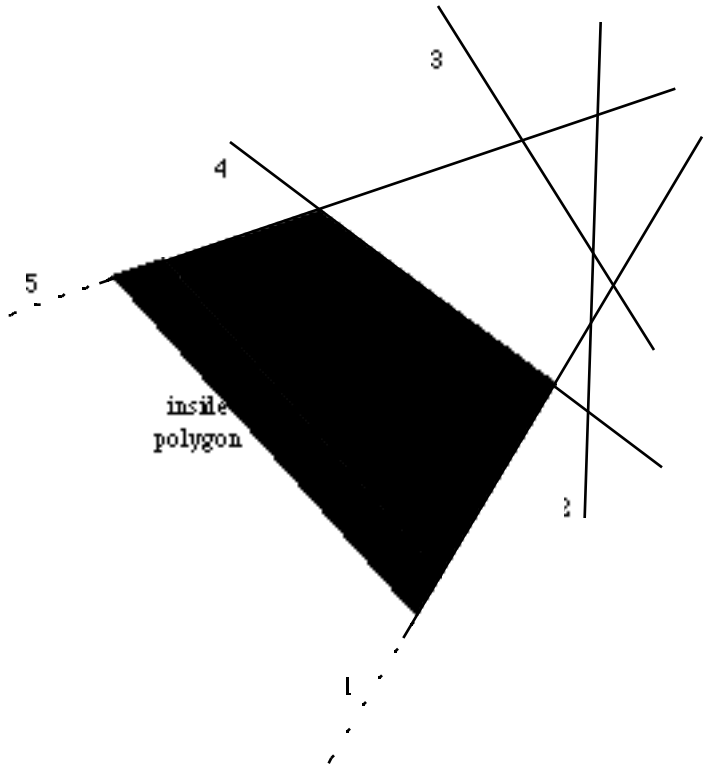

Fig.3. The shaded area is (part of) a polygonal region, defined as the space inside a set of lines. The lines 2 and 3 are redundant.

Considering only the direct neighbours we see that line 3 is cut off by 2 and 4, but line 2 is not excluded by its neighbours 1 and 3. In general we have say lines *i* and *j*, excluding all lines *i*+1..*j*−1 between them. At least one of these excluded lines is then cut off also by its own direct neighbours. POLYGN uses this fact to remove from an input set of lines all redundant ones: for each line exclusion is decided only with respect to its neighbours. Having traversed the whole set, POLYGN restarts the loop if one or more lines were removed, until no more redundant lines are found.

polyhedra

A polyhedron can be defined as the region of space inside a set of planes. Again, given a set of planes some of these may be redundant and should be removed.
Each of the remaining proper planes defines a 'face' of the polyhedron. The relevant part of that plane is a polygon, defined by the intersection lines with the other planes. A polyhedron is in BAND represented as *a)* a set of planes plane(3,nplane),dplane(nplane) plus *b)* a list of vertices vertex(3,*) with an index array index(nplane+1). The vertices corresponding to plane *i* are the numbers index(i)+1 through index(i+1); Index(1)=0. Each vertex occurs at least three times in the list, once for each face it belongs to. The subset of vertices of a particular face are in clock-wise order (viewed from inside the polyhedron), so that, when the normal on the plane is rotated to the positive z-axis (the standard orientation) the rotated vertices are in increasing order of their direction angles. This ordering of the vertices is assumed (and checked) for instance in the integration package POINTS (the routines PYRPT3 and PYRPT4).

Routine POLYHE generates the polyhedron data structure from an input set of planes; redundant planes are removed. The algorithm is as follows. For each plane the rotation matrix is constructed that rotates it to the *xy*-plane. In the rotated frame all intersection lines with the other planes are computed. The resulting polygon is analyzed (POLYGN): if all sides are redundant, that is, if the 'inside' of the polygon does not exist, then the corresponding plane is redundant for the polyhedron and can be removed; otherwise the vertices are computed, back rotated to the original coordinate frame and added to the list.

The line of intersection of a plane ($\boldsymbol{v}$,d) and a plane parallel to the xy-plane, with z-coordinate z_0 is easily found: the normal on the line is (apart from a normalizing scale factor) given by the x- and y-components of $\boldsymbol{v}$; the distance parameter d_{xy} for the line is (fig.4)

$$d_{xy} = \frac{d}{\cos(\theta)} + z_0 \ \mathrm{tg}(\theta) \qquad (12.4)$$

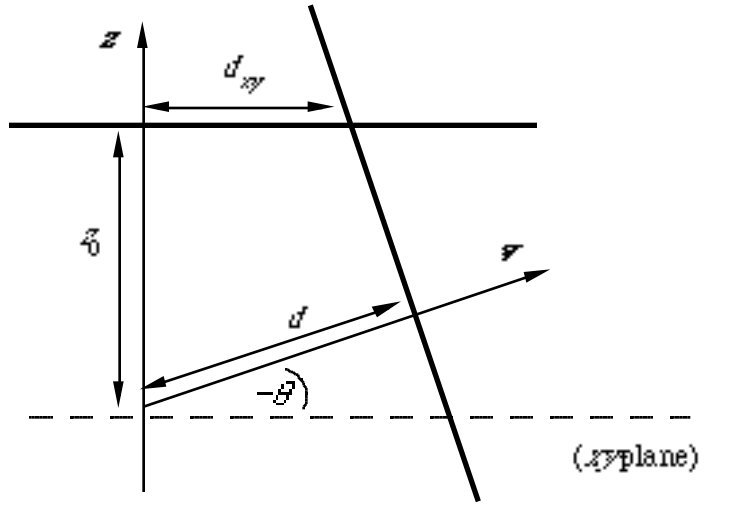

Fig.4. A plane, parallel to the xy-plane at $z=z_0$ is cut by a second plane ($\boldsymbol{v}$,d). The normal $\boldsymbol{v}$ makes an angle θ with the xy-plane. The distance d_{xy} is given by (12.4); see text.

symmetry

The irreducible wedge of a polygon is constructed by PLGIRR. Input are the vertices of the polygon (ordered) and output the vertices of a connected irreducible wedge (a polygon again). In order to construct this PLGIRR sets up an index array for the sides of the polygon. The index is 1 for a side belonging to the wedge, 0 for a side outside (which must then be symmetry equivalent to one of sides belonging to the wedge), and −1 if the boundary of the wedge cuts the side in two equal parts. All indices are initiated at 1. A loop over the sides is then executed, starting with the first and counting upwards. For each of them the equivalent sides are found and assigned index 0 (: to be removed). The loop is interrupted when the boundary of the wedge is reached. This is the case as soon as a side to be considered has already index 0, so that it falls outside the wedge, or when one of the symmetry operators interchanges the two vertices belonging to the side: only half of the side belongs then to the irreducible region (index −1).

Next a second loop is performed, starting with the last side and counting backwards until the other boundary of the wedge is found. In this way the irreducible wedge is constructed 'around' side 1, by traversing the circumference of the polygon in both directions until the edge of the wedge is encountered.

The final index array is then used to compute the vertices of the symmetry unique subregion.

The origin is a special point. Assuming the symmetry group not to be trivial, the origin is not one of the vertices of the original polygon. In many cases however it is a vertex of the irreducible wedge; if this is the case PLGIRR permutes cyclically all final vertices such that the origin is the first in the output list of vertices.

If the original polygon is halved (only one reflection plane for example), one of the sides of the wedge passes through the origin, but the origin is not one of the vertices. Nevertheless it may be convenient (SS^BZ integration) to define the origin as a vertex also in this case, spanning an angle π. This is not done by PLGIRR, but to facilitate this adaptation PLGIRR permutes also in this case the final set of vertices such that, if the origin is to be added, it fits in between the first and the last of the output vertices.

In three dimensions we may need the irreducible wedge of a polyhedron. Associate with each of the polygonal faces of the polyhedron the pyramid with the polygon as its base and the origin as the top. The symmetry analysis is then a two-step proces. First we retain only the symmetry unique pyramids by checking which of the normal vectors are transformed into each other by the symmetry operators (the corresponding faces are then necessarily also equivalent). Then each of the remaining polygonal faces is subsequently rotated to the xy-plane and dealt with by PLGIRR.

simplices

Simplices play a role in BAND in connection with the integration method in *k*-space. Simplices are defined in any n-dimensional space by n+1 points: the simplex in one dimension is an interval, in two dimensions it is a triangle, etcetera.
A special problem occurring in the generation of *k*-space integration points is the subdivision of a simplex into smaller simplices by repeated bisection of the edges. SIMPLS performs this task. Input is a generating simplex simplx(ndim,ndim+1) and the number of refinement-steps nmesh; ndim is the dimensionality and hence the number of coordinates for each of the (ndim+1) points.
Output is a list of distinct points point(ndim,npnt) and an index array idsimp(nrow,nsimpl); nrow is the row-dimension of idsimp (must at least be ndim+1); nsimpl is the generated number of simplices of the specified refinement. Idsimp(1,i), idsimp(2,i),.. idsimp(ndim+1,i) specify the vertices of the i-th simplex by pointing to entries in the list of points (point). Each refinement step splits every simplex into nsub=2^{ndim} smaller ones, so that the total number of smallest simplices will be nsimpl=$2^{ndim \times nmesh}$.

The algorithm is basically a multiple nested loop, one for each level of refinement. Each of the loops runs over all nsub sub-simplices of the one-level-larger simplex.

```
do i1 = 1,nsub
      ..                                  construct the i1-th subsimplex
   do i2 = 1,nsub
      ..                                  the i2-th subsub(...)simplex
      do i3 = 1,nsub
        ..                                (etcetera)
```
(12.5)

Since the number of nestings, nmesh, is a variable the loops cannot be implemented explicit; they are constructed implicitly, with 'goto' statements and tests on loop termination; the loop counters are stored in

array idsub(nmesh); array vertex(ndim,ndim+1,0:nmesh) stores the vertices of the simplices under current treatment, one for each level; level 0 is the input simplex.
A subsimplex may have one of its vertices in common with its parent simplex; the other vertices are then the midpoints of the adjacent edges. Other subsimplices may have only a particular set of these midpoints as vertices. In general the vertices of the subsimplex can then be defined as the averages of particular pairs of parent vertices, where in some cases the average may be taken of one and the same point. Each of the 2^{ndim} subsimplices is thus characterized as a special set of ndim+1 'pairs'. This data structure is the fixed data array idpart(2,ndim+1,2^{ndim}) in SIMPLS. Inspection shows that with an appropriate ordering of the subsimplices the structures for the lower dimensional cases can conveniently be embedded in those for higher dimensions. The implemented array idpart is the 3D case; when SIMPLS is called with ndim<3 the appropriate submatrix is used.

13 Input

This section deals with the processing in BAND of input data and with the ways in which the operation of the program can be directed via input. A few remarks will be made concerning output.
Input is optional in many respects; omission leads to default settings. Input relating to a large number of details is dealt with in the corresponding Software Sections and is not discussed here. Reading this section should supply sufficient information however to run the program.

BAND reads only one input file. Restart possibilities have not yet been implemented.
INIT excludes the unit numbers 5 and 6 from use by BAND's file manager (see SS^files), assuming that these are associated with the standard input and output channels respectively.

The input file is defined to have two parts: the comment part and the data part. The comment part, which may be empty, consists of all consecutive first records that have the character 'c' in the first column; the other records constitute the data part. The data part and hence the input file is defined to end with either the FORTRAN 'end-of-file'-code, or a specific record (see below), whichever comes first.

INIT copies the complete input file to output. Simultaneously the comment part is copied to a file itcomm and the data part to itinpt. These formatted files are processed further in HEADIN and GETINP, which are both called from INIT.

HEADIN prints the 'heading' of the output, directly after the copy of the input file. The heading provides some general information about the program and the calculation, such as the release number of the program, the date of the release and the requested CP-time and memory-usage for the current run. Naturally some of this information is picked up by machine-dependent code. This has to be adapted when BAND is implemented on another computer. The involved routines are SECTIM and HEADIN.

HEADIN retrieves also the comment lines again from itcomm and writes them (without the first-column 'c') into the output heading.

GETINP analyzes most of the information on itinpt. The rest, for example the characteristics of the basis functions, is written to a new file unit; this is (at the end of GETINP) assigned to itinpt, replacing the old value. In various parts of the program the remaining information is extracted from (the new) itinpt.

All (data) input is structured by keys; a key is a (short) string, usually a single word. Each key defines a separate key section. Two forms are used. In the first form the section consists of only one record; it contains the key and, depending on the case, additional (numerical) information. The second form is a sequence of records: the first record is the key; the last contains (only) two asterisks, '**', signifying the end of the section; the intermediate records provide all information; in one special case the first (key) record has to contain also additional information.
We will denote the two forms by key *record* and key *block* respectively.
The form of the key section is not optional: each admissable key is associated with a particular form. Generally speaking the block form is employed only for keys that relate potentially to large amounts of data, such as a list of basis functions or atom coordinates. All keys that have the block form will be mentioned in this section.

Empty lines in input are allowed and meaningless; INIT omits them when itinpt is written.

All numerical information is 'free format': the absolute positions of numbers and the form in which reals are specified is irrelevant.

The ordering of the keys in input is free and has no implications (with one obvious exception, as we will see).

Most of the keys are optional: omission leads to defaults for the corresponding variables or options. For some keys *second* defaults are available; these are activated by giving the key without specifying further information.

GETINP checks the occurrence of all keys that are known to the program. Unknown keys are neglected and their presence in input has no consequences. As said before, in most cases GETINP extracts all information and assigns values to the variables and/or sets the options that correspond to the encountered keys; for some keys the information is only globally surveyed and copied to a new file to be processed later.

The occurrence in input of a few special keys is 'kept in mind' by storing them in a list of keys, the array keylst. This operation is performed by routine KEYSET, which is called from GETINP whenever such a key is found. The logical function KEY checks the presence of a specified key in the list, i.e. KEY('this key') tells us whether `this key` has been stored in the list.

This structure has made it easy to implement new options or to adapt the operation of BAND to new insights, either permanently or temporarily for testing or debugging purposes. Suppose for instance that we consider the replacement of some algorithm by a possibly more efficient one. To test this we would like to compare the two alternatives in a number of calculations. Instead of keeping two versions of the program during the testing period, we just define a new key, say `algorithm`, and insert a few lines in GETINP to search for that key and to call KEYSET when it occurs. The alternative algorithm can then be implemented side by side with the original one, with a simple if-then-else structure:

```
if (key('algorithm')) then
   ..
   (new algorithm)
   ..
else                                                    (13.1)
   ..
   (old version)
   ..
endif
```

In this way we can keep both possibilities available as long as we wish, without bothering about the maintenance of two programs instead of one.

Since BAND is a program in perpetual development, the set of keys changes rather often. The keys mentioned in this section and in other Software Sections may therefore not exhaust the set employed in BAND; at the other hand some of the keys may not be in use anymore. One should examine the code of GETINP to ascertain which keys are in fact recognized in the current release.

In the next paragraphs we discuss the input of necessary information, such as the geometry and basis functions. After that we will examine keys related to output printing and the 'control' keys, which determine the operation of the program in general.
The keys are typed outlined (`lattice`, `debug`). If the block form is to be used for the corresponding data, this will be indicated between brackets. Keys may have synonyms; the alternatives will be listed (`sto`, `slater`). In most cases a key is recognized also when it is embedded in a larger string; for instance the string `slaters` is recognized as denoting the key `slater`.

geometry

Information must be provided concerning the Bravais lattice of the crystal, its dimensionality, the coordinates of the atoms and the units in which the data are specified on the input file.

`angstrom`. Lattice vectors and atom coordinates are interpreted as being specified in ångstrøms. Default (omission of the key): atomic units.

lattice (block). Each record contains a lattice vector: three cartesian coordinates; if fewer coordinates are specified zeros are supplemented; at least one coordinate per record must be given (empty records are discarded). The number of records is the dimensionality ndim of the crystal.

The lattice vectors (in atomic units) are stored in array avec(3,3). The inverse-transpose, which describes the reciprocal lattice (apart from a factor 2π), is stored in bvec(3,3).

natural. The positions of the atoms are specified in natural units, i.e. in units of the lattice vectors. Default: cartesian units.

If the dimensionality of the crystal is less than three, only the first ndim coordinates can possibly refer to the lattice vectors; the others are automatically cartesian.

The (cartesian) coordinate values are stored in array xyzatm(3,natomt). Natomt is the total number of atoms in the crystal unit cell.

atoms (block). The positions of the atoms. The key has to be specified anew for each (chemical) type of atom that occurs in the crystal. The number of types of atoms, ntyp, is defined as the number of occurrences of the key atoms. Atoms belonging to different types as defined here cannot be symmetry equivalent.

This key is the exceptional case in which the leading key-record itself must contain additional information: the atomic number (=the nuclear charge).

Each of the intermediate records in the block gives the coordinates of one atom; zeros are suppplied when fewer than three coordinates are found.

Note that the meaning of the input coordinate values depends on the absence or presence (anywhere in the input file) of the keys natural and angstrom.

function sets

Data have to be supplied concerning the free atoms that make up the crystal (key dirac), the Slater type valence basis functions (sto) and the Slater type fit functions (fit). For each of the ntyp types of atoms, as defined above, dirac, sto and fit are searched for. Dirac is obligatory, but sto and fit are optional. The order in which the keys dirac, sto and fit occur is relevant (this is the exception to the rule that the order of keys in input has no meaning). In the first place: the keys atoms and dirac correspond in their order of appearance. In the second place: for each of the atom types GETINP searches the keys sto and fit after the corresponding dirac-block but before the next dirac-block (or the end of input); if they are not found in that part it is assumed that they have been omitted for that type. In the third place if both the fit-block and the sto-block are present for a certain type, the sto-block must precede the fit-block.

dirac (block). The first record in the block (following the key) states the number of numerical one-electron states natorb to be computed and the number of them that are to be interpreted as core states in the crystal ncore: two integers; omission of the second implies ncore(ityp)=0.

The next records give for each of the natorb orbitals n,l,q: the main and angular quantum numbers and the number of electrons. Example: '3 2 7' implies 7 electrons in the 3d-shell of the spherically symmetric atom. The occupation q may be omitted; default: fully occupied ($q=4l+2$).

Anywhere inside the dirac-block the following additional keys may be supplied:

valence (or: basis). The numerical valence states are incorporated in the crystal valence basis.
Default (omission of the key): no.
Note that the numerical core states are used anyway.

fit. For each numerical one-electron state which has an angular quantum number $l \le$ lfit, the *square* of the orbital (i.e. the orbital density function) is used as a *spherically symmetric* fit function (a one-center fit function with l=0) in the crystal. The value of lfit may be specified in the key record; default: lfit=0.
Absence of the key implies lfit=−1: no fit functions are derived from the numerical orbitals.

radial. All one-center functions, both those from DIRAC and the Slater type valence and fit functions are represented in BAND as tables $f(r_i), i=1..$nr. The radial coordinates r_i constitute a logarithmic grid (r_{i+1}/r_i is constant). Key radial specifies the number of points in this grid: nr, the first value r_1 and the last value, r_{nr}, in that order. Defaults are used for absent data (nr=2000, $r_1 = 10^{-4}$ a.u., r_{nr} =40 a.u.).

Remark: the radial grids (one for each type of atom) are stored by BAND in array rad(nrx,ntyp); nrx is the maximum nr. of radial points in any of the grids. After the setting-up of the tables, the values in rad are replaced by the reciprocals 1/rad; this is more convenient in the interpolation routines (ATMFNC, BASPNT, FITPNT). In some places in BAND the array is accordingly denoted rinv(nrx,ntyp). The array nr(ntyp) stores the number of radial points in each grid.

valence, basis, sto, slater (block). Each record characterizes one set of Slater type functions for the valence basis by three variables n,l,α: two integers and one real. The corresponding functions are $Z_{lm}(\Omega)\, r^{n-1}\, e^{-\alpha r}$, centered on the atoms of the type under consideration.

fit, stf (block). Analogous to the previous; the functions are used in the fit set to describe the (deformation) charge density in the crystal.

plane waves. Apart from the one-center numerical orbitals (from DIRAC) and the Slater type orbitals, the valence basis may contain plane waves. The plane waves $e^{i(\boldsymbol{k}+\boldsymbol{K})\cdot\boldsymbol{r}}$ used for each k-point $\boldsymbol{k}$ in the BZ, are characterized by the lattice points $\boldsymbol{K}$ of the reciprocal lattice. The key record must state the number of stars of these lattice points to be used in the valence basis, nwavst. Star no.0 consists only of the central point $\boldsymbol{K}$=0. Each subsequent star consists of all $\boldsymbol{K}$ with the next higher distance to the origin, regardless of symmetry relations.
The central 'star' is used only if no other valence functions are employed. Otherwise it is omitted to prevent dependency problems in the valence set; only the star numeros 1 through nwavst are taken then. The resulting number of plane waves in the basis (per k-point) is nvalwv.

print directives

The amount of printed output is determined by a number of general print options and may further be directed by print instructions.

The general print options are encoded by the variables iprntp, iprnti, iprnts, iprnte and iprntr in common FIXDAT. They determine the general output levels for respectively the *p*reparation part, the numerical *i*ntegration package, the *S*CF-procedure in general, the *e*igensystems of the iteratively computed hamiltonian and the *r*esults (properties section). The higher their value, the more is printed. The (low) default settings provide already a fair amount of information; more output is usually only required to test accuracies and to examine intermediate results. The corresponding keys and defaults are

`print prepar`. defaults (first and second): 1, 2

`print integ`. defaults: 1, 2.

`print scf`. defaults: 0, 1.

When the SCF procedure encounters convergence problems, the value of iprnts is automatically increased in SCFTST (but not higher than 2). This induces the output of information which may clarify the type of convergence problem.

`print eig`. first default: 0 : eigenvalues at the first and the last cycle; no eigenvectors.

second default: 1 : complete eigensystems at the first and last cycle.

other values: 2 : (only) eigenvalues at all cycles. 3 : complete eigensystems at all cycles.

`print prop`. defaults: 0, 1

More specific print instructions are activated by keys that resemble those of the previous set, but that are processed differently in the program. All these keys have the form `print abc`, where *'abc'* is some special string. Whenever such a key is found by GETINP, i.e. when '*abc'* does not equal (or contain) *'pre'*, *'int'*, *'scf'*, *'eig'* or *'prop'*, routine PRNTST is called, which adds the string 'abc' to a list of print instructions, stored in the array prtlst. The occurrence of a specific string in the list is checked by the logical function PRNT-: PRNT(*'abc'*) is true if *'abc'* has been added to the print instruction list. This set-up is analogous to that for the keys (cf. routines KEYSET, KEY) and has been devised to adapt the output easily to new demands. The print strings recognized by BAND are

`print fermi` : detailed information about the determination of the fermi energy at every cycle and related data.

`print fit` : fit coefficients and related data at every cycle.

`print occup` : the occupation numbers for all one-particle states at every cycle.

general control

`execute` (SS).BAND stops after execution of SS; *'ss'* must be the name of a subroutine or section that is recognized by BAND's controller; all major subroutines satisfy this requirement (see SS^control).

`skip` (SS1, SS2,..). BAND skips the sections or subroutines SS1, SS2, ... Again *'ss1'*, etcetera must be recognized by the controller. Any consequences of the not-executing are not (yet) taken care of in BAND; the program may even 'crash' because e.g. variables have not been computed. In practice this option is consequently only safe for a few specific sections (e.g. ELSTAT: the electrostatic interaction energy is not computed then, but this has no further consequences; also various parts of the properties section may safely be skipped).

The sections to be skipped are stored (by routine SKIPST, called from GETINP) in a 'skip-list', the array skiplst; the presence of a specific string in the list is checked by the logical function SKIP(*'ss'*). Compare the manipulation of keys and print directives.
The skip-list is printed in the output heading.

Of course it is an easy matter to extend the applicability of the skip-command beyond the standard sections and routines in BAND. Execution of any part of the program may be subjected to the value of SKIP(*'ss'*); *'ss'* has thereby automatically been made into a recognizable name. We have done so for instance with the *printing* of all overlap populations (in POPANA); this output is suppressed now by the input command: `skip (overlap)`.

Other keys that influence the operation of BAND in a general way are

`spin`, `unrestricted`, `polarized` or `magnetic`. Both spins are treated independently as regards the potential, charge density, eigensystems. The basis sets are identical (the free atom equations solved in DIRAC are spin-restricted). Omission of the key implies a spin-restricted crystal calculation.

`debug`. The default value of all general print options are set so high that all possible output is generated. Moreover BAND will not stop, as it would do otherwise, when intermediate results are suspect.

`test`. This incorporates most of the `debug` effects. In addition some (other) defaults attain different values. In particular integration levels in real space and in *k*-space are lowered. The default resettings are performed by TESTST.

keys with block type input

We conclude this section with the enumeration of all keys that carry with them in the input file a block-structure, as defined before:

`lattice`, `atoms`, `dirac`, `sto` and `fit` have already been mentioned.

`integration` provides information for the integration package. See SS^integration.

`gross population` and `overlap population` determine which partial densities of states have to be computed. See SS^DOS.

14 Integration

Integrals over the crystal unit cell are evaluated by numerical integration. The integration formula is based on a partitioning of space in atomic polyhedra, core-like spheres and (except for 3D crystals) an outer region. For each of the sub regions efficient product-Gauss rules are generated. The procedure is described extensively in [chapter III].

In other software sections some specific aspects are dealt with, such as the construction of the atomic polyhedra and the irreducible wedge (SS^geometry) and the computation of the space group operators and their application (SS^symmetry). Here we mention a few details that have not been covered yet.

The theory and global set-up of the integration formulas are discussed in chapter III. The integration package POINTS is suited for periodic systems as well as for molecules; the latter have been used extensively to test the performance and we will therefore refer to the molecular application from time to time in the discussion below.

POINTS is called from GEMTRY. GEMTRY supplies the necessary data (atom positions, lattice structure and integration parameters) to POINTS via file itipnt. POINTS returns a file with points, itpnt, and a file with geometric data, itgeom. The latter contains in particular the space group operators and the division of the atoms in sets of symmetry equivalent ones.

The file with points is processed further in RPNTID. Each record written by POINTS contains precisely one complete set of symmetry equivalent points. After the reorganization in RPNTID the block structure on the file is such that *a)* the number of points per block does not exceed a prescribed maximum length npx and *b)* each block contains only complete sets of symmetry related points. Also written to this file is information concerning the symmetry relation between the points. This serves to facilitate in various parts of BAND symmetry operations based on numerical integration: e.g. the symmetrization of a function by averaging over the equivalent points (the density) and the expansion over all points of a symmetric function that has been computed in the unique points only (the potential). Furthermore RPNTID computes for each point which atom is nearest by (according to some metric); this is used by CHARGE to partition the self consistent charge density over the atoms.

Each block on the resulting points file (itpnt) consists of the following records:

1 np: the total number of points in the block.

2 xp(np): the x-coordinates of the points.

3,4 yp and zp(np): the y- and z-coordinates.

5 wp(np): the integration weights.

6 npsym: the number of symmetry unique points in the block.

7 nequiv(npsym): the number of equivalent points for each of the unique points.

8 idatom(np): for each point the index of the atom nearest by. Idatom() may have a positive or a negative value; the indicated atom is abs(idatom()), positive and negative values signify that the points are inside, respectively outside the atomic sphere.

As discussed in SS^workspace itpnt is reorganized several times (RPNTRE, REORGF). The same structure is maintained. Only the maximum block length npx may be changed.

input

A file itintg is opened in INIT and used by BAND to collect integration parameters. In GEMTRY this is combined with the atom coordinates etc. to write the input file for POINTS itipnt. Integration parameters can be specified via input with the key `integration`. The contents of the associated data block (see SS^input) is copied to itintg and finally to itipnt. This data block is read again by the input routine of the integration package, RDINTG. It must consists of a sequence of keys with additional numerical information. RDINTG knows two types of keys: *a)* single records containing the key and a number, and *b)* one record with the

key and a second record with an array of numbers, one for each type of atom in the system. The admissible keys, their meanings and the default values are listed below; the names of the corresponding variables are identical to the keys.

type a: single-value keys

1 accint; default: 3.5. This is the general accuracy parameter. In normal operation only this parameter should be supplied, if any at all. Almost all other parameters depend by default on accint.
The computed integration formula is intended to give an accuracy of accint significant digits for integrals that normally occur in electronic structure calculations.

2 accsph; default: accint. Analogous to accint accsph is an accuracy parameter referring to the number of significant digits of the formula, in this case for the radial integration in the atomic spheres only.

3 accpyr; default: accint. Similarly accpyr refers to the atomic polyhedra and the constituting pyramids.

4 accpyu; default: accpyr. The integration over the atomic pyramids is a threefold product formula in variables *u*,*v* and *w*. *U* and *v* describe more or less the angular integration and *w* parametrizes the radial variation relative to the central atom in the polyhedron. Accpyu specifies of course the accuracy of the outer integration 'loop' over the variable *u*.

5,6 accpyv and accpyw; defaults: accpyr. Similar to accpyu, now for the variables *v* and *w*.

7 accout; default: accint. The accuracy parameter for the outer region (not relevant for three-dimensional crystals). The treatment of the outer regions in POINTS is not very sophisticated compared with the spheres and polyhedra: notest functions are employed to monitor and tune the number of points to the specified accuracy. The parameter name accout is therefore a little misleading; of course it is meant to have the meaning suggested, but the necessary code has not yet been developed. Fortunately the outer regions are in general not very relevant for the integrals. Moreover accout does determine the number of points in the outer region (see below) and our experience thusfar suggests that the resulting accuracy is at least good enough in the large majority of cases.

8 dishul; default: 2.3×rsphx. Rsphx is the radius of the largest atomic sphere (see parameter rspher below). Dishul is the distance from the outer atomic spheres to the enveloping boundary planes that constitute the inner limit of the outer region. The atomic spheres have usually radii in the order of 1 a.u., so that the outer region starts at approximately 3 à 3.5 a.u. from the nuclei.

9 frange; default depends on the nuclear charge and on accout. Frange specifies the distance at which all one center functions become negligible (not counting the multipole potentials). In the integration formula it is the position of the outward limit of the outer region, measured from the outermost atoms.
The default depends on the nuclear charge of the heaviest atom, Z_X and on the integration parameter accout. The relation below is based on some trial and error and on common sense. Set *fr*=10 ($Z_X \le 2$), 12 ($Z_X \le 18$), 15 ($Z_X \le 54$) or 20 (otherwise). Then frange=*fr*×(3−2$e^{-0.07\times\text{accout}}$). Fig.5 depicts frange as a function of accout for Z_X =30.

10 nouter; default: 1+nint(2.5/dishul).
The outward integration in the outer region is logarithmically subdivided in nouter subintervals. In most situations two subintervals is optimal: a relatively narrow interval near the outer atoms and a

large interval for the decaying tails of the functions. If dishul attains extreme values, some adaptation is necessary of course, hence the applied default relation.

11 outrad; default: 1.3+0.9×accout.
This paramater governs the 'radial' integration in the outer regions (outward, away from the atoms). The default relation has been fitted by a number of test calculations on various molecules. Outrad is the number of integration points per outward subinterval (see nouter above).

12 outpar; default: 0.5+1.5×accout.
Parameter for the 2D integrations parallel to the boundary planes of the outer region. The default relation has again been determined from test calculations.

13 linrot; default: lintgx+1. This parameter is relevant only for systems with an axis of infinite rotational symmetry such that all atoms lie in a straight line. The symmetry unique points can all be chosen in a half plane: we have in fact a 2D integration problem. BAND however needs also the equivalent points to evaluate correctly integrals over the rotational angular variable.
For each of the symmetry unique points (in the half plane) POINTS generates a circle of equidistant points. The necessary number of points depends on the angular momentum quantum numbers of the integrands. The maximum *l*-value is lintgx (see below); the required number of angularly equidistant points to integrate the periodic functions of this order is lintgx+1.
If linrot is specified on input, the generated number of points on the circle is the smallest integer multiple of linrot that equals or exceeds lintgx+1.

type b: keys with ntyp data in the next record

1 rspher. The radii of the atomic spheres. By default they are determined for each type by *a)* the distance d_0 to the nearest atom and *b)* the nuclear charge *Z*, such that
   a rspher$<d_0$ /2: the atomic sphere is inside the atomic voronoi polyhedron.
   b if d_0 is small: rspher$\approx d_0$ /2: the sphere is as large as possible in narrow regions.
   c if d_0 is large: rspher$\approx d_0$ /2: for free, isolated atoms integration in spherical coordinates is optimal, so we extend the sphere as much as possible in such a situation. 'Very large' is defined to be frange, the assumed function range, so rspher depends also on this parameter.
   d for 'normal' values of d_0 : rspher equals approximately a standard value that depends on the nuclear charge: heavier atoms get a larger 'core'-sphere. The implemented function can be found in RDINTG. It results for instance in rspher=0.5 for hydrogen, 1.0 for oxygen, 1.3 for copper and 1.7 for uranium. (In fact a smooth function has been devised that produces precisely these values). To get an impression of the combined effect of all aspects fig.6 displays the value of rspher as a function of d_0 for copper, with frange=25 a.u.

2 linteg: the maximum angular momentum quantum number of one-center integrands, one value for each type of atom. (Lintgx, used to determine the default value of linrot above, is the maximum over this array). BAND automatically computes these values from the fit functions and basis functions employed in the calculation. Lintgx is used by POINTS to choose the appropriate angular integration formulas for the atomic spheres.
The default value for linteg as implemented in RDINTG depends on the nuclear charge: linteg=0 ($Z\leq 2$), 4 ($Z\leq 18$), 8 ($Z\leq 54$) or 12. This is not relevant here because BAND overrules the default values anyway.

remarks

The defaults stated above might yield evidently ridiculous values in some situations. For instance dishul (=2.3×rsphx) would become unreasonably small if we explicitly specify extreme values for the atomic spheres. This is accounted for in RDINTG by checking such extraordinary situations and changing the default to more sensible values. The details of these checks and adaptations can be found by inspection of RDINTG. We have attempted to provide reasonable defaults for all parameters, whatever the specifications for the others.

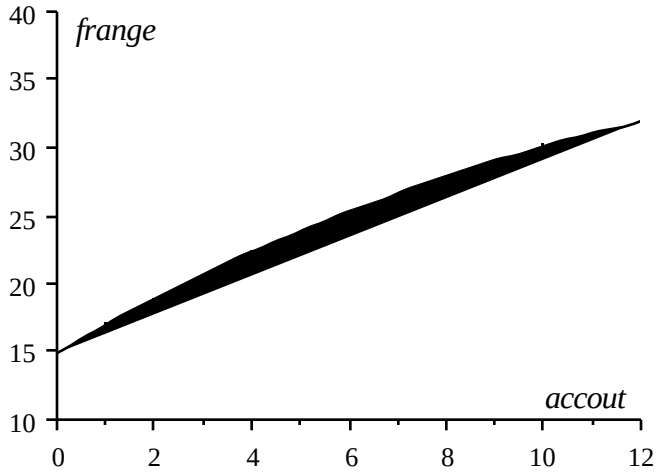

Fig.5. Assumed function-range (frange), in a.u., depending on the integration parameter accout, for Z_x =30. (see text)

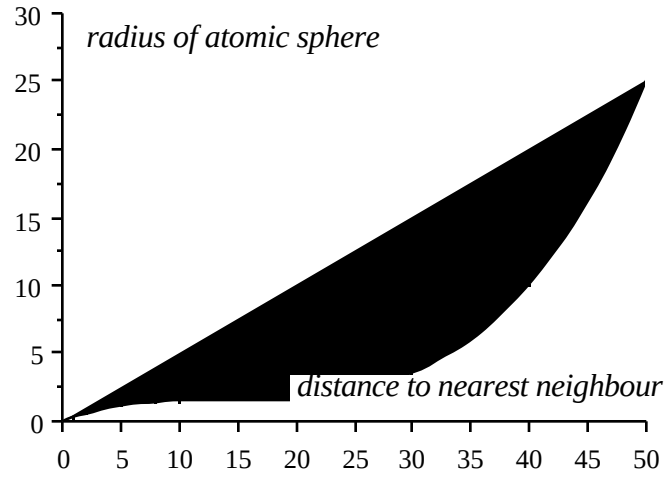

Fig.6. Radius of the atomic sphere (a.u.) for Cu, as a function of the distance d_0 (a.u.) to the nearest atom. (assumed function range: 25 a.u. (see text))

As mentioned before, the treatment of the outer region is not so well developed (yet) as the atomic spheres and polyhedra. Although this is usually not a problem due to the relative unimportance of that part of space and because the implemented strategy functions reasonably well in practice, we have encountered (minor) difficulties with some molecules. Invariably this had to do with the 'parallel' integration (parameter outpar) over large polygons that were part of the enveloping molecular polyhedron. These polygons are subdivided in quadrangles (and possibly a triangle). No further subdivision is made. Consequently we

may occasionally have to integrate over very large quadrangles with one single product Gauss-Legendre formula. Functions that are relatively localized are not integrated easily then and a subdivision in smaller quadrangles would probably be better.

When such troubles come up one may specify a larger value for dishul, thereby shifting the problematic region away and making it less important, or one may increase outpar. It is not easy to say which of the two leads effectively to more integration points. Increasing dishul implies enlarging some of the atomic polyhedra; this results usually in significantly more points in the involved pyramids. At the other hand, if the outer polygons are essentially too large for efficient integration, outpar will have to be increased considerably before the problem is solved.

15 Interpolation and bloch sums

The construction of crystal basis functions involves interpolation and bloch summation. The crystal functions are represented by their values in the integration points and they are computed (in most cases) as bloch sums of one center functions

$$f(\boldsymbol{k};\boldsymbol{r}) = \sum_{\boldsymbol{R}} e^{i\boldsymbol{k}\cdot\boldsymbol{R}}\ \chi(\boldsymbol{r}-\boldsymbol{R}) \tag{15.1}$$

The summation over the lattice points, or cells $\boldsymbol{R}$ is infinite but for the value of $f(\boldsymbol{k};\boldsymbol{r})$ in a point $\boldsymbol{r}$ in the central unit cell the loop may be cut off at some large cell distance because the employed one-center functions $\chi(\boldsymbol{r})$ decay exponentially for large r.

The functions $\chi(\boldsymbol{r})$ are of the form $Z_{lm}(\Omega_\alpha)\,P(r_\alpha)$, consisting of a spherical harmonic times a radial function in coordinates relative to some atom α. The radial functions are in BAND represented as tables, $P(r)\equiv\{P(r_i),i=1,..,nr\}$.

interpolation

To compute the sum (15.1) we must in particular determine from the table the radial function value $P(r)$ for any distance r. This is achieved by a three-point Lagrange interpolation: given r we determine the three nearest points in the radial grid, r_i , r_{i+1} and r_{i+2} . $P(r)$ is then computed as a linear combination

$$P(r) = e_1\,P_i\ + e_2\,P_{i+1}\ + e_3\,P_{i+2} \tag{15.2}$$

such that $P(r)$ is implicitly approximated by a parabola through the three points.

For each occurring distance r we have to determine the index i in the radial table and then the combination coefficients e_1 , e_2 and e_3 . Let c be the multiplication constant that characterizes the radial mesh, $c\equiv r_{j+1}/r_j$. The interpolated value is given by the Lagrange formula

$$P(r) = \frac{(r-r_{i+1})(r-r_{i+2})}{(r_i-r_{i+1})(r_i-r_{i+2})}\ P_i\ + \frac{(r-r_i)(r-r_{i+2})}{(r_{i+1}-r_i)(r_{i+1}-r_{i+2})}\ P_{i+1}\ + \frac{(r-r_i)(r-r_{i+1})}{(r_{i+2}-r_i)(r_{i+2}-r_{i+1})}\ P_{i+2}\ =$$

(define $b=r/r_i$)

$$= \frac{(b-c)(b-c^2)}{(1-c)(1-c^2)}\ P_i\ + \frac{(b-1)(b-c^2)}{(c-1)(c-c^2)}\ P_{i+1}\ + \frac{(b-1)(b-c)}{(c^2-1)(c^2-c)})\ P_{i+2}\ =$$

$$= d_1\,(b-c)(b-c^2)\ P_i\ + d_2\,(b-1)(b-c^2)\ P_{i+1}\ + d_3\,(b-1)(b-c)\ P_{i+2} \tag{15.3}$$

d_1 , d_2 and d_3 are constants of the radial mesh: d_1 =1/{(1−c)(1−c^2)}, d_2 =1/{(c−1)(c−c^2)}, d_3 =1/{(c^2 −1)(c^2 −c)}. The interpolation coefficients in (15.2) are thus given by

$$e_1 = d_1\,(b-c)(b-c^2)$$

$$e_2 = d_2\,(b-1)(b-c^2) \qquad (15.4)$$

$$e_3 = d_3\,(b-1)(b-c)$$

Of course r may be so large that we need *extra*polation from the table. This should make no difference. The tables are expected to contain the whole range of the function so that the last few values P_{nr} , P_{nr-1} ,.. are negligible and the result is almost zero. Polynomial extrapolation of an exponentially decaying function is in principle hazardous however and we set the coefficients explicitly to zero as a safety measure.

The radial grid points are $r_i = r_1\, c^{i-1}$. If r_{i+1} is the grid point closest to r, the index i is computed as

$$i = \text{nint}\left(\frac{\log(r/r_1)}{\log(c)}\right) \qquad (15.5)$$

Nint(x) is the integer nearest to x. The index i to be used is of course constrained by 1≤i≤nr−2.

implementation

Radial functions are interpolated in ATMFNC, BASPNT and FITPNT. The organization is in all three routines analogous. The radial mesh points (that is, their reciprocals) are stored in array rinv. For each type of atom subroutine INTDAT provides various constants related to the radial mesh: d1, d2 and d3, c and csq(=c^2), nr2(=nr−2), rfac(=1/r_1 =rinv(1)) and clogi(=1/log(c)) (c.f. (15.4) and (15.5)).

Then we compute for a vector of integration points, with the coordinates relative to some atom α in some cell $\boldsymbol{R}$, the interpolation coefficients for each point as follows:

1. determine the radial distance $r_{\alpha,\boldsymbol{R}}$ and (except in ATMFNC) also the values of the spherical harmonics.
2. get index i: intpl=nint {clogi×log(r×rfac)}
3. apply the constraints: intpl=max(intpl,1) and intpl=min(intpl,nr2) (routine VBND2I).
4. b=r×rinv(intpl)
5. determine the coefficients from (15.4)
6. no extrapolation: if b>csq, set e_1 =e_2 =e_3 =0 (routine CONDIT).

Some of these operations have been isolated in separate routines (CONDIT, VBND2I) because the code appeared to be not so easily vectorized (by the Cy205 compiler) in the original context.

accuracy and normalization

Interpolation implicates obviously some inaccuracy, which depends on the denseness of the radial mesh. We have tested this by interpolating exponential functions $r^n\, e^{-\alpha r}$ and comparing the resulting integrals with the analytical values; a numerical integration scheme with very high precision has been used for this, so that the errors are determined by the interpolation. Fixing the first and last mesh-points at 10^{-4} and 40 a.u.

(the defaults in BAND) we found relative errors of the order 2×10^{-5} for 300 mesh points, 1.5×10^{-6} for 600 points, 2×10^{-7} with 1200 points and 3×10^{-8} using 2400 points. These figures are the r.m.s. errors found for the employed set of test functions; the worst cases deviated less than an order of magnitude. The default used in BAND, 2000 points, may thus be expected to give no significant interpolation errors. Application of a *four*-point interpolation did not yield a substantial improvement. We have not tried a five-point interpolation; the interpolation routines (especially BASPNT) take a major part of the execution time (in the preparation stage of the program) and the use of five instead of three interpolation coefficients would considerably increase the cost.

Of some functions the exact integral is known. This holds for instance for various functions of the numerical free atoms: the charge density, the potential, the individual orbital densities and the corresponding kinetic energy functions. Their integrals, as evaluated numerically over the radial logarithmic grid, are so accurate that they may be called exact. The numerical integral over the crystal grid gives an indication of the interpolation accuracy, but is also determined of course by the quality of the crystal grid itself. One may now apply some normalization to correct the deviations, by simply multiplying the interpolated function values by a constant. We have experimented with this possibility but found that it is better not to do so: the total energy of the crystal and the cohesive energy were less stable against variations in the integration precision and hence, on the average, their deviations from the exact (converged) result were larger. We have applied the normalization to several combinations of the obvious candidates (only the density, the density and the kinetic energy, the potential, and so on); in none case we found an improvement.

bloch sums

To limit the number of terms in the bloch sums (15.1) all radial function tables are analyzed; for each function the minimum number of cells is determined to evaluate (15.1) with sufficient precision. First RADMAX calculates the maximum radial extension of any of the tabulated functions. The computed maximum cell-distance needed in any bloch sum, rcelx, is then used by CELLS to generate a list of ncel lattice points, xyzcel(3,ncel), ordered according to their distances from the origin. Finally CELMAX re-examines the radial tables and copies them to another file together with the maximum number of cells to be used for each of them individually.
RADMAX and CELMAX employ the same auxiliary routine RADMAA for the analysis of a single table: RADMAA takes the absolute values of the function and integrates then with a repeated simpson rule (SIMPSL) over a trial subrange 1..ntry of the nr points. This is compared with the total integral and the minimum value of ntry is determined (by bisection) where the missing part represents less than a fraction cutoff.
The criterium for negligibility cutoff is $10^{-\max(3.0,\ \mathrm{accint}+1.0)}$ (computed in RADIAL) ; accint is the general accuracy parameter for integration over the crystal unit cell (SS^integration). cutoff is optionally read from input (key: `cut off`; the second default 10^{-2} leads to a rather crude approximation of the bloch sums).

remarks

1. The generated list of 'cells' is usually larger than according to this section. The reason is that the lattice summation of the multipole potentials in VMULTI requires more terms and employs the same list xyzcel(). Rcelx is therefore determined also by the parameters rmadel and dmadel used in the evaluation of these potentials (SS^coulomb potential).
2. Rcelx may be read from input with key `rcel` or `cell distance`.

16 Iteration

The crystal hamiltonian equation is solved by an iterative procedure. Self consistency may be checked by monitoring in subsequent cycles for instance the total energy, the electronic charge density or the potential. BAND uses the potential.

Define the cycle operator F. It comprises one complete cycle, in which the potential V leads, via diagonalization of the hamiltonian and construction of the resulting density to a computed new potential $F(V)$.

$$F: V \quad H \quad \{e,\psi\} \quad \rho \quad F(V) \tag{16.1}$$

In the self consistent situation $F(V)=V$. Starting from some trial V_0 we obtain $F(V_0)$ and may insert this into the next cycle as the new potential V_1. The sequence V_k, k=1,2,.. may then be hoped to converge, but in many cases it displays oscillatory behaviour.

optimized damping

BAND solves this by damping. The potential used in the next cycle is a mixture of the previous and the computed one:

$$V_k = (1-\gamma)V_{k-1} + \gamma F(V_{k-1}) \equiv H_\gamma(V_{k-1}) \tag{16.2}$$

where we introduced the operator $H_\gamma = (1-\gamma)\mathbf{1} + \gamma F$.
In virtually all cases the oscillations are suppressed and convergence of the sequence $\{V_k\}$ is achieved if the mixing parameter γ is small enough. Of course too small a value for γ slows down the development of V towards self consistency and one would like to use some optimum value γ_{opt}.

It turns out that γ_{opt} not only varies from one atomic system to another, but also that it may depend on the stage of the iterative proces. Often it is necessary to use stronger damping, by diminishing γ, as self consistency is approached. BAND tries to optimize γ from cycle to cycle. The way in which this is done is based on a discussion of self consistency strategies in [Marchuk 1975].
Define by $G \equiv \mathbf{1} - F$ the operator that yields the difference between input and output of a cycle.

$$G(V) = V - F(V) \tag{16.3}$$

The self consistent solution V^* to be found satisfies $G(V^*)=0$. We assume now that F and G can be approximated by linear operators with real eigenvalues. Let then $\{u_n, \mu_n\}$, n=0,1,.. be the eigensystem of F

$$F(u_n) = \mu_n u_n \tag{16.4}$$

with $u_0 \sim V^*$, $\mu_0 = 1$.
A proper self consistent V^* must at least be locally stable, so that $\mu_n < 1$, n=1,2,.. [Dederichs and Zeller 1983] The related eigensystem of G, $\{u_n, \lambda_n \equiv 1-\mu_n\}$ thus satisfies $\lambda_0 = 0$, $\lambda_n > 0$, n=1,2,...
If V has the expansion

$$V = V^* + \sum_{n=1} c_n u_n \tag{16.5}$$

then

$$F(V) = V^* + \sum_{n=1} \mu_n c_n u_n \tag{16.6}$$

The (local) stability of the self-consistent solution implies $\mu_n < 1$, but not $|\mu_n| < 1$, so it is not assured that the sequence $F(V)$, $F(F(V))$,.. should converge to V^*.

Consider now simple damping and define (16.2):

$$H_\gamma(V) = V - \gamma G(V) = V^* + \sum_{n=1} (1-\gamma\lambda_n) c_n u_n \tag{16.7}$$

The convergence rate of the n-th component is $\eta_n \equiv (1-\gamma\lambda_n)$. The sequence $H_\gamma(V)$, $H_\gamma(H_\gamma(V))$,.. converges if $|\eta_n| < 1$ for all n.
The (asymptotic) rate of convergence is determined by the most slowly decreasing component, i.e. by $\max_n |\eta_n|$. Denote the smallest and largest λ by α and β respectively, $0 < \alpha \le \lambda_n \le \beta$, n=1,2,.. and the corresponding (normalized) eigenvectors by $|\alpha\rangle$ and $|\beta\rangle$. Best overall convergence is achieved with

$$\gamma_{opt} = 2/(\beta+\alpha) \tag{16.8}$$

For $|\alpha\rangle$ and $|\beta\rangle$ this gives an absolute rate of convergence

$$|1-\gamma_{opt}\alpha| = |1-\gamma_{opt}\beta| = (\beta-\alpha)/(\beta+\alpha) \tag{16.9}$$

and all other components decrease faster. With optimal damping the deviation of V_k from V^* becomes more and more dominated by $|\alpha\rangle$ and $|\beta\rangle$ as the iterations progress. Let us assume then for simplicity

$$V_k = V^* + c_\alpha |\alpha\rangle + c_\beta |\beta\rangle \tag{16.10}$$

so that

$$g_k \equiv G(V_k) = \alpha c_\alpha \hat{o}\alpha + \beta c_\beta \hat{o}\beta \equiv A\hat{o}\alpha + B\hat{o}\beta \tag{16.11}$$

This gives at the next cycle

$$V_{k+1} = V_k - \gamma g_k = V^* + (1-\gamma\alpha)c_\alpha \hat{o}\alpha + (1-\gamma\beta)c_\beta \hat{o}\beta =$$

$$= V^* + \eta_\alpha c_\alpha \hat{o}\alpha + \eta_\beta c_\beta \hat{o}\beta \tag{16.12}$$

$$g_{k+1} = \eta_\alpha A\hat{o}\alpha + \eta_\beta B\hat{o}\beta \tag{16.13}$$

In optimal damping $\gamma=2/(\beta+\alpha)$, so that $\eta_\alpha = -\eta_\beta$. For all reasonable values of γ we have at least $0 \le \eta_\alpha < 1$ and $-1 < \eta_\beta \le 0$: both components decrease and the β-component oscillates. fig.7 displays g_k and g_{k+1} as vectors in the two-dimensional space spanned by $\hat{o}\alpha$ and $\hat{o}\beta$. The coordinate axes are depicted oblique to stress that $\hat{o}\alpha$ and $\hat{o}\beta$ are not necessarily orthogonal.

Our main purpose now is to achieve optimal damping. In that case the angle θ between successive vectors g_k is a constant, independent of k, and g_k and g_{k+2} are parallel. The potential V and hence g are available by their values in the integration points. The norm $\hat{o}g_k\hat{o}$ and the inner product $(g_k \cdot g_{k+1})$ are thus easily computed and defne the angle θ

$$\cos\theta_k = \frac{(g_{k-1} \cdot g_k)}{\hat{o}g_{k-1}\hat{o}\ \hat{o}g_k\hat{o}} \tag{16.14}$$

If A and B of (16.11) and (16.13) differ very much in size $\cos\theta_k \approx \pm 1$ and a change in θ may be computed less accurately. Therefore we strive to have $\hat{o}A\hat{o} = \hat{o}B\hat{o}$ as a secondary goal, together with $\gamma = \gamma_{opt}$. In the optimal situation that $\hat{o}A\hat{o} = \hat{o}B\hat{o}$ and $\gamma = \gamma_{opt}$ the vectors g_{k-1} and g_k are orthogonal, $\cos\theta = 0$. The overall strategy, combining the two goals, is to let $\cos\theta$ go slowly to zero in the course of the iterations. The *smallness* of the changes in θ assures that $\gamma \approx \gamma_{opt}$; the *direction* of the changes corresponds to making the two components equal.

At every cycle the angle θ_k is calculated and compared with θ_{k-1}. The development in the angle tells us whether γ is smaller or larger than γ_{opt}, while the value of θ itself indicates the relative sizes of the α- and β-components; γ is then adapted accordingly.

This set-up is fairly straightforward but contains a few problems. In the first place we have to determine *by how much* γ should increase or decrease. Secondly we have made some strong assumptions. If they were exact it would be possible to determine the vectors $\hat{o}\alpha$ and $\hat{o}\beta$ and their coefficients A and B from a few consecutive 'measurements' g_k, g_{k+1},... We could then compute V^* exactly. In reality however other components than $\hat{o}\alpha$ and $\hat{o}\beta$ are also present. Furthermore the true operator G is not linear. The linearized form represents a first order approximation in the neighbourhood of the current potential V. Consequently the apparent eigensystem $\{u_n, \lambda_n\}$ varies from cycle to cycle as V changes. The angle θ_k and its relation to θ_{k-1} are then not so simply related to the mixing parameter γ as we have assumed above.

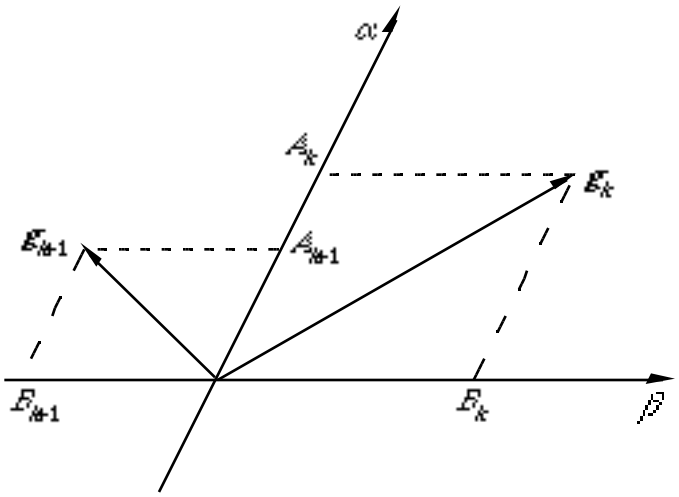

Fig.7. Two consecutive difference vectors ***g*** *and their components along the* α*- and* β*- axis (see text).*

In the past years we have tried in several ways to use various assumptions more rigorously. Again and again this has yielded spectacular convergence in some cases and divergence or oscillations in others. The strategy implemented in BAND employs the stated assumptions cautiously and has proven to be robust. Convergence is reached almost always, though it may occasionally take a large number of cycles, up to a few hundred. Typically 30 cycles suffice.

implementation

Subroutine RHOPOT computes the new potential $F(V)$ from the density and calls MIXITR, which determines the new mixing parameter. The variables that are relevant for monitoring the development of γ and θ are in common block VARDAT. At every cycle the previous potential V_{k-1} and the corresponding difference vector g_{k-1} are on file itpot. Together with the computed mixing parameter, parmix (γ), they define the current potential: $V_k = V_{k-1} - \gamma g_{k-1}$.

To compute a next value of γ it is assumed that it lies between a lower bound γ_l and an upperbound γ_u, the variables parmxl and parmxu in the program. To accomodate to changes in the apparent eigensystem of G, γ_l and γ_u are updated together with γ itself. The interval (γ_l, γ_u) is intended to indicate something like an error bar on γ itself and is used to scale the amount of change in γ. The interval becomes smaller when γ is more or less stable over several cycles and it is enlarged when the adaptations to γ appear to be insufficient.

To monitor the development of the angle θ_k between the vectors g_{k-1} and g_k MIXITR employs an 'error' function ε_k , which is sensitive in the whole domain of θ, in particular near the undesirable extreme values 0 and π, which varies from $-\infty$ to $+\infty$, equalling zero in the optimum $\theta=\pi/2$.

$$\varepsilon_k = \mathrm{tg}(\theta_k/2) - \frac{1}{\mathrm{tg}(\theta_k/2)} \qquad (16.15)$$

ε_k , rather than $\cos\theta_k$ is checked and guided to zero. The optimal development is defined to be $\varepsilon_k = 0.95\varepsilon_{k-1}$. The relative deviation from this is

$$\delta = \frac{\varepsilon_k - 0.95\varepsilon_{k-1}}{\| \varepsilon_k \|} \tag{16.16}$$

γ is increased if $\delta<0$ and diminished otherwise. The size of δ is used to compute the new γ in the appropriate interval $(\gamma_{previous}, \gamma_u)$ or $(\gamma_l, \gamma_{previous})$ respectively. The map $d\gamma(\delta)$ is chosen as a fermi-distribution function: approximately linear dependency on δ around the cut-off point (δ=0) and an exponential approach to the limit γ_l (or γ_u) for large $\| \delta \|$.

The update of the boundaries is as follows. If γ increases the lower boundary γ_l is increased somewhat and the upper boundary is shifted upward; this shift is larger when *a)* the last few changes in γ have been in the same direction and/or *b)* the previous increase of γ was relatively large. Both cases suggest that the upper bound has been too close and consequently the increments of γ too small. A variable, pmixch, keeps track of the number of consecutive changes of γ in the same direction.
If γ diminishes the boundaries are updated in a similar way, but the increasing and decreasing cases are not treated on the same footing. The reason is that a too large γ may easily lead to a (temporary) divergence in the SCF procedure, while a too small value only slows down convergence (16.7). Therefore the implemented adaptations are such, that a speed-up in boundary adaptations is triggered more easily for *de*creases than for *in*creases of γ.

The norms $\| g_k \|$ are used to check convergence. The current value, potdif, is compared in routine SCFTST with the convergence criterium, convrg. The quotient of successive values of potdif defines the *rate* of convergence. From fig.7 it can be inferred that this quotient will oscillate if the 'coordinate' axes are oblique (assuming optimal damping). Such an oscillatory trend in the convergence rate is encountered fairly often. The situation is then better judged by considering the development over two cycles at a time; we do this by taking the harmonic average of two successive quotients. Conv0 is the current quotient, convr0 (in common VARDAT) that of the previous cycle, and averag≡(conv0×convr0)$^{1/2}$ the quantity used. In order to measure the overall development, neglecting moderate variations from one cycle to another, the convergence rate convrt is defined in BAND as a weighted (harmonic) mean of averag-values of subsequent cycles:

$$\text{convrt} \leftarrow \text{convrt}^{0.85} \times \text{averag}^{0.15} \tag{16.17}$$

Convrt is initialized in INIT (0.5) and checked at every cycle in SCFTST by comparison with a criterium scfrtx. It may happen that the automatic adaptations fail, in the sense that the SCF procedure converges too slowly (or not at all): convrt exceeds scfrtx. This may be due to some weak point in the optimization algorithm or to problematic aspects in the atomic system at hand. If for instance many bands are close to the fermi level, previously unoccupied bands may drop below occupied ones as a consequence of the changes in the potential. The occupation numbers are determined according to the aufbau-principle so that abrupt variations in the electronic configuration may result with corresponding large changes in the computed $F(V)$ and hence in the difference vector g. When convergence seems to fail, SCFTST attempts to recover by

halving γ abruptly; the resulting slowing down of the iterative potential updates has proved to solve the convergence problems in the large majority of cases.
At such an intervention by SCFTST, the boundary values γ_l and γ_u are adapted accordingly; convrt is re-initialized at 0.5 and scfrtx is slightly increased to allow slower convergence in the continuation.
This trick is repeated until it is concluded that convergence is still failing. This is judged by comparing the values of potdif at the successive interventions by SCFTST; if *this* sequence does not converge the program is stopped; the last value in that sequence is stored in the variable difold.

γ as well as the boundaries γ_l and γ_u are allowed to assume values between zero and some maximum, parmax. parmax is initialized at 2.0, but it is decreased by SCFTST when convergence is problematic.
INIT initializes parmix (0.15), parmxl (0.10) and parmxu (0.20).
Via input various aspects of the iterative procedure may be influenced, by using the following keys:

`mix`, or `damp` specifies the initial value of parmix. If the key record contains a second real value, this is assigned to a factor delmix (default: 0.75). The boundary values parmxl and parmxu are initialized at parmix×delmix and parmix/delmix.

`converg` overrules the default of the absolute convergence criterium convrg.

`scfrtx` supplies the slowest allowable rate of convergence scfrtx.

`cycle` specifies the maximum number of cycles to be executed. For organizational reasons at least two cycles will be executed. If maximally one cycle is ordered, then parmix is set at zero, so that the final results correspond to the starting-up potential, without any update.

Chebyshev acceleration

Strategies for iterative processes are discussed frequently in the literature. A few words may be spend here on a method, called the Chebyshev acceleration [Marchuk 1975]. It may provide a significant improvement over damping. Terminology and definitions are as above. In particular it is assumed again that the operator $G=\mathbf{1}-F$ is linear with real eigenvalues.
In damping V_k is a linear combination of $F(V_{k-1})$ and V_{k-1}. The Chebyshev method is based on a straightforward generalization of this: V_k is a linear combination of $F(V_{k-1})$ and *all* previous V_m, $m=k-1,k-2,...$ It will turn out that it is not necessary to store all these previous potentials.
Let V_0 be some trial potential and let $P_k(x)$ be a polynomial of degree k in the variable x. Define the sequence $\{V_k\}$ by

$$V_k = P_k(G)\ V_0 \tag{16.18}$$

With the expansion (16.5) for V_0 this gives

$$V_k = P_k(G) \sum_{n=0} c_n u_n = \sum_{n=0} P_k(\lambda_n)\ c_n\ u_n \tag{16.19}$$

with $c_0 u_0 = V^*$. Two conditions are imposed on the polynomials P_k :

a) $P_k(G)\,V^*=V^* \rightarrow P_k(0)=1$. This is a normalization condition; it is analogous to the coefficients of $F(V_{k-1})$ and V_{k-1} adding up to unity in (16.2).

b) $P_k(\lambda)$ is as small as possible for all λ in the spectrum of G. This states that any component in $V-V^*$ should be made as small as possible (16.19). Since we do not know all (discrete) λ_n, this condition is generalized to the whole interval (α,β)

$$\max_{\alpha\le\lambda\le\beta} P_k(\lambda) \text{ is minimal} \tag{16.20}$$

The Chebyshev polynomials of the first kind have the minimax property, so the conditions *a* and *b* are satisfied by

$$P_k(\lambda) = \frac{T_k(\frac{\beta+\alpha-2\lambda}{\beta-\alpha})}{T_k(\frac{\beta+\alpha}{\beta-\alpha})} \tag{16.21}$$

where $T_k(x)$ is the Chebyshev polynomial (of the first kind).

The recurrence relation for Chebyshev polynomials, $T_k(x)=2xT_{k-1}(x)-T_{k-2}(x)$, can be used to obtain a convenient expression for V_k from (16.19) and (16.21). Define

$$x=\frac{\beta+\alpha-2G}{\beta-\alpha} \qquad z=\frac{\beta+\alpha}{\beta-\alpha} \tag{16.22}$$

This gives

$$V_k = P_k(G)\;V_0 = \frac{T_k(x)\;V_0}{T_k(z)} = \frac{2zT_{k-1}(x)V_0 - T_{k-2}(x)V_0 - \frac{4}{\beta-\alpha}GT_{k-1}(x)V_0}{T_k(z)} =$$

$$= \frac{2zT_{k-1}(z)P_{k-1}(G)V_0 - T_{k-2}(z)P_{k-2}(G)V_0 - \frac{4}{\beta-\alpha}T_{k-1}(z)GP_{k-1}(G)V_0}{T_k(z)} =$$

$$= \frac{2zT_{k-1}(z)V_{k-1} - T_{k-2}(z)V_{k-2} - \frac{4}{\beta-\alpha}T_{k-1}(z)g_{k-1}}{T_k(z)} \tag{16.23}$$

Equation (16.23) shows that the Chebyshev method is in principle hardly more complicated than damping. We have to store V_{k-2} on file, in addition to V_{k-1}. In damping we must determine one parameter, γ; here we need two parameters, α and β. If these are known the numbers $T_k(z)$, k=1,2,... are computable with the recurrence relation.

The iterative procedure is started with some trial potential V_0. To use (16.23) also for the first cycle, define

$$T_{-1}(z) = T_1(z) = z \tag{16.24}$$

(satisfying the recurrence relation) and $V_{-1}=V_1$. For k=1 we obtain then

$$V_1 = V_0 - \frac{2}{\beta+\alpha} g_0 \qquad (16.25)$$

i.e.: optimal damping at the first cycle.

Compared with damping the Chebyshev procedure gives theoretically a considerably faster convergence. In optimal damping the convergence rate is given by (16.9): $\eta^{damp} = 1/z$. In the Chebyshev method the components decrease as $P_k(\lambda_n)$. Since $T_k(x)$ is a periodic function of k for all $|x| \leq 1$, all components in (16.19) decrease on the average as $1/T_k(z)$, giving for the convergence rate

$$\eta^{cheb} = T_{k-1}(z) \;/\; T_k(z) \qquad (16.26)$$

For large $z>>1$ $T_k(z)$ is dominated by the leading term $2^{k-1} z^k$, making $\eta^{cheb} \approx 1/(2z)$. The gain over damping is a factor two per cycle.
For z close to unity ($\beta>>\alpha$), say $z=1+\delta$ with $\delta<<1$, damping converges very slowly: $\eta^{damp} \approx 1-\delta$. In the Chebyshev method convergence is not very fast either in the first few cycles, but significantly better than with damping: in a first order Taylor expansion $T_k(1+\delta) \approx 1+\delta k^2$ ($\delta k^2<<1$) giving $\eta \approx 1-\delta(2k-1)$. For larger k the variation in η from one cycle to another will diminish. Assuming for simplicity that it is constant, the recurrence relation can be used to compute

$$\eta = \frac{T_{k-1}(z)}{T_k(z)} = \frac{1}{2z - \dfrac{T_{k-2}(z)}{T_{k-1}(z)}} = \frac{1}{(2z-\eta)} \qquad \eta \approx 1 - (2\delta)^{1/2} \qquad (16.27)$$

So, if for instance $\delta=0.1$, damping needs 25 cycles to proceed one order of magnitude, while this is achieved in 4 cycles with the Chebyshev method (for large k); if $\delta=0.01$ these figures become 230 and 15 cycles respectively.

All this presupposes of course that the assumptions are correct: that G is a linear operator and that α and β are known, at least approximately. A useful implementation will in particular require some way to adapt the procedure to changes in the apparent spectrum of G, i.e. to update α and β iteratively and to incorporate such changes in the method.

Impressive results are reported by Akai and Dederichs [1985], but the presented data are very few, making one suspicious that these may correspond only to non-problematic cases. We have done some experimentation in a simple Hartree-Fock program, in calculations on small first-row molecules. With rigid, fixed values for α and β, the results varied from reasonable improvement over damping to problematic oscillatory behaviour. In all cases it proved advantageous to use simple damping in the first few cycles, before starting up the Chebyshev method; the same experience was mentioned in [Akai and Dederichs 1985]. We have thusfar not worked out a practical implementation with control of the developments and provisions for problems that might occur.

17 Lattice vectors

LATTYP reads the lattice vectors from itinpt into array avec(3,3). The inverse-transpose, bvec(3,3) describes the Bravais lattice in *k*-space (apart from a scaling factor 2π). The dimensionality in real space ndim is the number of lattice vectors found in input. Ndimk is the dimensionality in *k*-space. By default ndimk=ndim, but a lower value can be requested with the key `suppress`. This specifies the number of directions in *k*-space for which the dispersion is to be suppressed; the second default, induced by giving only the key `suppress` without a numerical value in the same record, is: ndimk=0, i.e. *all* dispersion is neglected; only the Γ-point $\boldsymbol{K}=0$ is used then.

In one special case the program overrules whatever the input specifications imply by setting ndimk=0, when each atom in the central unit cell is (or can be chosen to be) separated so far from any atom in a neighbouring cell that the functions have no overlap. Effectively we have then in each unit cell a 'molecule' isolated from all other atoms in the system. This is decided by ATMSEP. ATMSEP translates first by Bravais translations all atoms in the central cell in such a way that they from a maximally compact group. Then the minimum distance to any other atom is calculated, which is then compared with $\frac{3}{2}\times$rfar; rfar is the maximum extension of any of the one-center functions in the crystal (not counting the multipole potentials of course); rfar is calculated by RADMAX.

If dispersion is to be neglected in some but not all directions in *k*-space we have to determine which of the ndim original vectors in bvec are discarded. BAND chooses for this the shortest of the lattice vector(s) that span the BZ.

Orientation

In many places in the program it is convenient to have the lattice vectors oriented such that only the first ndim, respectively ndimk cartesian coordinates are needed to describe the Bravais lattices. As the input lattice vectors are not subject to any condition of this kind BAND performs a rotation to a standard orientation. The transformation matrix stdrot(3,3) is computed in LATTIC. The input-specified atom coordinates are interpreted in the original coordinate system; they are rotated in GEMTRY to the new frame.

18 Legendre points

Points and weights for Gauss-Legendre quadrature are generated by routines LEGPNT(n,x,w) and LEGEND(n,x,w,a,b).

LEGPNT is the basic routine. Input is *n*, the number of requested points; output are the *n* zeros x(i),*i*=1,*n* of the *n*-th order Legendre polynomial and the associated weights w(i),*i*=1,*n*. The numerical integration with this scheme is exact for polynomials of degree ≤2*n*−1 [Krylov 1962]. The points are located on (−1,1). The integration scheme can be used for any finite interval by a linear transformation; the algebraic degree of precision is then preserved. LEGEND calls LEGPNT to obtain the points on (−1,1) and transforms them to (a,b)

$$x_i \leftarrow a + 2\,\frac{x_i+1}{b-a}$$
$$\mathrm{w_i} \leftarrow \frac{2w_i}{b-a} \qquad (18.1)$$

computation of the points and weights

Let $P_n(x)$ be the Legendre polynomial of degree *n*, with zeros x_i , *i*=1,*n*.

For each zero a first guess x is made with a distribution function (see below) for the zeros. This guess is improved iteratively by the Newton-Raphson method (i.e. first order Taylor expansion) until convergence is reached

$$x \leftarrow x - \frac{P_n(x)}{P_n'(x)} \qquad (18.2)$$

$P_n(x)$ and the derivative $P_n'(x)$ are evaluated with the recurrence formulas for Legendre polynomials [Arfken 1970] ($P_0(x)$ =1, $P_1(x)$ =*x*)

$$P_n(x) = \frac{1}{n}\left\{(2n-1)xP_{n-1}(x) - (n-1)P_{n-2}(x)\right\} \qquad n=2,3,.. \qquad (18.3)$$

$$P_n'(x) = \frac{n}{1-x^2}\left\{P_{n-1}(x) - xP_n(x)\right\} \qquad (18.4)$$

If *x* has converged to a zero, the associated weight can be computed from the relation [Krylov 1962]

$$w = \frac{2}{(1-x^2)\,(P_n'(x))^2} \qquad (18.5)$$

Using recurrence relations for the Legendre polynomials this can be rewritten in numerous different forms. Since *x* is only approximately the zero the weight is also only approximately the exact integration weight. The different possible formulas for the weights result in significantly different deviations from the correct value. The reason for this is that terms involving $P_n(x)$ are omitted in the analytical forms but they are not strictly zero in the computation. Trial and error taught us to adopt the formula (from 18.4 and 18.5)

$$w = \frac{1-x^2}{n^2\,(P_{n-1}(x) - xP_n(x))^2} \qquad (18.6)$$

Omission of the second term in the numerator, analytically zero, deteriorates the result.

the distribution function

The zeros lie symmetrically around x=0. The distribution function gives the approximate location of the zeros in (0,1). This approximation is

$$x_i \approx \sin\left\{\frac{\pi}{2}\,\frac{(n+1-2i)}{[(n+1/2+1)/(8\,f(n,i))]}\right\} \qquad i=1..n/2 \tag{18.7a}$$

$$f(n,i) = 1/2 + n\cos\left\{\frac{\pi}{2}\,\frac{(n+1-2i)}{(n+1/2)}\right\} \tag{18.7b}$$

remarks

The Newton-Raphson procedure is only guaranteed to converge (to the intended zero) if the initial approximation is accurate enough. The approximation function (18.7) is purely empirical; we have not tried to derive a formal proof of the adequacy. Tests with $n \le 10{,}000$ gave no problems; they indicate even that the approximation is better for higher n.

We have compared our results with the tables of Stroud and Secrest [1966] for n up to 1,000. For the higher n-values the weights tend to be less precise than the points, presumably because the weights are related to the derivative (18.5). For higher n the polynomial $P_n(x)$ oscillates rapidly, especially near the endpoints x=±1, where the zeros cluster; small errors in x give thus much larger deviations in $P_n\,'(x)$ and hence in the weights.
The computed integration weights are added in LEGPNT. This sum should equal 2.0, the length of the interval (−1,1). The deviation from this is 5e−13 for n=100, 7e−12 for n=1,000 and 7e−11 for n=10,000. The test computations have been carried out on a Cyber750, with approximately 14 digits floating point accuracy.

LEGPNT multiplies finally all weights by a uniform normalizing factor to make their sum equal 2.0.

19 SCF linearization

In big calculations by far the most time-consuming parts in the SCF procedure are
a) the evaluation of the matrix elements of the potential, the iteratively computed part of the hamiltonian.

$$V^k_{pq} = \sum_{i=1}^{N} w_i \; V(\mathbf{r}_i)\; \phi^{k*}_p(\mathbf{r}_i)\; \phi^k_q(\mathbf{r}_i) \qquad k=1,K \quad p,q=1,n \tag{19.1}$$

$V(\mathbf{r})$ is the potential; $\{w_i\}$ are the integration weights. With K k-points in the BZ, n basis functions (in each k-point) and N integration points the computational effort is propertional to KNn^2 .
b) the construction of the density.

$$\rho(\boldsymbol{r}_i) = \sum_k \sum_{pq} P^k_{pq}\ \phi^{k*}_p(\boldsymbol{r}_i)\ \phi^k_q(\boldsymbol{r}_i) \qquad i=1,N \tag{19.2}$$

P^k is the density matrix in the representation of the basis functions in the given k-point. Again we have the factor KNn^2 .
(In fact there are two ways to compute the density: from the density matrix or directly from the occupied eigenstates; see SS^charge density for a discussion of this).

A considerable saving of time is achieved when the involved multiple loop structures can be circumvented. For both cases this is possible, be it only at some cycles of the iterative procedure. The integration points and the basis functions are fixed quantities, so the only factors that change from cycle to cycle are the potential $V(\boldsymbol{r})$ in (19.1) and the density matrices P^k in (19.2). The quantities to be computed, V^k_{pq} and $\rho(\boldsymbol{r}_i)$, depend linearly on them. So if in the first case the current potential $V_0(\boldsymbol{r})$ is accurately described by a linear combination of potentials in previous cycles, then the same holds for the potential *matrices* V^k .
The second case is analogous: a linear combination of density matrices P^k defines the corresponding linear combination of density functions.
In the SCF procedure of BAND these approximations are tested in the routines POTAPP and RHOAPP respectively for the potential and the density. The implementation is structured as follows.

approximation of the potential

File itvstr contains a sequence of nstrp *orthonormal* (potential) functions $v_j(\boldsymbol{r}_i)$, j=1,nstrp. These functions are not the potentials on subsequent cycles, but they are derived from them. The current potential $v_0(\boldsymbol{r}_i)$ is on file itp. By straightforward numerical integration v_0 is expanded (POTAPP) in the set $\{v_i\}$

$$\bar{v}_0 \equiv \sum_{j=1}^{nstrp} c_j\, v_j \tag{19.3}$$

$$c_j = \sum_{i=1}^{N} w_i\ v_0(\boldsymbol{r}_i)\ v_j(\boldsymbol{r}_i) \tag{19.4}$$

Define the error in the approximation as the length (squared) of the difference vector $v_0 - \bar{v}_0$.

$$\text{err} = \text{vnorm} - \sum_j c_j^2 \tag{19.5}$$

$$\text{vnorm} = \sum_{i=1}^{N} w_i\ v_0^2(\boldsymbol{r}_i) \tag{19.6}$$

A logical approx tests the accuracy of the approximation, using a relative tolerance test

$$\text{approx} = \text{err} < \text{test} \times \text{vnorm} \tag{19.7}$$

As the SCF procedure approaches convergence ever smaller differences in the potential become relevant. The tolerance parameter test must therefore vary accordingly. SCF convergence is expressed by δ (the variable potdif in BAND), measuring the difference between the crystal potential at the previous cycle and the potential computed from the new density. The approximation criterium is set to

$$\text{test} = 10^{-7}\ \delta \tag{19.8}$$

Now, if the test is passed (approx=true) the coefficients c_j are subsequently used in EIGSYS to construct the potential matrices (one in each *k*-point) as linear combinations of the (potential) matrices that correspond to the set orthonormal potential vectors on file itvstr. The matrices are on file ithstr.

If the approximation is not sufficiently accurate, the difference vector $v_0 - \bar{v}_0$ is constructed, normalized and added to the set; nstrp increases by one. In EIGSYS the potential matrix is computed in the normal way, by numerical integration (19.1). The file ithstr is extended with the matrix (one for each *k*-point) corresponding to the potential vector that has been added to itvstr. The new matrix is the difference between the exact potential matrix for that cycle and the linear combination, multiplied by the normalization factor for the potential vector.

remarks

1. The number of potential vectors on itvstr and corresponding matrices on ithstr has a maximum nstrpx; nstrpx is assigned a value (70) in INIT. If this maximum has been reached and approx=false, i.e. a new vector should be added, the first vector in the set is removed. The new vector is then defined such, that the true potential is exactly approximated by the new orthonormal set, i.e. as if the removed vector had never been there. So, apart from normalization

$$v_{new\ in\ set} = v_0 - \sum_{j=2}^{nstrp} v_j \tag{19.9}$$

   Note that the lower bound in the summation is 2.
   In practice nstrp hardly ever exceeds 19 at the end of the SCF stage; in most cases convergence is already reached with nstrp equal to 4 or 5.
2. In a spin-unrestriced calculation we have different potentials for up- and down spins. The implementation contains therefore a loop over the number of independent spins (one or two), which we have omitted here for sake of clarity. For normalization and approximation tests the spin coordinate is treated on the same footing as the real space coordinates, e.g. (19.4) is replaced by

$$c_j = \sum_{\sigma} \sum_{i=1}^{N} w_i\ v_0^{\sigma}(\boldsymbol{r}_i)\ v_j^{\sigma}(\boldsymbol{r}_i) \tag{19.10}$$

In particular this implies that the linear approximation is either used for both spins, or for neither.

3. File itp does not contain the true potential v_0 directly, but two vectors; v_0 is defined as a specific linear combination of them (SS^iteration).

approximation of the density

The approach in routines RHOAPP and RHOPMT is analogous to that in POTAPP and EIGSYS. File itpstr contains nstrr (density) matrices (for each *k*-point in the BZ); nstrr is initially zero (INIT) and cannot exceed nstrrx (=70,INIT).
The current P-matrix, on file itpmat, is expanded in the set on itpstr. Norms and orthogonality for the matrices are defined by an inner product *S*

$$S(P_i, P_j) = \sum_{k=1}^{K} \sum_{pq} P^k_{i\,pq}\; P^k_{j\,pq} \tag{19.11}$$

Note the summation over the *k*-points.
The corresponding density vectors $\rho_j(\boldsymbol{r}_i)$ are on file itrstr.

remarks

1. As in the potential case, in spin-unrestricted calculations the density approximation is used for both spins or for neither; a loop over the spins is included in the definition of the inner product *S* (19.11); only *one* set of expansion coefficients is determined, applying to both spins simultaneously. The density matrices and vectors are distinct of course for up- and down spins.
2. Real and imaginary parts of the density matrices are treated as if the matrix *P* had just one more index to be summed over (with values 'real' and 'imaginary'). So in fact the inner product is defined (including also spin now)

$$S(i,j) = \sum_{k=1}^{K} \sum_{pq} \sum_{\sigma} \mathrm{Re}(P^{k,\sigma}_{i,pq})\,\mathrm{Re}(P^{k,\sigma}_{j,pq}) + \mathrm{Im}(P^{k,\sigma}_{i,pq})\,\mathrm{Im}(P^{k,\sigma}_{j,pq}) \tag{19.12}$$

Of course this is just a matter of definition. The only goal of the procedure is to have some way to test the adequacy of the linear approximation. See however below.

final notes

1. The savings in computer time by the linearizations depend strongly on the size of the calculation. In a big one the *approximate* evaluations take only a few percent of *exact* evaluations. During the first cycles there are of course no data for an expansion: the set has to be build up. During the iterations an additional vector is needed in the expansion set from time to time. In a typical big calculation with slow convergence we would have, say 100 cycles to reach self-consistency and exact evaluations in 20 of them, i.e. linear approximations would be used in 80 cycles. Accounting for other parts of the SCF

procedure and for the fact that the linearization procedures involve some overhead of testing and file-manipulation, the gain is then a factor between 2 and 4 (in the SCF part).

2. The linearizations of the density and potential are tested and used independently. As a matter of experience the potential behaves better in this respect than the density: linear approximations in EIGSYS are used more often that in RHOPMT. In some cases the density is exactly computed twice as many times as the potential matrix.
3. The criterium factor 10^{-7} in the approximation tests is a result of trial and error. Obviously a less stringent test would allow the linear approximations more often, increasing at first sight the gain in efficiency.
   However the switch from approximate to exact evaluation unavoidably takes place from time to time during the iterations. At such a moment there is a (slight) discontinuity in the development towards the self-consistent solution: the function space in which the potential, or density, is described, changes abruptly. The iterative procedure is rather sensitive to this and suffers from instability and oscillations if these changes are too large. The adopted value 10^{-7} appears to be a reasonable compromise.
4. The approximation procedure for the potential is simple and straightforward; in particular the vectors, their norms and the inner product are defined in a natural way. With respect to the density approximation we feel less comfortable. The fact that this case performs worse and that the instability always seems to occur when the computational mode is switched for the density, but not if it happens for the potential, suggests some inbalance in the procedure. Further investigation might be useful; this has no high priority however because in practice the implemented strategy functions reasonbly well.

20 Symmetry

The most obvious symmetry in crystals is the translational symmetry of the Bravais lattice. The corresponding operators, the Bravais translations, are denoted by a capital $\boldsymbol{T}$. All $\boldsymbol{T}$ are products of n basis $\boldsymbol{T}_i$ that span the unit cell of the n-dimensional crystal, by which we mean an atomic system with translational symmetry in n directions; n=0 (molecule), 1 (polymer), 2 (slab) or 3 (bulk crystal).

The complete space group consists of affine transformations $\{\boldsymbol{t};R\}$

$$\{\boldsymbol{t};R\}\boldsymbol{x} = \boldsymbol{t} + R\boldsymbol{x} \tag{20.1}$$

R is a unitary operator and $\boldsymbol{t}$ is a translation which may have components in the n directions of periodicity. The *generators* of the space group is a subset of $\{\boldsymbol{t};R\}$ that generates all $\{\boldsymbol{t};R\}$ by multiplication with the Bravais translations. The set of generators is defined to be minimal in the sense that not any pair of them is related by a pure Bravais translation.

A few aspects:

The inverse of $\{\boldsymbol{t};R\}$ is $\{\boldsymbol{t};R\}^{-1} = \{-R^{-1}\,\boldsymbol{t};R^{-1}\}$:

$$\{\boldsymbol{t};R\}\{-R^{-1}\,\boldsymbol{t};R^{-1}\}\boldsymbol{x} = \boldsymbol{t} + R\left(-R^{-1}\boldsymbol{t} + R^{-1}\boldsymbol{x}\right) = \boldsymbol{t} - RR^{-1}\,\boldsymbol{t} + RR^{-1}\,\boldsymbol{x} = \boldsymbol{x} \qquad (20.2)$$

\# All generators have different point group parts R. Suppose $\{\boldsymbol{t}_1\ ;R\}$ and $\{\boldsymbol{t}_2\ ;R\}$ are both generators. Then

$$\{\boldsymbol{t}_1\ ;R\}\{\boldsymbol{t}_2\ ;R\}^{-1}\,\boldsymbol{x} = \boldsymbol{t}_1\ - RR^{-1}\,\boldsymbol{t}_2\ + RR^{-1}\,\boldsymbol{x} = (\boldsymbol{t}_1\ - \boldsymbol{t}_2\ ) + \boldsymbol{x} \qquad (20.3)$$

The translation $(\boldsymbol{t}_1\ - \boldsymbol{t}_2\ )$ is then a symmetry operator and hence a Bravais translation $\boldsymbol{T}$. So

$$\{\boldsymbol{t}_1\ ;R\}\times\{\boldsymbol{t}_2\ ;R\}^{-1} = \boldsymbol{T} \qquad \{\boldsymbol{t}_1\ ;R\} = \boldsymbol{T}\times\{\boldsymbol{t}_2\ ;R\} \qquad (20.4)$$

contrary to the definition that two generators are not related by a pure Bravais translation.

\# For each space group operator $\{\boldsymbol{t};R\}$ the point-group part R is a symmetry operator of the Bravais lattice. Let the symbol ~ denote symmetry equivalency. Then

$$\boldsymbol{x}' \sim \{\boldsymbol{t};R\}^{-1}\,\boldsymbol{x}\ \sim\{\boldsymbol{t};R\}^{-1}\,\boldsymbol{x} + \boldsymbol{T} \sim \{\boldsymbol{t};R\}\left(\{\boldsymbol{t};R\}^{-1}\boldsymbol{x} + \boldsymbol{T}\right) = \boldsymbol{x} + R\boldsymbol{T} \qquad (20.5)$$

So the translation $R\boldsymbol{T}$ is a symmetry operator and hence a Bravais translation.

\# Until now it has tacitly been assumed that the unit cell is chosen as small as possible. This is implicit in the argument that a pure translation must be a Bravais translation if it is a symmetry operator. One may of course define a larger unit cell however, for example the simple cubic 'double' unit cell in a bcc crystal. This modifies the statements above a little:

With any choice of unit cell, the resulting Bravais group is a subgroup of the true Bravais group. The larger unit cell can be considered as a conjuction of N smaller unit cells. They are characterized by operators $\boldsymbol{T}_i$, i=1..N, of the true Bravais lattice.

These $\boldsymbol{T}_i$ are naturally symmetry operators of the system, but they do not occur now in the employed Bravais translation group. Consequently they will have to be included in the set of generators. In other words, in case of a larger unit cell the point group parts R of the $\{\boldsymbol{t};R\}$ are symmetry operators of the true Bravais lattice, but not necessarily of the employed one; moreover each R in the set of generators now occurs N times; the different translational parts $\boldsymbol{t}$ belonging to the same R differ (or can be chosen to differ) by the $\boldsymbol{T}_i$ of the true lattice that define the enlarged unit cell.

\# From time to time we have to deal with sets of points in the unit cell and their symmetry relations. For a point $\boldsymbol{x}$ in the unit cell the image $\{\boldsymbol{t};R\}\boldsymbol{x}$ may be located in a neighbouring cell. We define now the operator $\{\overline{\boldsymbol{t};R}\}$ as the operator $\{\boldsymbol{t};R\}$ followed by a Bravais back-translation to the central unit cell, i.e.

$$\{\overline{\boldsymbol{t};R}\} = \{\boldsymbol{t}+\boldsymbol{T};R\} \qquad (20.6)$$

where $\boldsymbol{T}$ depends on the point to be operated upon; it is always the unique Bravais translation which projects the image into the central unit cell. This definition is convenient because, given a set of all equivalent points $\{\boldsymbol{x}\}$ in the unit cell the points $\{\overline{\boldsymbol{t};R}\}\,\boldsymbol{x}$ now belong to the same set and are not 'only' translationally equivalent points. We will call $\{\overline{\boldsymbol{t};R}\}$ a *centralizing* operator. It is easily verified that the set of centralizing generators constitutes a mathematical group.

computation of the operators

The operators $\{\boldsymbol{t};R\}$ are represented and computed in the coordinate representation: R is a unitary 3×3 matrix and $\boldsymbol{t}$ a 3-component vector.
In accordance with the notes above the construction of the operators takes place in two steps:

A compute the point group symmetry of the Bravais lattice (the holosymmetric point group)

B for each operator R find the translational part $\boldsymbol{t}$, if any, that makes it a space group operator.

point group operators

We address a more general problem. Let N sets of points be given, $\boldsymbol{x}_{ij}$, j=1..N, i=1..M_j, in n-dimensional space: $\boldsymbol{x}_{ij} \equiv(x_{ij}^1 ,x_{ij}^2 ,..,x_{ij}^n)$. We want to compute all real unitary transformations R in the form of n×n matrices, that are simultaneously symmetry operators for each of the N groups of points, $R\boldsymbol{x}_{ij} \quad \{\boldsymbol{x}_{kj} \}_k$. An example is the point group symmetry of a molecule: the points are the locations of the atoms; chemically different atoms fall in distinct sets and the dimensionality is (usually) three. In case of the Bravais lattice we have only one set, but infinitely many points; we will see however that it is sufficient to consider only the first few 'stars' of lattice points around the origin.

The algorithm is based on the fact that n pairs of points $(\boldsymbol{p}_i ,\boldsymbol{q}_i)$, i=1..n, define a linear transformation in n-dimensional space via the system of equations

$$R\boldsymbol{p}_i = \boldsymbol{q}_i \qquad i=1..n \tag{20.7}$$

The set $\{\boldsymbol{p}_i \}$,i=1..n will be called the basis and the set $\{\boldsymbol{q}_i \}$,i=1..n the projection. Any admissible basis and projection must each constitute a linearly independent set, otherwise the system (20.7) does not properly define the transformation R; in the following we assume that every basis or projection has been checked to be independent.
Let now R be one of the symmetry operators to be found and let $\{\boldsymbol{p}\}$ be an arbitrary basis, the points of which may be taken from any combination of the N sets. Since R is a by assumption a symmetry operator there must obviously be a projection $\{\boldsymbol{q}\}$ such that the combination of R, $\{\boldsymbol{p}\}$ and $\{\boldsymbol{q}\}$ satisfies (20.7). So the algorithm is

1. find any basis $\{\boldsymbol{p}\}$.
2. loop over all distinct projections $\{\boldsymbol{q}\}$ and compute R from (20.7).
3. check for every point $\boldsymbol{x}_{ij}$, j=1..N, i=1..M_j , whether $R\boldsymbol{x}_{ij}$ coincides with a point in the appropriate (j-th) set. If not, discard R.

Although this set-up suffices to determine all symmetry operators, the efficiency can be improved somewhat. To avoid duplications and unnecessary work in loop 3) we check first, for each computed R the unitarity and whether it has already been found before. Furthermore, as for each point $\boldsymbol{x}_{ij}$ its image $R\boldsymbol{x}_{ij}$ must belong to the same set j, we impose this condition also on the projection set $\{\boldsymbol{q}\}$: $\boldsymbol{q}_i$ and $\boldsymbol{p}_i$ must belong to the same set, for all i=1..n; this shortens loop 2).

implementation

Subroutine PNTGRP computes the $n \times n$ matrices corresponding to a given configuration of points in n-dimensional space.

First a linearly independent basis is constructed; array indgrp stores the types of basis-points: the i-th basis point belongs to set indgrp(i). Then all distinct n-tuples of points are considered to serve as projection $\{\boldsymbol{q}\}$. Indgrp is used to control that $\boldsymbol{p}_i$ and $\boldsymbol{q}_i$ belong to the same set. After assuring linear independency of $\{\boldsymbol{q}\}$ and unitarity of R, all image points $R\boldsymbol{x}_{ij}$ is examined to see whether R is a symmetry operator.

Two index arrays, indbas and indprj keep track of which points of the sets are used in the basis and (current) projection respectively.

remarks

The identity operator, a symmetry operator of every system of points, is generated first; it is the first symmetry operator in the output list.
The inversion operator, if it occurs, is second in the output list. PNTGRP moves it to this position after the calculation of all operators.
Identity of points and operators is checked by comparing real numbers. The tolerance is set at a fixed absolute value 10^{-6} . To avoid numerical problems it may therefore be necessary to rescale the input points.
The requirement that the basis be independent in n-dimensional space implies that PNTGRP cannot compute for instance the three-dimensional point group of a flat molecule. A possible solution to this is:

   a. rotate the molecule to the xy-plane. (20.8)
   b. generate the point group in two-dimensional space (PNTGRP).
   c. copy the obtained 2×2-matrices to the left-upper parts of 3×3-matrices, setting R_{33} =1 and the remaining matrix elements zero.
   d. Incorporate the reflection in the xy-plane in the symmetry group (SYMADD).
   e. rotate the operators back to the original coordinate frame.

space group operators

First we construct the point group of the Bravais lattice. For this purpose PNTGRP is called with one set of points: the lattice points of a few 'stars' around the origin. Let $\boldsymbol{T}_i$, i=1..n be the unit cell basis vectors of the n-dimensional crystal. It is then sufficient to consider only the stars of these $\boldsymbol{T}_i$, because their combined sets of points has the same symmetry as the complete Bravais lattice:

Let R be a symmetry operator of the Bravais lattice. The image of $\boldsymbol{T}$ under R, $\boldsymbol{T}^R$ say, belongs to the star of $\boldsymbol{T}$ because R is unitary and hence preserves lengths. So R is a symmetry operator of each star separately. In particular this applies to the basis vectors $\boldsymbol{T}_i$ and their stars.
Let now R be a symmetry operator for the basis stars. Then for any lattice point $\boldsymbol{T}$, necessarily a linear combination of the basis vectors, the image under R is

$$R\boldsymbol{T} = R\sum_{i=1}^{n} m_i \boldsymbol{T}_i = \sum_{i=1}^{n} m_i \boldsymbol{T}_i^R \qquad (20.9)$$

which belongs to the Bravais lattice because each $\boldsymbol{T}_i^R$ is a lattice point. Hence R is a symmetry operator of the whole Bravais lattice.

Construction of the space group generators thus takes place (SYMCRY) as follows:

1. generate the stars of points corresponding to the basis vectors (LATTPT).
2. compute from these points the Bravais point group operators R (PNTGRP).
3. for each R find out whether it can be made into a space group operator $\{\boldsymbol{t};R\}$:
   The atoms are divided in (chemically) distinct types.
   a. take any atom in the central unit cell, say the first atom of the first type, at position $\boldsymbol{x}$ and compute the image under R: $\boldsymbol{x}'=R\boldsymbol{x}$.
   b. run over all atoms of the same type and in the same unit cell, at positions **y** and define the translational part $\boldsymbol{t}$ of the space group operator as $\boldsymbol{t}=\boldsymbol{y}-\boldsymbol{x}'$, so that $\{\boldsymbol{t};R\}\boldsymbol{x}=\boldsymbol{y}$.
   c. loop over all atoms of all types and check whether their images under $\{\boldsymbol{t};R\}$ coincide with atoms of the same type, possibly in neighbouring cells. If not, discard this $\{\boldsymbol{t};R\}$ and try the next possible translation (step *b*) by taking another atom $\boldsymbol{y}$.

remarks

1. We have to consider only atoms $\boldsymbol{y}$ in the central unit cell. Atoms in other cells result in operators that differ from those obtained already by Bravais translations.
2. If for a particular R no translational part is found that satisfies the requirements, R is discarded altogether.
3. If for a particular R more than one $\boldsymbol{t}$ is found, then the crystal unit cell has not properly been defined, as discussed above. The true, smaller unit cell is then computed and the procedure restarted. This is necessary because with a too large unit cell we may not have found all point group operators of the true Bravais lattice.
   The determination of the true unit cell is as follows:
   Two arrays of lattice vectors are present. The first, the array avec in the program, is fixed and stores the lattice vectors as defined by the user. The second, vlatt, describes the true unit cell; it is only used locally (in SYMCRY). Vlatt is initiated by copying avec; it may be changed in the course of the symmetry analysis.
   Since the identity E is the first operator in the output list of PNTGRP, this is the first operator processed in the construction of the space group generators. Obviously $\{\boldsymbol{0};E\}$ is a space group operator. Any next $\boldsymbol{t}$ that is found with E, must be an element of the true Bravais lattice. $\boldsymbol{t}$ is expanded in the vectors vlatt.

$$\boldsymbol{t} = \sum_{i=1}^{n} c_i \ \text{vlatt}_i \qquad (20.10)$$

If vlatt describes a too large unit cell, then for at least one $\boldsymbol{t}$ belonging to E, one of the expansion coefficients is non-integer. By adding or subtracting integer multiples of the $\{\text{vlatt}_i\}$ we construct the Bravais translation (of the true lattice)

$$\boldsymbol{v} = \sum_{i=1}^{n} \bar{c}_i \ \text{vlatt}_i \tag{20.11}$$

where all $\bar{c}_i$ are in the unit interval [0,1) and differ from the c_i in (20.10) by integers. By assumption at least one of the obtained coefficients, $\bar{c}_j$ say, is non-zero; in the set {vlatt} we replace then the j-th vector by $\boldsymbol{v}$. The new set describes a smaller unit cell. The procedure can now be restarted: generate the pointgroup corresponding to {vlatt}, etc.

When each $\boldsymbol{t}$ found with E has an integer expansion (20.10) the correct unit cell has been found and the procedure can be continued with all other operators R.

4. The same Bravais lattice may be generated with different sets of basis lattice vectors. The crystal unit parallellepiped is not uniquely defined: we may add to any of the basis vectors an integer multiple of any of the other basis vectors. For various reasons it is convenient in the program (though not strictly necessary) to avoid extreme choices in this respect and define the unit parallellepiped as compact as possible. This is done by recombining the vectors such that they have minimal lengths. LATTCH (lattice check) performs that task; LATTCH checks in this way also linear dependency of the lattice vectors.

symmetry in k-space

The connection between the space group symmetry in real space and the symmetry in k-space is easily derived [Jones 1975]. According to Bloch's theorem any eigenstate $\psi_n(\boldsymbol{k};\boldsymbol{r})$ can be written in the form

$$\psi_n(\boldsymbol{k};\boldsymbol{r}) = u_n(\boldsymbol{k};\boldsymbol{r})\ e^{i\boldsymbol{k}\cdot\boldsymbol{r}} \tag{20.12}$$

where $u_n(\boldsymbol{k};\boldsymbol{r})$ is symmetric with respect to all Bravais translations

$$u_n(\boldsymbol{k};\boldsymbol{T}+\boldsymbol{r}) = u_n(\boldsymbol{k};\boldsymbol{r}) \tag{20.13}$$

The occurrence of $\boldsymbol{k}$ in the argument list of the function u does therefore not imply that it transforms as the corresponding irrep of the translation group, but it signifies only that the u-parts of eigenstates $\psi(\boldsymbol{k})$ at different k-points are different.

It follows from the hamiltonian equation

$$\{-\Delta/2 + V(\boldsymbol{r})\}\ \psi(\boldsymbol{k};\boldsymbol{r}) = e(\boldsymbol{k})\ \psi(\boldsymbol{k};\boldsymbol{r}) \tag{20.14}$$

that the periodic function $u(\boldsymbol{k};\boldsymbol{r})$ satisfies

$$\{-\Delta/2 + k^2/2 - \mathrm{i}\boldsymbol{k}\cdot\ + V(\boldsymbol{r})\}\ u(\boldsymbol{k};\boldsymbol{r}) = e(\boldsymbol{k})\ u(\boldsymbol{k};\boldsymbol{r}) \tag{20.15}$$

Let now $\{\boldsymbol{t};R\}$ be a space group operator, defining the coordinate transformation $\boldsymbol{r}\rightarrow\boldsymbol{r}'$

$$r_i' \equiv (\{\boldsymbol{t};R\}\boldsymbol{r})_i = t_i + \sum_j R_{ij}\, r_j \tag{20.16}$$

For the derivatives this gives

$$\frac{\partial}{\partial r_i} = \sum_j \frac{\partial r_j'}{\partial r_i}\frac{\partial}{\partial r_j'} = \sum_j R_{ji}\frac{\partial}{\partial r_j'} \tag{20.17}$$

Define then the transformation in k-space $\boldsymbol{k}\rightarrow\boldsymbol{k}'$ by

$$k_i' \equiv (R\boldsymbol{k})_i = \sum_j R_{ij}\, k_j \tag{20.18}$$

Simultaneous application of the transformations (20.16) and (20.18) leaves (20.15) invariant because R is unitary and $V(\boldsymbol{r}')=V(\boldsymbol{r})$. Hence the solutions $\{e,\psi\}$ in $\boldsymbol{k}$ and $\boldsymbol{k}'$ are symmetry related:

$$e_n(R\boldsymbol{k}) = e_n(\boldsymbol{k}) \tag{20.19a}$$

$$u_n(R\boldsymbol{k};\{\boldsymbol{t};R\}\boldsymbol{r}) = u(\boldsymbol{k};\boldsymbol{r}) \tag{20.19b}$$

$$\psi_n(R\boldsymbol{k};\{\boldsymbol{t};R\}\boldsymbol{r}) = e^{\,i\sum_{mn} R_{mn}t_m k_n}\,\psi_n(\boldsymbol{k};\boldsymbol{r}) \tag{20.19c}$$

So, if $\{\boldsymbol{t};R\}$ is a symmetry operator in real space, then R is a symmetry operator in k-space.
In addition the inversion operator J is a symmetry operator in k-space, regardless of the space group: (20.15) is transformed into its complex conjugate; all eigenvalues $e_n(\boldsymbol{k})$ are real because the hamiltonian is an hermitian operator. Hence

$$e_n(-\boldsymbol{k}) = e_n(\boldsymbol{k}) \tag{20.20a}$$

$$u_n(-\boldsymbol{k};\boldsymbol{r}) = u_n^*(\boldsymbol{k};\boldsymbol{r}) \tag{20.20b}$$

The construction of the point group operators in k-space is thus straightforward once we have calculated the space group generators $\{\boldsymbol{t};R\}$:

a. gather all distinct point group operators R occurring among the $\{\boldsymbol{t};R\}$.

b. add the inversion operator J if it is not yet present (and of course all products JR).

Whereas real space is 3-dimensional this does not necessarily hold for k-space: in an nD crystal k-space has only n dimensions. The operators must then of course also be operators in nD space and can be represented as $n\times n$ matrices. For a slab for instance the reflection in the plane is an irrelevant operator in k-space.

With an appropriate orientation of the coordinate system the relevant part of the real space operators R is then the left-upper $n\times n$ submatrix. From (20.18) we conclude that this is an operator in k-space *if* the summation can be restricted to the first n terms, that is if all R_{ij} are zero when $i\leq n$, $j>n$ or vice-versa. Otherwise a point $\boldsymbol{k}$ would be mapped onto a point 'outside' the nD region.
The computation of the k-space symmetry group is therefore

1. For each $\{\boldsymbol{t};R\}$ check that it does not couple the first n coordinates to the last $3-n$, and gather all resulting distinct $n\times n$ matrices (SYMPRJ).
2. Add inversion in nD space (SYMADD).

remark
The dimensionality of k-space is in BAND not only determined by the translational symmetry of the atomic system. If that were the case, none of the operators $\{\boldsymbol{t};R\}$ could couple the nD space to the other coordinates. However, BAND optionally neglects dispersion in certain directions in k-space, thereby artifically reducing the dimensionality of the BZ. This option may be activated (via input) for example when the crystal unit cell is much larger in one particular direction that in others (SS^BZ-integration).

integration in k-space

The point group symmetry in k-space is applied to reduce integrations over the BZ to integrals over the irreducible wedge, the symmetry unique part of the BZ. In the employed analytic-quadratic integration method [Wiesenekker *et al.* 1988, Wiesenekker and Baerends 1990] the BZ is divided in simplices and the integration points are the vertices and the midpoints of the edges. The BZ is constructed as a Wigner-Seitz polyhedron (in two dimensions a polygon, in one dimension an interval). The irreducible wedge is determined (SS^BZ-integration) and *that* region is divided in simplices etc. By use of symmetry the number of k-points needed is thus reduced; this is an important saving in computer time since virtually all cost-determining aspects in a calculation scale with the number of k-points processed.

The energy bands $e_n(\boldsymbol{k})$ are totally symmetric; hence a quantity like the density of states (DOS) can be computed by integration over the irreducible wedge only. This does not apply however to all properties. The eigenstates in symmetry related k-points are not identical (but they are symmetry related). If $\{\boldsymbol{t};R\}$ is a space group operator and $\boldsymbol{k}'$ and $\boldsymbol{k}$ are related by $\boldsymbol{k}'=R\boldsymbol{k}$, then

$$\psi_n(\boldsymbol{k}';\boldsymbol{r}) = e^{i\boldsymbol{t}.\boldsymbol{k}'}\,\psi_n(\boldsymbol{k};\{\boldsymbol{t};R\}^{-1}\boldsymbol{r}) \tag{20.21}$$

Let now F be a quantity to be computed as

$$F = \sum_n \int_{\mathrm{BZ}} \mathrm{d}\boldsymbol{k}\; F_n(\boldsymbol{k}) \tag{20.22}$$

where $F_n(\boldsymbol{k})$ is the contribution to F from the one-particle state $\psi_n(\boldsymbol{k};\boldsymbol{r})$

$$F_n(\boldsymbol{k}) = F[\psi_n(\boldsymbol{k};\boldsymbol{r})\,] \tag{20.23}$$

The summation is over the bands and integration is over the complete (first) BZ. Using the space group generators $\{\boldsymbol{t};R\}$ in real space and the corresponding operators R in k-space (20.22) can be rewritten as

$$F = \sum_{\{\boldsymbol{t}R\}} \sum_{n} \int_{irr.BZ} \mathrm{d}\boldsymbol{k}\; F_n(R\boldsymbol{k}) \;=\; \sum_{\{\boldsymbol{t}R\}} \sum_{n} \int_{irr.BZ} \mathrm{d}\boldsymbol{k}\; F[\psi_n(\boldsymbol{k};\{\boldsymbol{t};R\}^{-1}\boldsymbol{r})\,] \tag{20.24}$$

The charge density for instance is given by

$$\rho(\boldsymbol{r}) = \sum_{\{\boldsymbol{t}R\}} \sum_{n} \int_{irr.BZ} \mathrm{d}\boldsymbol{k}\; \hat{o}\,\psi_n(\boldsymbol{k};\{\boldsymbol{t};R\}^{-1}\boldsymbol{r})\,\hat{o}^2 \;\equiv \sum_{\{\boldsymbol{t}R\}} \rho^{irr}(\{\boldsymbol{t};R\}^{-1}\boldsymbol{r}) \tag{20.25}$$

$\rho^{irr}(\boldsymbol{r})$ is the charge density resulting from integration over the irreducible wedge only.

The occupation numbers $o_n(\boldsymbol{k})$ for the one-particle states, which are the weights for the numerical integration over the irreducible BZ, are in the program determined such, that they represent also the equivalent k-points. The integrals over the BZ are therefore scaled already and e.g. the symmetrized density is computed as

$$\rho(\boldsymbol{r}) = \frac{1}{N_G} \sum_{\{\boldsymbol{t}R\}} \sum_{n} \sum_{\mathrm{k}} o_n(\boldsymbol{k})\;\hat{o}\,\psi_n(\boldsymbol{k};\{\boldsymbol{t};R\}^{-1}\boldsymbol{r})\,\hat{o}^2 \;\equiv \frac{1}{N_G} \sum_{\{\boldsymbol{t}R\}} \bar{\rho}^{irr}(\{\boldsymbol{t};R\}^{-1}\boldsymbol{r}) \tag{20.26}$$

N_G is the number of symmetry operators.
We see that the total charge density is obtained by projecting out the symmetric component of $\bar{\rho}^{irr}(\boldsymbol{r})$.
In BAND the density, like all crystal functions, is represented by its values in the crystal integration points. The integration scheme is symmetric, which allows the symmetrization (20.26) by straightforward numerical integration (see below).

integration in real space

The numerical integration formula is symmetric: if $\boldsymbol{r}$ is an integration point and $\{\boldsymbol{t};R\}$ a space group operator then $\boldsymbol{r}'\equiv\{\boldsymbol{t};R\}\boldsymbol{r}$ is also an integration point and the weights are equal. The symmetry unique points are called the *generators* (of the integration formula).
The *construction* of the symmetric scheme is extensively discussed in chapter III. Here we are concerned with the *utilization* of the symmetry property.

1. The component of a particular irreducible representation (irrep) of the group can be projected out of a function. Whereas this applies to any irrep, the most common application is the symmetrization of a function, i.e. the projecting out of the (totally) symmetric conponent.
This is done in BAND with the density. As we saw above, the integration in k-space over the irreducible part of the BZ yields an incorrect, that is, non-symmetric charge density: we need the symmetric component. The projector is (20.26)

$$P^{A1} = \frac{1}{N_G} \sum_{\{tR\}} \{t;R\} \qquad (20.27)$$

The points on file itpnt are grouped in a number of (large) blocks. Each block contains a sequence of sets of symmetry related points. The file with basis function values is organized accordingly and the a-symmetric density, computed from the basis functions (SS^charge density) has the same structure. Symmetrization is performed by averaging the values in each set of symmetry related points (20.27). This happens in RHOPMT (or RHOPSI, see SS^charge density).

2. If a function belongs to an irrep of the group, we have to compute and store only the values in the generators, the symmetry unique points. The values in related points can be derived when desired from the transformation properties of the irrep. This possibility is used in BAND for the density, the potential, the fit functions and the fit potentials. All these functions are totally symmetric.
With the data structure on itpnt the functions can easily be expanded again over all points. This is done for the potential in EIGSYS, where the matrix elements of the hamiltonian are evaluated by numerical integration of the potential against products of the basis functions (the basis functions currently used do not belong to irreps).

3. The numerical integral of a function is greatly simplified if it belongs to an irrep. Any irrep which is not the symmetric one integrates to zero. For a symmetric integrand we have to loop only over the generator points because the values in the related points are equal and they can be accounted for by multiplication by the appropriate factor (the number of equivalent points), which can be included in the weights of the generators.
This is used for the expansion of the density in the fit set (7.6a) in RHOFIT.

It would be an enormous improvement in efficiency of BAND if also the valence basis functions were organized according to the irreps. This block diagonalizes the hamiltonian matrices. The off-diagonal blocks are zero on grounds of symmetry and hence need not be computed at all. For the diagonal blocks the loop over the integration points reduces to the generators.

special symmetry routines

Some specific tasks related to symmetry are performed by special routines. These are examined here. Most of them are general as regards the dimensionality; the point group operators are represented as $n \times n$ matrices.

extension of a point group

Given a point group G^0 and an operator R, SYMADD constructs the compounded group, consisting of G^0 and R and all additional operators required to make the new set a group.
A set of point group operators can be extended to a group by computing all possible product operators and adding them if they are not yet present, until the set is closed under multiplication. Both the 'left' and the

'right' products have to be checked since for an arbitrary pair of operators R_1 and R_2 the products $R_1 \times R_2$ and $R_2 \times R_1$ need not be identical.
This straightforward method assures that also the identity E and the inverse of each operator will be generated: for every point group operator R there is an integer M such that $R^M = E$. Hence $R^{-1} = R^{M-1}$ so that R and E both occur among the products of R with itself.

SYMADD uses this algorithm but supposes that the input set G^0 is already a group: products among its operators are not checked for occurrence in the set.

checking and reducing a point group

SYMCHK checks a point group against a set of points and removes the operators that are not symmetry operators of the point set. The points may be divided in subsets of equivalent points.
The algorithm is simple: for each operator R compute all image points and check whether these coincide with points in the appropriate subsets.
Note that symmetry operators of the point set that were not present in the input group are not generated, i.e. the output group does not necessarily represent all symmetry of the point set (compare routine PNTGRP).

checking completeness of a point group

GRPCHK checks whether a set of $n \times n$ matrices is closed under multiplication, i.e. whether it constitutes a group. An output error parameter ier gives the outcome (0:group, 1:not a group).

analysis and characterization of a symmetry operator

MAT3AN analyzes a unitary 3×3 matrix (unitarity is checked). Output are the determinant D, an axis $\boldsymbol{a}$ (normalized vector) and an angle α. For a pure rotation (D=1) $\boldsymbol{a}$ and α are the axis and angle of clockwise rotation. An improper rotation (D=−1) can be written as a reflection times a rotation around the normal on the reflection plane; the axis and angle refer to that rotation; a pure reflection has α=0.
Two special operators are the identity E (D=1, α=0) and the inversion J (D=−1, α=π). For both of them the axis is undefined and it is taken arbitrarily as the z-axis. MAT3AN checks first whether R is either of these two operators. For all other operators R the axis and angle are computed as follows.

An auxiliary vector $\boldsymbol{w}$ orthogonal to the axis $\boldsymbol{a}$ is constructed. α is then the angle between $\boldsymbol{w}$ and $R\boldsymbol{w}$; the axis $\boldsymbol{a}$ is the vector product $\boldsymbol{w} \times R\boldsymbol{w}$. A suitable $\boldsymbol{w}$ can be computed from an arbitrary vector $\boldsymbol{v}$ by removing the component along the axis. For a pure rotation (D=1):

$$\boldsymbol{w} = (R-1)\boldsymbol{v} \tag{20.28}$$

and for an improper rotation (D=−1):

$$\boldsymbol{w} = (R^2-1)\boldsymbol{v} \tag{20.29}$$

Of course the 'arbitrary' vector $\boldsymbol{v}$ must not coincide with the axis, since that would give $\boldsymbol{w}=0$. This is prevented in MAT3AN by taking for $\boldsymbol{v}$ the x-axis unless $R_{11}=1$ (20.28) respectively $R_{11}=-1$ (20.29); in that case the y-axis is taken.

A zero result for $\boldsymbol{w}$ is also found for special angles of rotation. For a pure rotation (20.28) if $\alpha=0$ (the identity E); for an improper rotation (20.29) if $\alpha=\pi$ (the inversion J) or $\alpha=0$ (a pure reflection). As E and J have been checked a priori the only possibility is the pure reflection. So, if $\boldsymbol{w}$ is found to be zero, the situation is immediately clear: $\alpha=0$ and the axis is given by

$$\boldsymbol{a} = (R-1)\boldsymbol{v} \qquad (20.30)$$

Finally, in case of a pure rotation (not the identity), the determination of the axis as the vector product $\boldsymbol{w}\times R\boldsymbol{w}$ may yield zero. If this situation is encountered ($D=1$, $\alpha=\pi$) a second general vector $\boldsymbol{w}'$ orthogonal to the axis is constructed (20.28), where $\boldsymbol{v}$ is now the y-axis (or the z-axis). The axis $\boldsymbol{a}$ is then found as the vector product $\boldsymbol{w}\times\boldsymbol{w}'$.

local symmetry around an atom

GRPTYP is used in the integration package POINTS to determine the appropriate type of integration formula for the atomic spheres. The symmetry of the formula must correspond to the local symmetry of an atom. The available special (Lebedev) formulas for the spherical surface all have the octahedral symmetry. Apart from these we may generate a product formula in the coordinates $\cos\theta$ and ϕ [chapter III]; this can be used for all axial groups. The only point group type not covered is the icosahedral symmetry. Special formulas of this type are known [Stroud 1971], but they have not yet been implemented in the integration package. Icosahedral symmetry is rarely encountered in polyatomic systems and thusfar this restriction in the possibilities has not played a role.

GRPTYP determines from a set of point group operators (3×3 matrices) whether the group is of the octahedral or axial type; the octahedral type includes subgroups such as the tetrahedral group; an icosahedral group will be detected as an 'error'.
Apart from the type of symmetry, GRPTYP computes the rotation matrix that will bring the coordinate frame into a standard orientation (in which the implemented integration formulas for the sphere are defined). The standard orientation of an axial group has the z-axis as the axis of rotation and the x-axis in a 'vertical' reflection plane (if there is any). For an octahedral group (or subgroup) the standard orientation is the usual orientation of the cube: 4-fold rotations around the coordinate axes and 3-fold and improper 6-fold rotations around the (111)-directions.
Finally GRPTYP outputs for axial groups the order of the rotation.

The algorithm in GRPTYP is based on the determination of the main axis and the highest order secondary axis of the group. These are found by subsequently analyzing all operators (MAT3AN) and updating the main and secondary axes and their orders of rotation.

A group is defined (in MAT3AN) to be of the axial type whenever the secondary axis has a rotation order not higher than 2; otherwise it is of the octahedral type. In the latter case the angle between the main and secondary axis is checked in relation to their rotation orders (6-fold, 4-fold or 3-fold). In this way the icosahedral symmetry is detected as an 'incorrect octahedral' symmetry.

correcting symmetry inaccuracies in a set of points

Given a set of points which is *approximately* symmetric with respect to a symmetry group, SYMTRZ removes the numerical noise from the position coordinates. The points are displaced slightly, such that the output set is *exactly* symmetric (to machine accuracy). This is done by explicitly projecting out the symmetric component of the set of points.

The symmetry group is a space group, of which the generators $\{\boldsymbol{t};R\}$ are input into SYMTRZ; the set of points in the crystal unit cell must be complete: if $\boldsymbol{x}$ is in the set then $\{\overline{\boldsymbol{t};R}\}\,\boldsymbol{x}$ must belong to the set, apart of course from the coordinate inaccuracies to be corrected. $\{\overline{\boldsymbol{t};R}\}$ is the centralizing generator, defined earlier in this section.
A 'normal' point group, e.g. for a molecule, can be handled by specifying the crystal dimensionality as zero and giving the zero vector for all translational parts $\boldsymbol{t}$ of the operators.

Given $\boldsymbol{x}$ there is for every $\{\boldsymbol{t};R\}$ exactly one point $\boldsymbol{y}$ in the set such that $\boldsymbol{x}$ is (approximately) the centralized image of $\boldsymbol{y}$ under $\{\boldsymbol{t};R\}$. The exact image of $\boldsymbol{y}$ under $\{\overline{\boldsymbol{t};R}\}$, approximately equal to $\boldsymbol{x}$, is denoted $\bar{\boldsymbol{x}}(\{\boldsymbol{t};R\})$

$$\boldsymbol{x} \approx \bar{\boldsymbol{x}}(\{\boldsymbol{t};R\}) = \{\overline{\boldsymbol{t};R}\}\,\boldsymbol{y} \tag{20.31}$$

Running over all operators, the points $\bar{\boldsymbol{x}}(\{\boldsymbol{t};R\})$ are all images that should coincide with $\boldsymbol{x}$. The exactly symmetric points are computed as the mean positions

$$\boldsymbol{x}' = \frac{1}{N_G} \sum_{\{\boldsymbol{t};R\}} \bar{\boldsymbol{x}}(\{\boldsymbol{t};R\}) \tag{20.32}$$

N_G is the number of operators. It is easily verified that this definition yields a perfectly symmetric set. Consider the image of such a point

$$\{\overline{\boldsymbol{t};R}\}\,\boldsymbol{x}' = \frac{1}{N_G} \sum_{\{\boldsymbol{t};R\}'} \{\overline{\boldsymbol{t};R}\}\bar{\boldsymbol{x}}(\{\boldsymbol{t};R\}') \tag{20.33}$$

Let $\boldsymbol{z}$ be the approximate image of $\boldsymbol{x}$ under $\{\overline{\boldsymbol{t};R}\}$, then

$$\{\overline{\boldsymbol{t};R}\}\bar{\boldsymbol{x}}(\{\boldsymbol{t};R\}') = \bar{\boldsymbol{z}}(\{\boldsymbol{t};R\}\times\{\boldsymbol{t};R\}') \tag{20.34}$$

is one of the image points close to $\boldsymbol{z}$. Hence

$$\{\overline{\boldsymbol{t};R}\}\,\boldsymbol{x}' = \frac{1}{N_G}\sum_{\{\boldsymbol{t};R\}'}\bar{\boldsymbol{z}}(\{\boldsymbol{t};R\}\times\{\boldsymbol{t};R\}') = \frac{1}{N_G}\sum_{\{\boldsymbol{t};R\}''}\bar{\boldsymbol{z}}(\{\boldsymbol{t};R\}'') = \boldsymbol{z}' \qquad (20.35)$$

This is precisely one of the points in the output set (the corrected point $\boldsymbol{z}$), as required. In the second equality in (20.35) we used that the centralizing generators constitute a group.

It is assumed here that the operators themselves are exact (otherwise the product of two operators might not equal one of the operators exactly and hence the second equality in (20.35) would not be correct. As a consequence, if the operators have been determined by PNTGRP from the (possibly inaccurate) set of points, the symmetrizing cannot be done with these operators. It would be necessary to symmetrize first the operators themselves.

operators in the spherical harmonics representation

A point group operator R transforms a spherical harmonic $Z_{lm}(\Omega)$ into a linear combination of the $(2l+1)$ functions $Z_{lm'}(\Omega)$, $m'=-l\ldots l$. SYMZLM computes the matrix Z^l of that representation (More precisely, SYMZLM computes for a sequence of operators all Z_{lm} -representation matrices up to a maximum l-value). By spherical harmonics we understand here the *real*-valued spherical harmonics (23.30) as they are employed in BAND.

The matrix elements of Z^l are

$$Z^l_{mm'} = \int Z^*_{lm'}(\Omega)\; RZ_{lm}(\Omega)\; d\Omega = \int Z^*_{lm'}(\Omega)\; Z_{lm}(R^{-1}\Omega)\; d\Omega \qquad (20.36)$$

The integrand is an angular polynomial of degree $2l$, so that (20.36) can be evaluated exactly by numerical integration with a formula of the appropriate degree of precision. SPHPRD generates a product formula in $\cos\theta$ and ϕ of any desired degree. Let the points and weights of the formula be $\Omega_i \equiv (x,y,z)_i \;\; w_i$, $i=1..n$. The matrix elements (20.36) are then computed by

1. generate an integration formula of the appropriate degree: SPHPRD (or SPHPNT, the general routine for spherical integration).
2. calculate $Z_{lm}(\Omega_i)$, $i=1..n$ for all required (l,m): VZLM.
3. rotate all points to obtain $R^{-1}\,\Omega_i$. Since the operator R and the points $(x,y,z)_i$ are both in the coordinate representation, this is straightforward.
4. call routine VZLM again, now with the rotated points as input, to evaluate $Z_{lm}(R^{-1}\Omega)$, $i=1..n$.
5. compute $Z^l_{mm'}$ as

$$Z^l_{mm'} = \sum_i w_i\; Z^*_{lm'}(\Omega_i)\; Z_{lm}(R^{-1}\Omega_i) \qquad (20.37)$$

SYMZLM calculates $Z^l_{mm'}$ for all l-values $l=0..$lmax and for a sequence of noper operators. The result is one large array oprzlm. It contains successively for all operators the (trivial) 1×1 $l=0$ matrices, then for all operators the 3×3 $l=1$ matrices, and so on.

symmetry adapted functions

The symmetry property of the crystal integration grid allows the computation of all symmetry components of any function from the function values in the integration points. To construct in this way from a *set* of functions the combinations that transform as irreps, would be a time consuming procedure. It is easier to calculate the combination coefficients from the analytical properties of the functions.

It is natural to set up the irreps in a crystal in two steps. First we take a point $\boldsymbol{k}$ in the (first) BZ, by which we specify an irrep of the translation group. The generators $\{\boldsymbol{t};R\}$ of the *little group* of $\boldsymbol{k}$ (i.e. $R\boldsymbol{k}=\boldsymbol{k}$) are then used for a further reduction to the irreps of the space group. For a general point $\boldsymbol{k}$ there are no such operators (except the identity) and we are finished immediately. Many of the *k*-points that are used in an average calculation have more symmetry however, because they lie often on symmetry elements (reflection planes, rotation axes) in *k*-space; the central Γ-point $\boldsymbol{k}=0$ even has all generators in its group.

Let $\{\boldsymbol{t};R\}$ be the operators of the *k*-point under consideration. The centralizing operators $\{\overline{\boldsymbol{t};R}\}$ constitute a group and can be treated like any point group: analysis of the multiplication table to compute the classes, characters and irreps. (For the moment we neglect the translational symmetry $\boldsymbol{k}$).

Denote the matrices of the irreps by D^{μ}_{ij} , $i,j=1...n_{\mu}$; n_{μ} is the dimension of irrep μ. The projector $P^{\nu n}_{i}$, which picks up from a given function the component corresponding to irrep ν and column *n* is derived from the orthogonality theorem of the theory of groups

$$\sum_{\{\boldsymbol{t};R\}} D^{\mu}_{ij}(\{\boldsymbol{t};R\}^{-1})\; D^{\nu}_{kl}(\{\boldsymbol{t};R\}) = \frac{N_G}{n_{\nu}}\;\delta_{il}\;\delta_{jk}\;\delta_{\mu\nu} \tag{20.38}$$

N_G is the number of operators. The projector is

$$P^{\nu n}_{i} = \frac{n_{\nu}}{N_G}\sum_{\{\boldsymbol{t};R\}} D^{\nu}_{in}(\{\boldsymbol{t};R\}^{-1})\;\{\boldsymbol{t};R\} \tag{20.39}$$

i is an arbitrary row-index of the representation matrix. It is usually kept fixed for all column-indices *n*; this assures for instance that the hamiltonian matrices of all partner representation columns are identical.

Suppose now that we have *M* functions which span a reducible representation, and that we have the operators $\{\boldsymbol{t};R\}$ as $M\times M$ matrices in the representation of that function set. (20.39) gives then also the projector as an $M\times M$ matrix. This projector matrix has eigenvalues 0 and 1. The eigenvectors corresponding to the latter are the νn-adapted functions. The combination coefficients can thus be computed by diagonalization of the projector matrix. Equivalently they can be found by a Schmidt orthogonalization procedure: take subsequently every column of the projector matrix, orthogonalize it on those obtained before (and normalize). Some columns yield the zero vector; these singular solutions correspond to the eigenvalues zero and can be discarded.

(The apparent arbitrariness in the Schmidt orthogonalization procedure (that we may take as the first solution any non-zero column) corresponds to the fact that all non-singular eigenvectors are degenerate. Any linear combination is also a solution. So it is also arbitrary which *eigenvectors* are actually computed, although this may not be realized when one implements a call to some standard diagonalization routine).

The functions employed in BAND fall in two classes: spherical harmonics and plane waves. We discuss first the spherical harmonics. The plane waves will be treated essentially in the same way.

spherical harmonics

Consider a function $f^{\alpha}_{lm}(\boldsymbol{r}) = P(r_\alpha) Z_{lm}(\Omega_\alpha)$ centered on atom α. A symmetry operator $\{\boldsymbol{t};R\}$ maps $f^{\alpha}_{lm}(\boldsymbol{r})$ on a symmetry equivalent atom β, possibly α itself, and in addition it may rotate and/or reflect the orientation. The radial part $P(r)$ is not relevant here and can be omitted from the discussion. The degree l of the angular polynomial is not changed by any operator because symmetry operators are linear operators. The functions spanning a representation can thus be limited to $\{Z_{lm}(\Omega_\beta)\}^{\beta}_{m=-l..l}$, where β runs over the equivalent atoms. We deal with the translational symmetry later and restrict β to the atoms in the unit cell, N in number say. A spherical harmonic $Z_{lm}(\Omega_\alpha)$ generates then a representation of dimension $M=(2l+1)\times N$.

SYMZLM computes for the point group operators R the matrices $Z^{l}_{mm'}$ which are their Z_{lm} -representations (see elsewhere in this section). From this we construct $\{\boldsymbol{t};R\}$ as the $M\times M$ matrix. This matrix is in a natural way divided in $N\times N$ blocks of size $(2l+1)$ each. If $\{\boldsymbol{t};\overline{R}\}$ maps atom α on the equivalent atom β, then the corresponding matrix $Z^{l}_{mm'}$ occupies the (β,α)-block of the large matrix; the other blocks in the α-th block-column are zero. The complete $M\times M$ matrix is determined by filling the appropriate block in each block-column. Viewed as a $N\times N$ matrix with unity for the occupied blocks and zero otherwise, we have the operator as a permuation of the equivalent atoms in the unit cell.

Summation (20.39) yields the projection matrix $P^{\nu n}$, which may then be diagonalized, or treated by the Schmidt-orthogonalization method. This gives us all distinct νn-adapted functions.

translational symmetry

Incorporation of the translation symmetry is as follows. Denote by $c^{\nu n,l}_{\alpha m}$, $\alpha=1..N$, $m=-l..l$ the coefficients which define a νn-adapted function as discussed above. Let $f^{\alpha}_{lm}(\boldsymbol{r}) = P(r_\alpha) Z_{lm}(\Omega_\alpha)$ be a function to be symmetry adapted and $\boldsymbol{k}$ the k-point under consideration. The required crystal function is then

$$\Phi^{\boldsymbol{k},\nu n}_{l}(\boldsymbol{r}) = \sum_{\boldsymbol{T}} \sum_{m\alpha} c^{\nu n,l}_{\alpha m} f^{\alpha}_{lm}(\boldsymbol{r}-\boldsymbol{T}) e^{i\boldsymbol{k}\cdot(\boldsymbol{R}_\alpha+\boldsymbol{T})} = \sum_{m\alpha} c^{\nu n,l}_{\alpha m} e^{i\boldsymbol{k}\cdot\boldsymbol{R}_\alpha} \left\{ \sum_{\boldsymbol{T}} f^{\alpha}_{lm}(\boldsymbol{r}-\boldsymbol{T}) e^{i\boldsymbol{k}\cdot\boldsymbol{T}} \right\} \quad (20.40)$$

The term in brackets is the familiar bloch sum of the generating function $f_{lm}^{\alpha}(\boldsymbol{r})$; $\boldsymbol{R}_{\alpha}$ is the position of atom α. The procedure is thus straighforward: compute first the coefficients $\{c\}$ from the operators. Then, for each *k*-point: construct the bloch sums, calculate the phase-factors $e^{i\boldsymbol{k}\cdot\boldsymbol{R}_{\alpha}}$ and combine (20.40).

plane waves

The plane waves $e^{i(\boldsymbol{k}+\boldsymbol{K})\cdot\boldsymbol{r}}$ are characterized by a point $\boldsymbol{k}$ in the BZ and a reciprocal lattice vector $\boldsymbol{K}$. For a given $\boldsymbol{k}$ a representation is generated by the star of $\boldsymbol{K}$: all $\boldsymbol{K}$ that transform into each other under the operators *R*. The matrix *Z* that represents $\{\boldsymbol{t};R\}$ in this function set has elements $Z_{KK'}=e^{-i\boldsymbol{K}'\cdot\boldsymbol{t}}$ if $R\boldsymbol{K}'=\boldsymbol{K}$ and zero otherwise. Diagonalization of the projection matrix (20.39) (or Schmidt orthogonalization) gives the coefficients $c_{K}^{\nu n}$ for the symmetry adapted combinations

$$\Phi_{\boldsymbol{K}}^{\boldsymbol{k},\nu n}(\boldsymbol{r}) = e^{i\boldsymbol{k}\cdot\boldsymbol{r}}\sum_{\boldsymbol{K}'} c_{K}^{\nu n}\; e^{i\boldsymbol{K}'\cdot\boldsymbol{r}} \qquad (20.41)$$

The translation symmetry is incorporated automatically by the prefactor $e^{i\boldsymbol{k}\cdot\boldsymbol{r}}$.

fit functions

The fit functions for the charge density are combined into symmetry adapted functions. The only representation needed is the totally symmetric one; in particular the translation symmetry is $\boldsymbol{k}$=0.

The projector (20.39) for the totally symmetric representation is simple. The dimension of the representation is n_{ν} =1 and all representation matrices *D* are scalars, equal to 1.
ZLMPRJ, with auxiliary routine ZLMPR1 constructs the projector. Via a Schmidt orthogonalization all distinct symmetric functions are obtained from a given generating spherical harmonic fit function (plane wave fit functions are not employed in BAND).

Given the number of generating one-center fit functions for each type of atom and their *l*-values the total number of symmetric fit combinations can be computed. This is done in FITSYM to assist in the efficient workspace organization (SS^workspace). The coefficient vectors describing the fit combinations, calculated in ZLMPRJ, are used in FITPNT during the interpolation and bloch summation ($\boldsymbol{k}$=0) of the one-center functions to construct the symmetry adapted fit functions.

basis functions

Symmetry adaptation of the basis functions has not yet been implemented in BAND. Since that would increase the efficiency enormously it may be one of the future developments of the program; we spend some words here on a few aspects of a possible set-up.

1. The computation of the combination coefficients, which define the symmetry adapted functions in terms of the primitive one-center functions and their bloch sums, can proceed as described before, repeating the procedure for each *k*-point.

Since in each k-point the local symmetry group is a subgroup of the space group and all generators belong to the local group of $\boldsymbol{k}=0$, it is also possible to determine the combination coefficients only once, using all generators (i.e. we do it for the local group of $\boldsymbol{k}=0$). The coefficients can then be used in each *k*-point and define suitable functions (20.40 and 20.41). However it has then to be determined yet, which of the irreps of the $\boldsymbol{k}=0$ group are coupled in the *k*-point under consideration.

2. The gain in efficiency from a symmetry oriented basis set is twofold. First: in each *k*-point the hamiltonian $H^{\boldsymbol{k}}$ is automatically block-diagonalized. The off-diagonal blocks are zero on grounds of symmetry and we don't have to compute them in EIGSYS. Similarly, in the construction of the density (RHOPMT, see SS^charge density) the double loop over the basis functions is reduced to a sequence of shorter double loops: the P-matrices are also block-diagonal.
   The additional level in the multiple loop structures (the loop over the irreps) requires that we know the partition of the basis functions over the irreps and suggests that the basis functions of the respective irreps should be grouped together.

   The computation of the basis functions in BASPNT may be organized in analogy with the construction of the fit functions in FITPNT (with the symmetry coefficients from ZLMPRJ). For a proper dimensioning of arrays one could use routines like FITSYM and ZLMPRJ to determine in advance the number of symmetry adapted functions for each irrep. This information can be stored on file for usage by EIGSYS, RHOPMT etcetera.
3. The second enhancement of efficiency results from the numerical integration. When the functions correspond to irreps, a large number of integrals is zero on symmetry grounds and the remaining integrands are (or can be chosen to be) totally symmetric so that the integration has to run only over the symmetry unique points. This saves of course CP-time in the evaluation of the matrix elements. A similar improvement results for the construction of the density. Consequently we have to store on file only the function values in the generator points; this reduces also the demand on disc storage facilities, which is currently a bottleneck in the program.
4. A slight complication is that the irreducible wedge is a different thing for different *k*-points. For each *k*-point we have to determine anew which integration points are the symmetry unique points.
   Then, for the evaluation of the potential matrix elements (EIGSYS) we have to expand the potential, known in the truly unique points (i.e. symmetry unique for the complete space group), over the integration points used in that *k*-point. In the current set-up the potential has to be expanded over *all* integration points; this is done with help of the symmetry information on the points file itpnt: the points are organized in sets of equivalent points (SS^integration). In a symmetry oriented set-up we have to know analogously the relation between the truly unique and the locally unique point sets. Since this depends on the *k*-point we may need a different point file for each *k*-point (more probably we will have different sections on one point file, since the total amount of these data is limited).
   The same type of *k*-dependent information is also needed in the construction of the density (SS^charge density): From the P-matrix (RHOPMT), or from the hamiltonian eigenfunctions (RHOPSI) the density is computed in the points which are used for the basis functions; finally the density is contracted to the truly unique points (and symmetrized in the proces by averaging, see elsewhere in this section).

21 Temperature

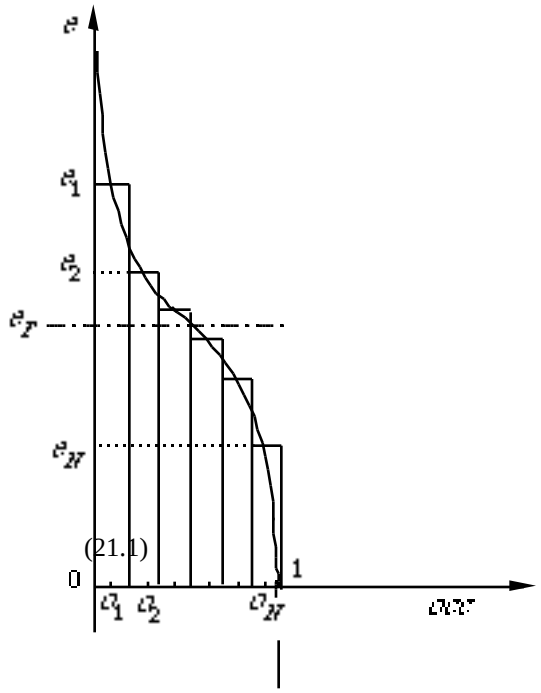

Fig.8. Fermi-dirac distribution with a Riemann-Stieltjes like integration.

Bandstructure calculations usually assume zero temperature. A finite temperature implies, among other things, a different distribution of electrons over the eigenstates. At T=0 the fermi energy e_F is a sharp boundary between fully occupied and completely empty states. This is the limiting case of the general fermi-dirac distribution at temperature T, giving the occupation of a state with energy e as

$$occ(e) = \frac{1}{1+e^{(e-e_F)/kT}} \qquad (21.1)$$

e_F is fixed by the condition that the summation of $occ(e)$ over all one electron states yield the total number of electrons. In finite systems, like molecules and clusters, the eigen energies are discrete. The determination of e_F and the occupation numbers is then straightforward. In bandstructure calculations this is more complicated as we have *bands* of eigenstates. Only states at a few discrete k-points in the Brillouin Zone are computed. These states may be thought to represent all states of the surrounding region in k-space and the occupation numbers are then associated with the fraction of represented states that is occupied. However, the occupations are *calculated* as weights belonging to a particular numerical integration scheme in k-space (SS^BZ-integration) and the values may be unexpected, even negative in some cases. Incorporation of a finite temperature in that integration procedure is not obvious. We approximate therefore the effect of the correct fermi-dirac distribution in another way.

Of course we are not interested in the occupation numbers themselves but in quantities that are computed by a summation over one-particle states. Let A be such a quantity and $A_n(\boldsymbol{k})$ the contribution from state $\psi_n(\boldsymbol{k};\boldsymbol{r})$, A is then computed as

$$A = \sum_{\boldsymbol{k}\,n} c_n(\boldsymbol{k})\ A_n(\boldsymbol{k}) \qquad (21.2)$$

The integration over k-space has been replaced by a summation over discrete k-points and index n runs over the bands. In case of the density for example $A=\rho(\boldsymbol{r})$ and $A_n(\boldsymbol{k}) = |\psi_n(\boldsymbol{k};\boldsymbol{r})|^2$. As mentioned above the employed numerical integration method in k-space implies that the coefficients $c_n(\boldsymbol{k})$ cannot simply be

associated with fermi-dirac distribution numbers. The true expression however, taking all infinitely many *k*-points into account, would be

$$A = \int_{BZ} \sum_{n} occ(e_n(\mathbf{k})) \, A_n(\mathbf{k}) \; d\mathbf{k} \tag{21.3}$$

where $occ(e)$ is given by (21.1).
The value of A depends on e_F and T via $occ(e)$ in (21.3), so $A=A(e_F,T)$.

Consider now the fermi-dirac distribution and imagine (fig.8) a Riemann-Stieltjes or similar integration scheme in the variable *occ* on the interval [0,1]. The points o_j , $0<o_1<o_2<\ldots<o_N<1$, have weights w_j such that w_j is the part of the occ-interval represented by o_j ; e_j is the energy value corresponding to o_j via equation (21.1): $o_j = occ(e_j)$. Hence (21.3) can be approximated by

$$A(e_F,T) \approx \sum_{j} w_j \; A(e_j,0) \tag{21.4}$$

Our *k*-space integration procedure yields coefficients $c_n(\mathbf{k})$ corresponding to $T=0$, but depending on the fermi energy: $c_n(\mathbf{k}) = c_n(\mathbf{k};e_F)$, so that A may be computed as

$$A(e_F,T) \approx \sum_{j} w_j \; A(e_j,0) \approx \sum_{j} w_j \sum_{\mathbf{k}\,n} c_n(\mathbf{k};e_j) \; A_n(\mathbf{k}) \equiv \sum_{\mathbf{k}\,n} \bar{c}_n(\mathbf{k}) \; A_n(\mathbf{k}) \tag{21.5}$$

$$\bar{c}_n(\mathbf{k}) = \sum_{j} w_j \; c_n(\mathbf{k};e_j) \tag{21.6}$$

The final coefficients $\bar{c}$ are thus defined as numerical integrals. The integration variable is *occ,* on the interval [0,1], so (21.6) has to be read as

$$\bar{c}_n(\mathbf{k}) = \sum_{j} w_j \; c_n(\mathbf{k};e(o_j)) \equiv \sum_{j} w_j \; c_n(\mathbf{k};o_j) \tag{21.7}$$

where $e(occ)$ is the inverse function of (21.1).

The accuracy of the result depends of course on the functional dependence of the coefficients *c* on *occ* and on the employed numerical integration method. One would like to use some high precision method like Gauss-Legendre quadrature. There are two types of problems. In the first place *c* may have discontinous derivatives at *occ*-values, for which $e(occ)$ is a band-edge or another van-Hove energy. This may be remedied by splitting the interval [0,1] in appropriate subintervals and using separate numerical integrations for each of them. In the second place the energy $e(occ)$ has infinite derivatives for the end-values 0 and 1; the same may be true for the coefficients *c,* at *occ*=0 or 1, as well as at band edges, so that some variable transformation may be needed to make Gauss-Legendre quadrature applicable.

At the current state of affairs BAND uses a straightforward Legendre integration without bothering about these problematic aspects.

The temperature *T* and the number of integration points (nfdirc in the program) are input via the keys `temp` and `fdirc` (or `fermi-dirac`) respectively; default values are nfdirc=2 and *T*=10. The latter has the effect of *T*=0 on the final results: in energy units a temperature of one degree equals approximately 3×10^{-6} a.u. so that on the scale of interest (10^{-3} a.u.) a temperature of 10 degrees gives virtually a step function for the fermi-dirac distribution.
The temperature is in BAND represented by the variable tfdirc; its numerical value is the temperature in atomic energy units (*input* is in degrees however).
When the SCF procedure has convergence problems, detected in SCFTST (SS^iteration), nfdirc is increased by 2. Convergence problems are often related to near degeneracies around the fermi-level; a more detailed approximation of the correct distribution of occupation numbers over the involved states may facilitate the iterative procedure (but usually this does not help much).

22 Workspace

dynamical allocation of arrays

BAND needs many arrays to store data: the coordinates of the atoms, the characteristics of the basis functions, the values of functions in the integration points and so on. Fixed sizes for all these arrays are unwieldy. With any configuration of fixed sizes we will soon encounter an application that does not fit, while just a redistribution of the claimed workspace would suffice. This would then necessitate a recompilation of the program with adapted array bounds. Fixed sizes imply also that in every run we pay for memory we do not use.

FORTRAN77 does not support dynamical array allocation. BAND simulates dynamical allocation of arrays with subroutine ARRAYS. We will refer to this routine and the related data structures as BAND's *workspace manager*.
The total workspace available is fixed of course (FORTRAN77) and consists of two large arrays in (blank) common, one for integers, iwork, and one for reals, rwork. Whenever an array is needed in the course of the program execution ARRAYS is called. It gives back which element of iwork or rwork respectively is to be used as starting address. Later it may be called to release the claimed space again, when the allocated array is not needed anymore.
A typical structure would be

```
n = ..
call arrays ('allocate','real','store',n,ii)
call b(n,rwork(ii),....)
..
call arrays ('delocate','real','store',....)                (22.1)
end

subroutine b (n,store,....)
dimension store(n)
..
```

The allocated arrays are identified by a name, which is stored by ARRAYS in a list of names, together with their positions in iwork or rwork. Between each pair of allocated arrays in iwork or rwork one word is kept free. ARRAYS writes a *marker* there: a specific integer or real value that is improbable to occur often. As soon as ARRAYS is called to give the space free again, it checks whether the markers directly before and after the array are intact: global array-bounds checking afterwards.

ARRAYS discriminates two types of data: integers (type 1) and reals (type 2). Array arname(nnames,2) contains the names of the currently allocated arrays. The maximum number per type, nnames, is a constant

in the program. Indexw contains the locations of the arrays: indexw(i,j) is the position of the marker before the *i*-th array of type *j* (*j*=1,2). ncwork stores how many arrays are currently allocated: the number of allocated arrays of type *j* is (ncwork(j)−1). The relevant data are stored in the common blocks CNTRLC (data of type 'character') and CNTRLV (other variables).

One of the arguments of ARRAYS is action, a string to identify the purpose of the call. action may have the values 'allocate' or 'delocate' with obvious meanings. Furthermore it may be

'available' to find out how much space is still free,

'space' to allocate an array as large as possible. This is a combination of 'available' and 'allocate'; output are the pointer for the array and its size.

'check' to check the markers around a specific array,

'init' to initialize the information structure in ARRAYS (in this way it is called only once of course, by INIT),

'dump' to output the current state of affairs,

'free', which implies 'delocate' for the specified array *and* for all arrays of the same type that were allocated after it.

remarks

1. If several arrays have been allocated, their de-allocations may be executed in any order. However, the space of a delocated array is not available for subsequent allocations until all arrays after it have been given free as well: free holes in iwork and rwork are not used, only the free final sections.
2. ARRAYS is also used to manage array allocation for logicals and complex numbers. Logicals are treated on the same footing as integers; complex data are treated as reals, taking two real words for each complex. The size of an array as specified in the argument list is therefore multiplied by *m* when sizes are judged; *m*=1 in all cases except for complex numbers, where *m* equals 2.
3. It is possible in principle to have only one workarray in blank common and allocate both integer and real arrays in it; this might be more efficient as we may now have a too small real array rwork and wasted space in iwork, or vice versa. The gain would be small, because the real array is an order of magnitude larger so that some extra space from the integer array is not very important.
   The reason to keep integers and reals separate is that type-mixing may cause problems on some machines. The same might of course be true as regards the mixing of logicals with integers and complex numbers with reals. In that case the number of types used in ARRAYS should be extended; the necessary adaptations will be obvious from inspection of the implementation.

optimized vector lengths

Even with the optimized use of available workspace, by way of the pseudo-dynamical allocation procedure just discussed, it is impossible to have all data in core simultaneously. BAND evaluates most integrals numerically. The relatively large number of integration points necessitates the storage of data on file, such as the values of the basis functions in the points. Each time we need them, for instance to compute matrix elements of the potential, they are read and used. In general only part of these data can be kept in core at the

same time and the numerical integration has to be performed in a number of steps. At each step the values corresponding to one block of points are read, processed in the computation and then replaced by the next block of data.

The heart of the computation consists of vector operations; the length of the vector is the number of points in a block. It is advantageous to have the vectors as long as possible or, to be more precise, to have as few blocks as possible, because each extra block costs one more start-up of the vector operation.

For organizational reasons the block structure is the same on all involved files. By the block structure we understand here the number of blocks (of points) and the length of each individual block. In each routine where we have to handle such data the available workspace is determined by what other data must currently be kept in core, so that for each routine there may be a different maximum possible vector length. Routine WRKORG computes these values for all sections in the preparation stage and gives back the minimum over them: the absolute maximum vector length for the whole preparation part: npx. This can then be used to write the various data to file in the appropriate block structure.

As regards the SCF part alone, the maximum possible vector length is often much larger than in the preparation stage: in the preparation part we need at certain moments the overlap matrices of the valence, core, or fit functions; these occupy considerable parts of the total workspace. REORGF computes the SCF vector length and reorganizes the relevant files after PREPAR and before SCF; auxiliary routines for REORGF are REORG1, -2 and -3; each of these rewrites a (number of) specific files.

The determination of the block structure for PREPAR has two complications.
a) In BASPNT the bloch sums of the basis functions are computed for all employed *k*-points of the Brillouin Zone. Much of the involved computational work does not depend on the *k*-point. Hence it seems obvious to do this for as many *k*-points together as possible, instead of repeating the work for each *k*-point. However this implicates that more basis function data have to be kept in core (proportional to the number of *k*-points processed simultaneously), thereby reducing the maximum possible vector length in BASPNT; this again makes the execution more expensive by increasing the cost of the vector operations.

To achieve the optimal situation we have to minimize some sort of cost function. This function has to depend both on the vector length and on the number of *k*-points handled together. The optimal balance may be determined by many conditions and not in the least by the machine; we find for example that the optimal situation on the Cyber205 differs markedly from that on the Cyber995.
We have implemented a cost function (in RPNTRE), but the form will not be discussed here: it is only a first guess; a systematic experimentation with it might very well yield a significant improvement in performance of the program.
The number of *k*-points treated together is the variable kgrp. By default it is determined by RPNTRE, but this may be overruled by fixing its value via input, with key `kgrp`.

b) For some functions, in particular the fit functions, we need only the values in the *symmetry unique* points. The maximum vector length according to the fit section is thus the maximum number of symmetry unique points in a block. The absolute maximum vector length can then be computed if we know the ratio between the symmetry unique points and the total number of points in a block. However this is not a fixed value (e.g. the number of symmetry operators): some points may not be *general* points. If they are located on a symmetry element, such as a reflection plane or a rotation axis, the number of equivalent points is only a fraction of the number of symmetry operators. So the ratio depends on what kind of points are in a particular block. Then, *assuming* a quotient (the variable symfrc in the program) it must be seen afterwards whether the actual value in the realized block structure does not deviate so much from it as to cause problems; RPNTRE checks this and adapts in such (exceptional) cases the block structure.

WRKORG is called first from PREPAR to obtain a first assessment of the maximum vector length npx; symfrc has been initiated (INIT) at the safe value 1.0. In RPNTID the file with integration points itpnt, output from POINTS, is rewritten in accordance with the value of npx; meanwhile the maximum fraction of symmetry unique points, symfrc, is determined. Then, in RPNTRE, the cost function mentioned above and the new value of symfrc are used to optimize the points-file. The actual rewriting of itpnt occurs in the auxiliary routine RPNTRW.

23 XC: exchange and correlation

The exchange and correlation (XC) interactions between the electrons are approximated with the density functional (DF) formalism. Many DF formulas have been advocated in the literature for the XC potential. The first proposal was by Slater [Slater 1951], later parametrized to the famous X_α potential. More recently the Gunnarson-Lundqvist (GL) [Gunnarsson and Lundqvist 1976] and the Vosko-Wilk-Nusair (VWN) [Vosko *et al.* 1980] form were published. All these approximations are based on homogeneous electron gas calculations. BAND has the possibility to take either of these. The VWN function is presumably the most accurate, but for sake of comparison the other options are occasionally useful.

Even the more sophisticated DF functions, like VWN, are rather severely in error, depending on the (poly-) atomic system. The error is roughly 10% in the exchange part and up to 100% in the correlation [Stoll *et al.* 1978, Gunnarsson and Jones 1985, Becke 1986]. The signs of these deviations are opposite and the sizes such that the net result, as regards the self consistent charge density and the total energy, is often fairly good. Not in all cases however and from a more principal point of view this situation is of course unsatisfactory.
According to Stoll *et al.* [1978] the correlation error is mainly due to an overestimation of the correlation between electrons of the same spin. The program has therefore the option to suppress this term, partially or completely. To prevent the total result from getting worse we should of course also repair the exchange error. That can be done to a large extent by including a non-local term, a function of the *gradient* of the density [Becke 1988]. Implementation in BAND has not yet been undertaken. It is not very difficult and will be discussed at the end of this section.

local XC functions

The energy density and potential will be denoted by $\varepsilon(\boldsymbol{r})$ and $\mu(\boldsymbol{r})$ respectively.
Define:

$\rho^+(\rho^-)$ is the charge density of spin-up (-down) electrons. (23.1)

$\rho = \rho^+ + \rho^-$

$\zeta = (\rho^+ - \rho^-) / \rho$ is the spin polarization function

$f(\zeta) = \left((1+\zeta)^{4/3} + (1-\zeta)^{4/3} - 2 \right) / \left(2^{4/3} - 2 \right)$

$r_s = \left(\frac{3}{4\pi\rho} \right)^{1/3}$ is the Wigner radius

$\gamma = \left(\frac{4}{9\pi} \right)^{1/3}$

$$E_{XC} = \int d\boldsymbol{r}\, \rho(\boldsymbol{r})\, \varepsilon_{XC}(\boldsymbol{r})$$

The energy density and the potential derived from it can conceptually be split into an exchange and a correlation part

$$\begin{aligned} \varepsilon_{XC} &\equiv \varepsilon_{XC}(r_s,\zeta) = \varepsilon_X + \varepsilon_C \\ \mu^{\pm}_{XC}(r_s,\zeta) &= \frac{d}{d\rho^{\pm}}(\rho\,\varepsilon_{XC}) = \varepsilon_{XC} - \frac{r_s}{3}\frac{d\varepsilon_{XC}}{dr_s} \pm (1\mp\zeta)\frac{d\varepsilon_{XC}}{d\zeta} = \mu_X + \mu_C \end{aligned} \qquad (23.2)$$

The exchange part is given for all local spin density functionals by

$$\varepsilon_X = \left(\frac{-3}{8\pi\gamma r_s}\right)\left((1+\zeta)^{4/3} + (1-\zeta)^{4/3}\right) \quad \left(= \frac{-3}{2}\left(\frac{3}{4\pi}\right)\frac{(\rho^+)^{4/3} + (\rho^-)^{4/3}}{\rho}\right) \qquad (23.3a)$$

$$\mu^{\pm}_X = \left(\frac{-1}{\pi\gamma r_s}\right)(1\pm\zeta)^{1/3} \quad \left(= -2\left(\frac{3}{4\pi}\right)(\rho^{\pm})^{1/3}\right) \qquad (23.3b)$$

The difference between the various XC functions is the treatment of the correlation.

X-alpha

In the X_α approximation the correlation part is represented by parametrizing the exchange term

$$\mu_{XC} = \frac{3}{2}\,\alpha_X\,\mu_X \qquad (23.4)$$

and similarly for ε_{XC} . The X_α parameter α_X is usually taken between 0.6 and 1.0.

Gunnarsson-Lundqvist

$$\varepsilon_C = \varepsilon^p_C + (\varepsilon^f_C - \varepsilon^p_C)\, f(\zeta) \qquad (23.5)$$

p and *f* indicate the para- and ferro magnetic states. (ζ=0,1).

$$\begin{aligned} \varepsilon^j_C &= -c_j \left\{((1+\xi_j^3)\log(1+1/\xi_j) + 1/2\,\xi_j - \xi_j^2 - 1/3\right\} \\ \xi_j &= r_s/d_j \qquad j=p,f \end{aligned} \qquad (23.6)$$

c_j and d_j are constants

$$\begin{aligned} c_p &= 0.0333 \text{ a.u.} & c_f &= 0.0203 \text{ a.u.} \\ d_p &= 11.4 \text{ a.u.} & d_f &= 15.9 \text{ a.u.} \end{aligned} \qquad (23.7)$$

Vosko-Wilk-Nusair

$$\varepsilon_C = \varepsilon^p_C(r_s) + 9/8\,(2^{4/3}-2)\,\alpha_C(r_s)(1-\zeta^4)\; f(\zeta) + (\varepsilon^f_C(r_s) - \varepsilon^p_C(r_s))\,\zeta^4\; f(\zeta) \qquad (23.8)$$

The values of the parameters ε_C^p , ε_C^f and α_C are all given by a function $H(r_s,x_i,B_i,C_i,Q_i,CA_i,CB_i,CC_i)$. The arguments are different for the three cases, $i=p,f,\alpha$.

$$H(r_s,x,B,C,Q,CA,CB,CC) = CA\left\{\log(r_s) - 2CB\log\left(\sqrt{r_s}-x\right) + \right. \tag{23.9}$$

$$\left. + (CB-1)\log\left(r_s+B\sqrt{r_s}+C\right) + CC\arctan\left(\frac{Q}{2\sqrt{r_s}+B}\right)\right\}$$

The constants x,B,C,Q,CA,CB and CC are given in table II.

The implementation of the function H needs a little care. As the density approaches zero several terms in (23.9) become unbounded. The 'infinities' cancel each other analytically. To prevent rounding errors and other numerical problems from influencing the result a few terms are recombined. Set $g=(3/4\pi)^{1/3}$, $r_s = g\rho^{-1/3}$:

$$H = CA\left\{\log(g) + \log\left(\rho^{-1/3}\right) - 2CB\left(\sqrt{g}-x\rho^{1/6}\right) - 2CB\log\left(\rho^{-1/6}\right) + \right.$$

$$\left. + (CB-1)\log\left(\rho^{-1/3}\right) + (CB-1)\log\left(g + B\sqrt{g}\rho^{1/6} + C\rho^{1/3}\right) + CC\arctan\left(\frac{Q\rho^{1/6}}{2\sqrt{g}+B\rho^{1/6}}\right)\right\} =$$

$$= CA\left\{\log(g) - 2CB\left(\sqrt{g}-x\rho^{1/6}\right) + (CB-1)\log\left(g + B\sqrt{g}\rho^{1/6} + C\rho^{1/3}\right) + \right.$$

$$\left. + CC\arctan\left(\frac{Q\rho^{1/6}}{2\sqrt{g}+B\rho^{1/6}}\right)\right\} \tag{23.10}$$

Stoll's correction for the correlation

As mentioned before it may be argued [Stoll *et al.* 1978] that the commonly applied DF formalism, based on the homogeneous electron gas, overestimates the correlation. They propose a correction yielding an effective correlation function

$$\varepsilon_C^{eff}(r_s,\zeta) = \varepsilon_C(r_s,\zeta) - \varepsilon_C^{correc}(r_s,\zeta) \tag{23.11}$$

$$\varepsilon_C^{correc}(r_s,\zeta) = \tfrac{1}{2}(1+\zeta)\ \varepsilon_C(r_s^+,\zeta=+1) + \tfrac{1}{2}(1-\zeta)\ \varepsilon_C(r_s^-,\zeta=-1) \tag{23.12}$$

$$r_s^{\pm} = \left(\frac{3}{4\pi\rho^{\pm}}\right)^{1/3} \tag{23.13}$$

The corresponding effective correlation potential is

$$\mu_C^{eff\ \pm} = \mu_C^{\pm} - \varepsilon_C(r_s^{\pm},\zeta=\pm1) - \rho^{\pm}\frac{d}{d\rho_{\pm}}\ \varepsilon_C(r_s^{\pm},\zeta=\pm1) \tag{23.14}$$

BAND applies the correction with an (input specified) multiplication factor λ between zero and one.

$$\varepsilon_C^{eff} = \varepsilon_C - \lambda\, \varepsilon_C^{correc} \tag{23.15}$$

implementation

The local density functional (LDF) formulas above are evaluated by two main routines: XCENER and XCPOT for the energy density and potential respectively. Input are a vector of density values and information regarding the type of DF formula. The GL form is not yet available because the evaluation of the energy density has not been implemented thusfar.

	p	*f*	α
x	−.10498	−.325	−.0047584
B	3.72744	7.06042	1.13107
Q	6.15199	4.73093	7.12311
CA	.0310907	.01554535	−.01688685
CB	−.0311676	−.144601	−.000414034
CC	1.24742	3.3766	.317708

Table II. Constants in the VWN formula for the correlation functional.

A few auxiliary routines compute various (parts of the) formulas occurring in the VWN approximation. These are XCVWND, XCVWNE, XCVWNF, XCVWNP.
The variable ioptxc determines the type of formula (1:X_α, 2:VWN, 3:GL); xcpar provides additional information: for the X_α formula it is the usual X_α parameter, in the other cases it is the amount of Stoll-correction applied (λ in 23.15). Default is the VWN formula without Stoll's correction: ioptxc=2, xcpar=0. Input is with the keys `X-alpha`, `VWN` or `Stoll`. The last two request both the VWN formula to be applied. In all three cases the key record may contain a value for xcpar; defaults respectively 0.7, 0, 1.

gradient correction for the exchange

In recent years considerable success has been met in the elimination of the exchange error by inclusion of a term that depends on the *gradient* of the density. Becke [1988] gives a function for the exchange energy

$$E_X = E_X^{LDA} - \beta \sum_\sigma \int \rho_\sigma^{4/3} \frac{x_\sigma}{1+6\beta x_\sigma \sinh^{-1} x_\sigma}\, d\mathbf{r} \tag{23.16}$$

E_X^{LDA} is the exchange energy in the local density approximation, given by (23.1) and (23.3), σ is the spin index, β is a constant whose value may be fitted to the exact Hartree Fock exchange energies for atoms (β≈0.0042 a.u.) and the dependence on the gradient is via

$$x_\sigma \equiv \frac{|\nabla \rho_\sigma|}{\rho_\sigma} \tag{23.17}$$

Incorporation of this function is straightforward if the density gradient can be evaluated in the integration points. The density in the crystal is known as an expansion in basis functions $\phi_i(\boldsymbol{k})$

$$\rho = \sum_{\boldsymbol{k}} \sum_{ij} P_{ij}^{\boldsymbol{k}} \; \phi_i(\boldsymbol{k}) \; \phi_j(\boldsymbol{k}) \tag{23.18}$$

$P_{ij}^{\boldsymbol{k}}$ is the density matrix. For the gradient we have then

$$\rho = \sum_{\boldsymbol{k}} \sum_{ij} P_{ij}^{\boldsymbol{k}} \left(\phi_i(\boldsymbol{k}) \; \phi_j(\boldsymbol{k}) + \phi_i(\boldsymbol{k}) \; \phi_j(\boldsymbol{k}) \right) = \sum_{\boldsymbol{k}} \sum_{ij} \left(P_{ij}^{\boldsymbol{k}} + P_{ji}^{\boldsymbol{k}} \right) \; \phi_i(\boldsymbol{k}) \quad \phi_j(\boldsymbol{k}) \tag{23.19}$$

The basis functions are either plane waves $e^{i\boldsymbol{k}\cdot\boldsymbol{r}}$ giving $e^{i\boldsymbol{k}\cdot\boldsymbol{r}} = i\boldsymbol{k}\cdot e^{i\boldsymbol{k}\cdot\boldsymbol{r}}$, or bloch sums of one center functions χ_i

$$\phi_i(\boldsymbol{k};\boldsymbol{r}) = \sum_{\boldsymbol{R}} \chi_i(\boldsymbol{r}-\boldsymbol{R}) \; e^{i\boldsymbol{k}\cdot\boldsymbol{R}} \tag{23.20}$$

The one center functions have the form

$$\chi(\boldsymbol{r}) = Z_{lm}(\Omega) \; P(r) \tag{23.21}$$

where the radial function $P(r)$ is an exponential function $P(r) = r^{l+k} \, e^{-\alpha r}$ or a numerical orbital from the free atom subprogram DIRAC. In both cases the gradient χ_i and hence $\phi_i(\boldsymbol{k})$ can be computed.

For the evaluation of the gradient of the density we need in this formulation the derivatives of the basis functions $\phi_i(\boldsymbol{k})$. These values, in each integration point three numbers per basis function, will have to be stored on file. In the current set-up of BAND the data of the basis functions themselves are already a major bottleneck for the application to large systems so that this approach poses a serious problem. It may therefore be advisable to employ the fit functions f_i that are also used for the evaluation of the coulomb potential. The density is approximated by

$$\rho = \sum_{\alpha} \rho_\alpha + \rho_{def} \approx \sum_{\alpha} \rho_\alpha + \sum_{i} c_i \, f_i \tag{23.22}$$

ρ_α is the spherically symmetric charge density of free atom α, ρ_{def} is the deformation charge

$$\rho_{def} \equiv \rho_{crystal} - \sum_{\alpha} \rho_\alpha \tag{23.23}$$

The fit functions are bloch sums (23.20) (with $\boldsymbol{k}$=0) of one center functions of the form (23.21). The total number of one center functions that are used for the fit is of the same order as the number of one center functions for the valence basis. The latter are combined in bloch sums for each k-point, the fit functions however only for $\boldsymbol{k}$=0. Furthermore BAND employs as fit functions only the totally symmetric combinations of one-center functions (SS^symmetry), while for the valence basis of course all irreps are used. So evaluation of ρ via (23.22), instead of (23.19) requires less additional data to be stored; in most cases the difference will at least be an order of magnitude.
A second advantage is the computing time. Expression (23.22) leads to

$$\rho \approx \sum_{\alpha} \rho_{\alpha} + \sum_{i} c_i \; f_i \tag{23.24}$$

The striking difference with (23.19) is that not only the summation over the *k*-points is absent, but that also the double summation over the functions has been replaced by a single sum.

derivatives of one-center functions

We conclude this section with the derivatives of one center functions of the form (23.21). It will be convenient to express the desired derivatives with respect to the cartesian coordinates in spherical coordinates. So we use

$$\mathrm{d}/\mathrm{d}x = \partial r/\partial x \; \mathrm{d}/\mathrm{d}r + \partial\phi/\partial x \; \mathrm{d}/\mathrm{d}\phi + \partial\theta/\partial x \; \mathrm{d}/\mathrm{d}\theta \tag{23.25}$$

and similarly for d/d*y* and d/d*z*, with

$$r = \sqrt{x^2+y^2+z^2}\,, \quad \theta = \mathrm{acos}(z/r), \quad \phi = \arctan(y/x) \tag{23.26}$$

to write

$$\begin{pmatrix} \mathrm{d}/\mathrm{d}x \\ \mathrm{d}/\mathrm{d}y \\ \mathrm{d}/\mathrm{d}z \end{pmatrix} = \begin{pmatrix} \cos\phi\sin\theta & \cos\phi\sin\theta/r & -\sin\phi/r\sin\theta \\ \sin\phi\sin\theta & \sin\phi\cos\theta/r & \cos\phi/r\sin\theta \\ \cos\theta & -\sin\theta/r & 0 \end{pmatrix} \begin{pmatrix} \mathrm{d}/\mathrm{d}r \\ \mathrm{d}/\mathrm{d}\theta \\ \mathrm{d}/\mathrm{d}\phi \end{pmatrix} \tag{23.27}$$

For an exponential function $P(r)=r^n\, e^{-\alpha r}$ the radial derivative is

$$\frac{\mathrm{d}}{\mathrm{d}r}\; r^n \; e^{-\alpha r} = \left(n/r - \alpha\right)\; r^n \; e^{-\alpha r} \tag{23.28}$$

If $P(r)$ is a numerical function, given as a table $P(\mathrm{r}_i)$ with logarithmically increasing r_i , $r_{i+1}\,/r_i = c$, the derivative can be computed numerically by a 3-point interpolation. Define $\mathrm{g}(\mathrm{r}_i) \equiv \mathrm{R}(\mathrm{r}_i)/\mathrm{r}_i$, then

$$\frac{\mathrm{d}P(r_i)}{\mathrm{d}r} = \begin{cases} \frac{1}{c^2}\left((c+1)^2 g_2 - (c+2) g_1 - c g_3\right) & i=1 \\ g_i + \frac{c}{c^2-1}\left(g_{i+1} - g_{i-1}\right) & i=2..N-1 \\ \frac{1}{c^2-1}\left(g_{N-2} + (2c^2+1) g_N - (c+1)^2 g_{N-1}\right) & i=N \end{cases} \tag{23.29}$$

derivatives of spherical harmonics

The angular derivatives operate on the spherical harmonics. BAND uses real-valued spherical harmonics

$$Z_{lm}(\Omega) = \begin{cases} c_l^{|m|}\; P_l^{|m|}(\cos\theta)\,\cos(m\phi) & m\le 0 \\ c_l^{m}\; P_l^{m}(\cos\theta)\,\sin(m\phi) & m>0 \end{cases} \tag{23.30}$$

P_l^m is the associated Legendre function and c_l^m is a normalization factor ($m \geq 0$)

$$c_l^m = (-)^m \sqrt{\frac{(2l+1)}{4\pi} \frac{(l-m)!}{(l+m)!}} \tag{23.31}$$

The derivative with respect to ϕ is obvious. For $dZ_{lm}/d\theta$ the recurrence relations and the definitions of the Legendre functions can be used [Arfken 1970].

$$dP_l^m/d\theta = -\sin\theta \, dP_l^m/d\cos\theta = \frac{1}{2}\left((l+m)(l-m+1)P_l^{m-1} - P_l^{m+1}\right) \tag{23.32}$$

From the relation to the Legendre polynomials [Arfken 1970]

$$P_l^m = \sin^m \theta \left(d/d\cos\theta\right)^m P_l \tag{23.33}$$

we may alternatively derive

$$dP_l^m/d\theta = \left(m/\tan\theta\right) P_l^m - P_l^{m+1} \tag{23.34}$$

The first term in (23.34) presents a problem for $\sin\theta=0$. For $m=0$ this term should be omitted (23.33), while for $m \neq 0$ P_l^m contains a factor $\sin^m \theta$ removing the singularity. In view furthermore of the restriction $\hat{o} m \hat{o} \leq l$ a possible set-up is

1. use explicit expressions for the derivatives of Z_{lm} with $l=0,1$
2. for $l \geq 2$, $m=-l+1,....,l-1$ use (23.32)
3. for $m=-l$ use (23.34)
4. for $m=l$ combine (23.32) and (23.34) to eliminate P_l^{m+1} :

$$dP_l^m/d\theta = (l+m)(l-m+1) \, P_l^{m-1} - \left(m/\tan\theta\right) P_l^m \tag{23.35}$$

For $\theta=0,\pi/2$ the first term in (23.34), respectively the second term in (23.35) have to be omitted.

Software Reference List A: subroutines

: