REPORT

High Performance Computing in Material Sciences

Higher Level BLAS in Symmetric Eigensolvers

Wilfried N. Gansterer, Dieter F. Kvasnicka,
Christoph W. Ueberhuber

AURORA TR1998-18
November 30, 1998

Special Research Program SFB "AURORA" funded by the Austrian Science Fund (FWF),
AURORA Groups 5 and 8

High Performance Computing
in Material Sciences

Higher Level BLAS in Symmetric Eigensolvers

Wilfried N. Gansterer[1], **Dieter F. Kvasnicka**[2],
Christoph W. Ueberhuber[1]

[1] Institute for Applied and Numerical Mathematics, TU Vienna
ganst@aurora.tuwien.ac.at, christof@uranus.tuwien.ac.at

[2] Institute for Technical Electrochemistry, TU Vienna
dieter@hermes.theochem.tuwien.ac.at

AURORA TR1998-18

November 30, 1998

The work described in this report was supported by the Special Research Program
SFB F011 "AURORA" of the Austrian Science Fund (FWF).

Abstract

In this report a way to apply high level BLAS to the tridiagonalization process of a symmetric matrix A is investigated. Tridiagonalization is a very important and work-intensive preprocessing step in eigenvalue computations. It also arises as a very central part of the material sciences code WIEN 97 (Blaha et al. [12]). After illustrating the drawbacks and limitations of the tridiagonalization implemented in LAPACK (Anderson et al. [1]), the dependency structure in the tridiagonalization process is analyzed. It is shown that data references cannot be localized for tridiagonalization in one sweep and that Level 2 operations cannot be avoided in a one-sweep tridiagonalization algorithm. Thus a different method is considered, by which the inherent potential for using higher level BLAS can be exploited. By performing tridiagonalization in two successive sweeps, it is possible to utilize Level 3 BLAS in the first sweep, which transforms A into a band matrix B. In a second sweep, B is transformed to a tridiagonal matrix T. The second sweep represents only a very small portion of the overall work, and for this reason the new method yields very good performance. Investigation of the backtransformation process for the corresponding eigenvectors reveals that it is preferable to calculate the eigenvectors from the band matrix B and not from the tridiagonal matrix T.

Contents

Chapter 1

Introduction

This report describes some results of the collaboration between project groups 5 and 8 in the first year of the *Spezialforschungsbereich AURORA*. The goal was to improve the efficiency of the standard eigensolver in the WIEN package (Blaha et al. [12, 13]). Previous work on the preceding step—the reduction of the generalized eigenvalue problem to a standard problem—is summarized in another report (Haunschmid and Kvasnicka [27]).

After a short overview of the chemical and theoretical background for the methods used in the WIEN package and a precise problem statement in Chapter 2, relevant concepts, terminology, and notation from mathematics and computer science are introduced in Chapter 3.

Chapter 4 focusses on the tridiagonalization process, which is a very important step in the eigenvalue calculations in WIEN 97. First, the blocked LAPACK approach (Anderson et al. [1]) is illustrated in detail, and its advantages and disadvantages are carefully investigated. Motivated by this discussion the dependencies in a similarity transformation, which is the basic step of the tridiagonalization process, are illustrated. It is concluded that full blocking utilizing Level 3 BLAS is not possible when tridiagonalizing in one sweep.

As an alternative the concept of tridiagonalization in *several sweeps*, which was suggested in Bischof et al. [6] (see also Bischof and Sun [9], Bischof et al. [7]) is discussed. This concept makes it possible to do a big portion of the work in a *fully blocked* first sweep with localized data access patterns. However, this first sweep reduces the matrix only to a banded form. Tridiagonal form cannot be achieved directly any more—the dependencies causing bad data-locality and thus low performance are left for later sweeps. Since these later sweeps operate on significantly smaller data, overall performance can be improved.

Multi-sweep tridiagonalization introduces additional transformation steps. Therefore, Bischof et al. [6] concluded that a multi-sweep approach is not competitive when eigenvectors are to be computed as well (despite the improved data-locality). However, in this report a modification of the multi-sweep eigensolver is suggested which is in many cases superior to the one-sweep approach—even when eigenvectors have to be computed.

When the tridiagonalization process is blocked, several single Householder transformations have to be combined to apply the corresponding updating operations to the rest of the matrix. Chapter 5 gives an overview of the most important ways to combine several Householder transformations and to represent them in a compact form.

Chapter 6 summarizes plans for future research. Motivated by the new approach from Chapter 4, efficient ways of computing eigenvectors of band matrices are investigated. In addition to a tridiagonalization-based method, a block Davidson method (see Wood and Zunger [44]) is used in WIEN 97. In many cases this alternative method does not show satisfactory convergence behavior. However, it seems to have the potential to be more efficient than the tridiagonalization-based method used at the moment. Hence, investigating the block Davidson method could be a central activity in future research.

Chapter 2

WIEN 97

The *linearized augmented plane wave* (LAPW) method is one of the most accurate methods for the computation of the electronic structure of solids based on *density functional theory* (Singh [38]). A full potential LAPW code for crystalline solids has been developed in the last ten years at the Institute for Technical Electrochemistry of the Vienna University of Technology. In 1990 the first copyrighted version was named WIEN (Blaha et al. [13]).

In the last few years, WIEN 93, WIEN 95, and WIEN 97, significantly improved and updated UNIX versions of the original WIEN code, were developed. The current version, WIEN 97, is the most sophisticated one, especially in terms of speed, versatility, user-friendliness and new features (for example, to compute optical properties, to optimize geometrical properties, and to obtain higher accuracy using spin-orbit coupling).

WIEN 97 is written in Fortran 77 and requires a UNIX operating system since several programs are linked together via C-shell scripts.

Hardware requirements change from case to case. Small cases with 10 atoms per unit cell can be run on a Pentium PC with 64 MB main memory under Linux, but generally a powerful workstation is recommended with 256 MB (or more) main memory and 1–4 GB of disk space.

The first sections of this chapter give a short overview over density functional theory which is the mathematical basis of WIEN 97.

The rest of this chapter describes how the program `lapw1`, which performs eigenproblem computations, is embedded in the WIEN 97 program package, and how its floating-point performance can be improved.

2.1 Density Functional Theory

The theorem upon which *density functional theory* and *local density approximation* are based is that of Hohenberg and Kohn[1] [29]. It states that the total energy E of a non-spin-polarized system of interacting electrons in an external potential (in the following, the Coulomb potential due to the nuclei in a solid) is given exactly as a functional of the ground state electronic density ρ, i. e.,

$$E = E[\rho]. \tag{2.1}$$

Hohenberg and Kohn showed that the true ground state density is the density that minimizes $E[\rho]$, and that the other ground state properties (for example, optical properties or X-ray spectra) are also functionals of the ground state density. The extension to spin-polarized systems is straightforward; E and the other ground state properties become functionals of both the up and down spin densities, i. e.,

$$E = E[\rho_\uparrow, \rho_\downarrow]. \tag{2.2}$$

Unfortunately, the Hohenberg-Kohn theorem provides no guidance as to the form of $E[\rho]$, and therefore the utility of density functional theory depends on the discovery of sufficiently accurate approximations. In order to do this, the unknown functional $E[\rho]$ is rewritten as the Hartree total energy $T_s[\rho]+E_{ei}[\rho]+E_{ii}[\rho]+E_H[\rho]$ plus another, but presumably smaller, unknown functional, called the exchange correlation (xc) functional $E_{xc}[\rho]$:

$$E[\rho] = T_s[\rho] + E_{ei}[\rho] + E_{ii}[\rho] + E_H[\rho] + E_{xc}[\rho]. \tag{2.3}$$

T_s denotes the single particle kinetic energy, E_{ei} is the Coulomb interaction energy between the electrons and the nuclei, E_{ii} arises from the interaction of the nuclei with each other, and E_H is the Hartree component of the electron-electron energy, i. e.,

$$E_H[\rho] = \frac{e^2}{2} \int d^3r d^3r' \frac{\rho(r)\rho(r')}{|r - r'|}. \tag{2.4}$$

In the local density approximation, $E_{xc}[\rho]$ is written as

$$E_{xc}[\rho] = \int d^3r \rho(r) \epsilon_{xc}(\rho(r)) \tag{2.5}$$

[1] Walter Kohn received the Nobel price for chemistry 1998 for his work on density functional theory.

where $\epsilon_{xc}(\rho)$ is approximated by a local function of the density, usually a function which reproduces the known energy of the uniform electron gas.

Kohn and Sham [31] wrote the electron density $\rho(r)$ as a sum of single particle densities, and determined the ground state energy $E[\rho]$ and density ρ. Thus, instead of having to solve a many-body Schrödinger equation, using density functional theory, only the solution of a series of single particle equations (satisfying a self-consistency condition) is required. In particular, Kohn and Sham showed that the correct density $\rho(r)$ is given by the self-consistent solution of a set of single particle Schrödinger equations, known as the *Kohn-Sham equations*, with a density dependent potential,

$$H\phi_i(r) = (T + V_{ei}(r) + V_H(r) + V_{xc}(r))\phi_i(r) = \epsilon_i\phi_i(r), \tag{2.6}$$

with the density ρ given by a sum over the occupied orbitals (Fermi sum), i. e.,

$$\rho(r) = \sum_{occ} \langle \phi_i(r), \phi_i(r) \rangle. \tag{2.7}$$

$\langle \cdot, \cdot \rangle$ denotes an appropriate scalar product. Here the highest occupied orbital is determined by the electron count, the ϕ_i represent single particle orbitals, ϵ_i are the corresponding eigenvalues, T is the kinetic energy operator, V_{ei} is the Coulomb potential due to the nuclei, V_H is the Hartree potential and V_{xc} is the exchange correlation potential. Both V_H and V_{xc} depend on ρ, i. e.,

$$V_H(r) = e^2 \int d^3r' \frac{\rho(r')}{|r - r'|} \tag{2.8}$$

and

$$V_{xc}(r) = \frac{\partial E_{xc}[\rho]}{\partial \rho(r)}. \tag{2.9}$$

In this framework, the self-consistent solution of the Kohn-Sham equations, (2.6) and (2.7), can be obtained. That is, a density ρ must be found which yields an effective potential $E[\rho]$ that when inserted (using Equations (2.8) and (2.9)) into the Schrödinger-like Equations (2.6) yields orbitals that reproduce the potential.

2.2 Solution of the Single Particle Kohn-Sham Equations

Density functional theory based electronic structure methods are classified according to the representations used for the density ρ, for the potential $E[\rho]$ and, most importantly, for the the Kohn-Sham orbitals ϕ_i. Particular representations are chosen to minimize the computational costs and human (e. g., programming) effort, while maintaining sufficient accuracy. These competing, material and application dependent goals have led to the development and use of a wide variety of techniques, one of them being the LAPW method.

One way is to represent the orbitals ϕ_i in terms of basis functions. It is possible to avoid the explicit use of a basis in constructing the Kohn-Sham orbitals, for example, by numerically solving the differential equation (2.6) on grids. However, nearly all approaches that have been proposed for solids, including the LAPW method, do rely on basis set expansions for the Kohn-Sham orbitals ϕ_i, because well chosen basis sets turn out to be very efficient. In this report the discussion is confined to methods which use an infinite expansion. There is a heuristic method for determining where to truncate the infinite expansion, i. e., for determining the number n_b of basis functions $\Phi_\alpha(r)$. Thus the Kohn-Sham orbitals ϕ_i are represented as

$$\phi_i(r) = \sum_{\alpha \leq n_b} c_{i\alpha} \Phi_\alpha(r) \ , \tag{2.10}$$

where the $c_{i\alpha}$ are expansion coefficients. Given a certain basis, these coefficients are the only variables in the problem (note, according to (2.7) the density ρ depends only on the Kohn-Sham orbitals ϕ_i). Using the theorem of Hohenberg and Kohn, a self-consistent solution to the Kohn-Sham Equation (2.6) can be found by determining the coefficients $c_{i\alpha}$ in Equation (2.10) that minimize the total energy $E[\rho]$.

The energy functional $E[\rho]$ can be rewritten using the single particle eigenvalues ϵ_i to eliminate the unknown functional $T_s[\rho]$ in Equation (2.3):

$$E[\rho] = E_{ii}[\rho] + \sum_{occ} \epsilon_i + E_{xc}[\rho] - \int d^3r \rho(r)(V_{xc}(r) + \frac{1}{2} V_H(r)) \ , \tag{2.11}$$

where the sum is over the occupied orbitals, and ρ, V_H, and V_{xc} are given by Equations (2.7), (2.8), and (2.9), respectively.

The optimization of the $c_{i\alpha}$ and the determination of the self-consistent charge density ρ can be done separately. In this scheme it is necessary to repeatedly determine the $c_{i\alpha}$ that solve the single particle equations (2.11) for a fixed charge density. This may be done using standard techniques from numerical linear algebra. Specifically, given the basis functions $\Phi_\alpha(r)$, the Hamiltonian and the overlap matrices, H and S, are constructed (see Section 2.2.1) and the eigenvalue problem

$$(H - \epsilon_i S) c_i = 0 \ , \tag{2.12}$$

is solved. The rank of the square matrices H and S is equal to the number of basis functions, n_b, and the c_i are eigenvectors containing the n_b coefficients $c_{i\alpha}$.

If the occupied Kohn-Sham orbitals can be expressed as linear combinations of the chosen basis functions, then optimizing the $c_{i\alpha}$ will yield the exact self-consistent solution. In practice, it is impossible to find such a set of basis functions. The quality of the basis chosen determines the accuracy of the solution from this method. Each approximate solution is optimal in the sense that it gives the lowest possible total energy $E[\rho]$ for a given basis. The usefulness of a basis set can, therefore, be measured by the extent to which the total energy evaluated using the orbitals ϕ_i of Equation (2.10) differs from the true Kohn-Sham energy $E[\rho]$. Using larger sets of basis functions can improve the accuracy of the solution, at the cost of higher computational effort.

2.2.1 Synthesis of the Hamiltonian and Overlap Matrices

The matrix elements, $S_{GG'}$ and $H_{GG'}$ (overlap matrix and Hamiltonian, respectively) are given as

$$S_{GG'} = \langle \Phi_G, \Phi_{G'} \rangle \tag{2.13}$$

and

$$H_{GG'} \; = \; \langle \Phi_G(r), E\Phi_{G'}(r) \rangle \; = \; \langle E\Phi_G(r), \Phi_{G'}(r) \rangle. \tag{2.14}$$

These matrices may be decomposed into interstitial and sphere components (see Section 2.3).

The LAPW basis functions allow to exploit inversion symmetry, if present, by choosing the origin of the unit cell to coincide with an inversion center. In general, H and S are complex Hermitian matrices. If inversion symmetry is present, H and S become real symmetric matrices, making their construction and diagonalization easier than for the complex Hermitian case.

2.2.2 Self-Consistency in Density Functional Calculations

As mentioned in Section 2.1, the Hohenberg-Kohn theorem shows that the total energy $E[\rho]$ has a minimum in terms of ρ, and this is the key to its usefulness. The true ground state density ρ is that density which minimizes the energy $E[\rho]$. This is the basis of the standard self-consistency cycle. Given a charge density ρ_{in}, Equation (2.12) is diagonalized, ensuring that the orbitals ϕ_i are orthonormal and that no orbitals are missed. This eliminates almost all local extrema of $E[\rho]$. An output charge density ρ_{out} is constructed from the eigenvectors c_i using Equations (2.7) and (2.10), and then mixed with the input charge density ρ_{in}

to yield a refined input charge density ρ_{in} for the next iteration. The simplest mixing scheme is straight mixing

$$\rho_{in}^{i+1} = (1-\alpha)\rho_{in}^{i} + \alpha\rho_{out}^{i} , \tag{2.15}$$

where the superscript refers to the iteration number and α is the mixing parameter. For sufficiently small α the iterations converge. However, the region of convergence can be small, and becomes smaller rapidly as the size of the unit cell (see Section 2.3) increases, particularly for metallic systems. As a result, considerable effort has been devoted to devising more sophisticated mixing procedures, using information from previous iterations to accelerate the convergence.

2.3 The Full Potential LAPW method

Like most "energy-band methods", the LAPW method is a procedure for solving the Kohn-Sham equations for the ground state density ρ, total energy $E[\rho]$, and (Kohn-Sham) eigenvalues ϵ_i (energy bands; see Equation (2.6)) of a many-electron system by introducing a finite basis set which is especially adapted to the problem.

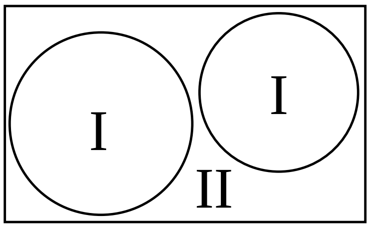


Figure 2.1: Partitioning of the unit cell into atomic spheres (I) and an interstitial region (II).

In the case of a crystal, a unit cell is defined (see Fig. 2.1). The adaptation of the basis set is achieved by dividing the unit cell into (I) non-overlapping atomic spheres (centered at the atomic sites) and (II) an interstitial region. In the two types of regions different basis sets $\{\Phi_\alpha\}$ are used.

(I) Inside an atomic sphere t of radius R_t, a linear combination of radial functions u_l and $\dot{u}_l$ times spherical harmonics $Y_{lm}(r)$ is used (the index t is omitted), i. e.,

$$\Phi_\alpha = \sum_{lm} [A_{lm}^{\alpha} u_l(r, E_l) + B_{lm}^{\alpha} \dot{u}_l(r, E_l)] Y_{lm}(\hat{r}) , \tag{2.16}$$

where $u_l(r, E_l)$ is the (at the origin) regular solution of the radial Schrödinger equation for energy E_l (chosen usually at the center of the corresponding band with its main contribution due to quantum number l) and the spherical part of the potential inside sphere t; $\dot{u}_l(r, E_l)$ is the energy derivative of u_l taken at the same energy E_l. A linear combination of these two functions constitutes the linearization of the radial functions; the coefficients A^{α}_{lm} and B^{α}_{lm} are functions of α determined by requiring that this basis function matches (in value and slope) the corresponding basis function of the interstitial region; u_l and $\dot{u}_l$ are obtained by numerical integration of the radial Schrödinger equation on a radial mesh inside the sphere.

(II) In the interstitial region plane waves,

$$\Phi_\alpha = \frac{1}{\sqrt{\omega}} e^{i\alpha r} \ , \tag{2.17}$$

are used as basis functions.

2.4 Programs of WIEN 97

A test case, rutile (titanium dioxide TiO_2, a common mineral), provided in the WIEN 97 package, was run on a workstation with an Alpha 21164A processor. The execution times of the programs of WIEN 97 are presented in Fig. 2.2, and Tables 2.1 and 2.2.

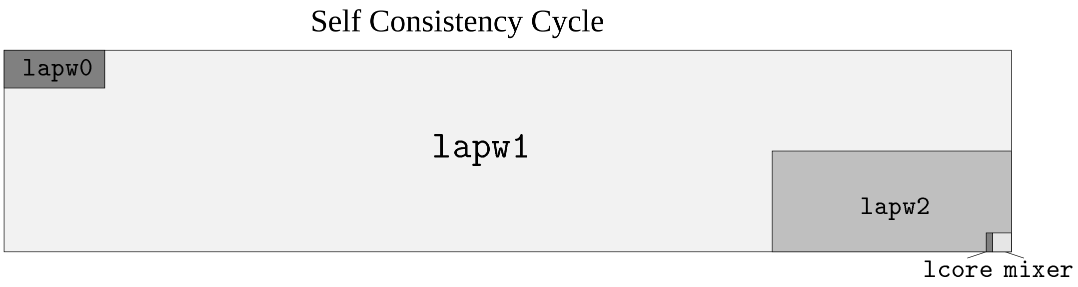


Figure 2.2: Execution time partition for the WIEN 97 modules of the self-consistency cycle (material: rutile, TiO_2). The area of each box is proportional to the execution time of the respective routine.

`lapw1` is one of the programs of the WIEN 97 package used in a self-consistency cycle (see Fig. 2.3). This cycle is repeated until convergence criteria are met. The cycle involves the modules `lapw0` (generates potential $E[\rho]$ from density ρ), `lapw1` (calculates valence bands, i. e., eigenvalues ϵ_i and eigenvectors c_i), `lapw2`

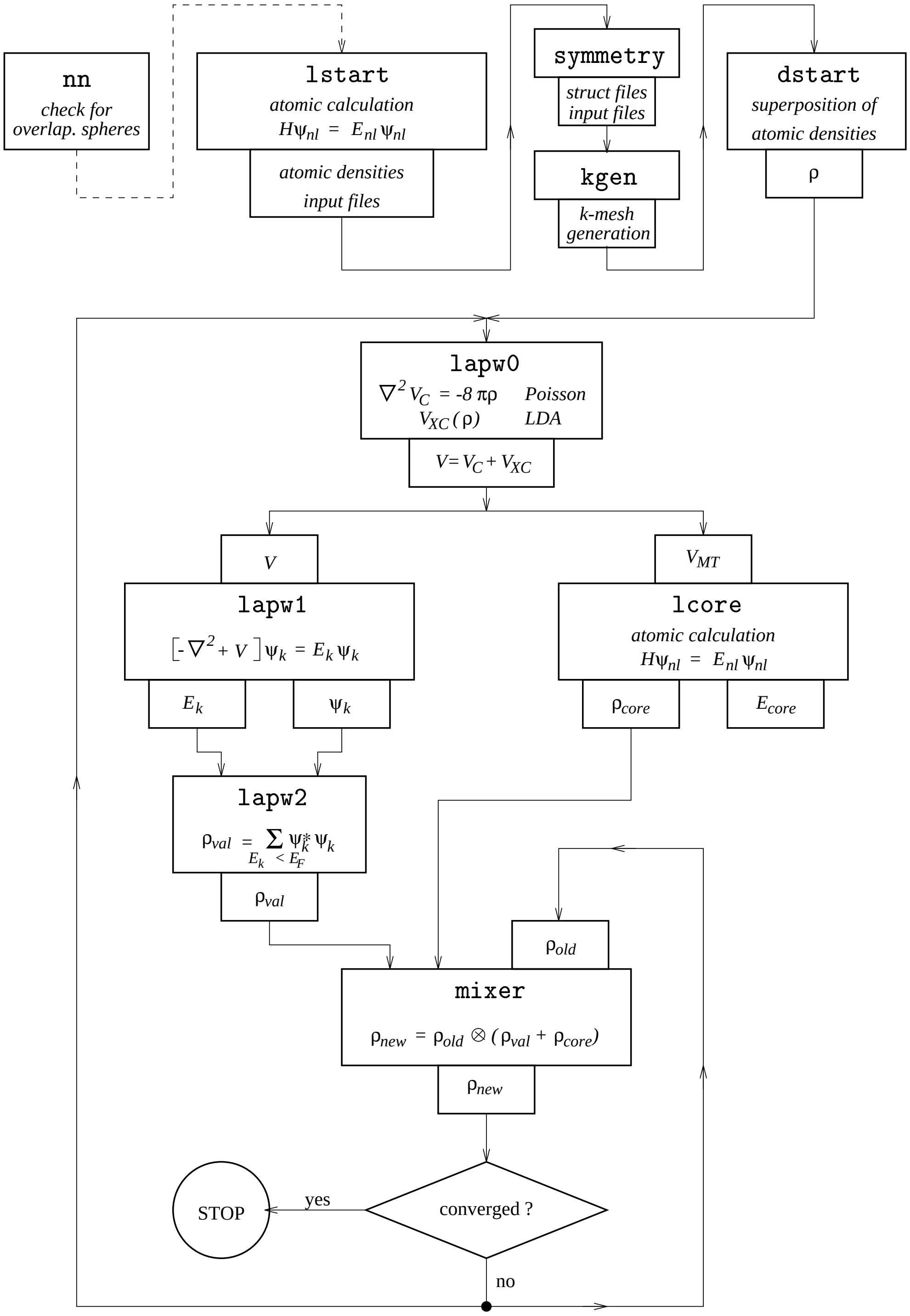


Figure 2.3: Program flow in WIEN 97.

lapw0	lapw1	lapw2	lcore	mixer
3.8 s	197.4 s	13.2 s	0.1 s	0.4 s
1.8 %	92 %	6 %	0.05 %	0.2 %

Table 2.1: Execution times for the programs of the self-consistency cycle (material: rutile, TiO_2).

nn	lstart	symmetry	kgen	dstart
0.02 s	0.6 s	0.03 s	0.02 s	3.2 s

Table 2.2: Execution times (in seconds) for the programs in the initialization phase (material: rutile (TiO_2)). Compared with the self consistency cycle these run times are insignificant.

(computes valence densities ρ from eigenvectors c_i), `lcore` (computes core states and densities), and `mixer` (mixes input ρ_{in} and output densities ρ_{out} to improve the convergence behavior).

An interactive initialization phase is needed before the self-consistency cycle is started. It consists of the routines `nn` (lists the nearest neighbors), `lstart` (generates free atomic densities and determines how the different orbitals are treated in the band structure calculations), `symmetry` (generates group symmetry operations from an input file), `kgen` (generates a k-mesh), and `dstart` (generates a starting density for the self-consistency cycle).

When the self-consistency cycle is finished other programs are needed, e. g., for generating graphical output. `tetra` calculates total and partial density of states. `spaghetti` generates an energy band structure plot (as a PostScript file). `lapw3` calculates X-ray structure factors from the charge density by Fourier transformation. `lapw5` generates the charge density (or the potential) in a specified plane of the crystal. `xspec` calculates near edge structure of X-ray absorption or emission spectra. `mini` performs movements of the atomic positions according to calculated forces. Using this program geometry minimizations can be performed in a big loop that contains the self-consistency cycle. `averx` calculates the average value of the operator X, which might be the number of electrons in the atomic sphere, or an orbital moment, or an orbital moment operator, or a hyperfine coupling operator. `optic` calculates optical properties. `joint` computes the inter-band or the intra-band contributions to the imaginary part of the dielectric tensor, alternatively, the density of states or the joint density of states can be derived. The output of `optic` is used by this program. `kram` carries out a Kramers-Kronig transformation using the imaginary part of the dielectric tensor derived in `joint`.

2.4.1 LAPW1

`lapw1` is the most time consuming module of WIEN 97 (see Table 2.1). In some exceptional cases `lapw2` may run nearly as long as `lapw1`, but all the other programs' run times remain insignificant. This is the reason why this report concentrates on `lapw1`.

The tasks performed by `lapw1` are (i) to set up the matrices H and S, and (ii) to solve the generalized symmetric eigenvalue problem. The first step requires $O(n^2)$ floating-point operations, whereas the second step requires $O(n^3)$ floating-point operations. Therefore, for larger problems the second step is the most time consuming part of `lapw1` and of the whole self-consistency cycle.

2.4.2 Eigenproblem Computation in WIEN 97

The main objective of our work was to improve performance (and efficiency) of the WIEN 97 package and at the same time to retain its accuracy. Our current work focuses on improving the efficiency of numerically solving Equation (2.12).

The equation

$$HC = SCE \tag{2.18}$$

has to be solved with a large (for example, 4000×4000), dense, symmetric/Hermitian matrix H, a symmetric/Hermitian positive definite matrix S, the diagonal matrix of (unknown) eigenvalues E, and the matrix of (unknown) eigenvectors C. Only the *lower* part (about 10 %) of the eigenpairs (eigenvalues together with the corresponding eigenvectors) is required.

The standard serial algorithms for computing the eigendecomposition of a symmetric matrix have the same two-phase structure: Apply orthogonal transformations to reduce the matrix to a *condensed* form and then apply an iterative algorithm to the condensed matrix to compute its eigendecomposition. Other approaches to the symmetric eigenproblem apply to *dense* matrices instead of *condensed* matrices. The best known is Jacobi's method .

For solving the Schrödinger equation, WIEN 97 switches between a condensed and a dense matrix eigensolver. Compared to the dense matrix eigensolver, the condensed matrix eigensolver involves much more floating-point operations. The condensed matrix eigensolver proceeds as follows:

1. **Reduction** of the generalized eigenvalue problem (2.18) **to a standard eigenvalue problem**

$$H'C' = C'E. \tag{2.19}$$

This involves a Cholesky factorization $S = U^\top U$ (requiring roughly $n^3/3$ operations; see Golub and Van Loan [26]), and a two-sided matrix update $H' = (U^{-1})^\top H U^{-1}$ (requiring roughly n^3 operations; see Golub and Van Loan [26]).

We have substantially improved this step. In contrast to the previous version of the code, WIEN 95, it is fully blocked, i. e., a Level 3 algorithm, now. Speedups up to 500 % have been achieved.

2. **Tridiagonalization** of H' (requiring roughly $4n^3/3$ operations; Golub and Van Loan [26]).

   This part also has been improved compared to the previous version of the code, WIEN 95. Speedups of up to 100 % have been achieved.

3. **Calculation of the k required eigenvalues** in the tridiagonal form (using iterative methods usually requiring $O(nk)$ operations).

4. **Calculation of the corresponding eigenvectors** in the tridiagonal form (using iterative methods requiring $O(nk^2)$ operations if reorthogonalization is required).

5. Performing all necessary **back-transformations** (requiring roughly $5n^2k/2$ operations). The eigenvalues remain unchanged in the previous steps, but the eigenvectors of the tridiagonal matrix have to be transformed back to the eigenvectors of the original matrix H.

The main design goal was to improve the spatial locality of memory references in the condensed matrix eigensolver. On modern computer architectures improved data locality results in an efficiency enhancement, i. e., in reduced execution times.

Several modifications to the original algorithms have been applied. EISPACK routines have been replaced by LAPACK routines. Then the LAPACK routines were modified in a way that combines the memory efficiency of the packed storage scheme and the runtime efficiency of Level 3 BLAS routines. A new storage scheme—block packed storage—has been introduced to guarantee memory efficiency. An additional level of blocking—hierarchical blocking—was introduced to allow use of Level 3 BLAS.

2.4.3 Further Improvement

Tridiagonalization

The tridiagonalization process in Step 2 does not utilize Level 3 BLAS to a satisfactory degree. Thus, attempts have been made to introduce more Level 3 operations (see Gansterer et al. [23, 24].

Improved Eigensolvers

In the future, the whole problem will be considered at a higher level: *different* methods for solving the linear eigenproblem will be investigated. Finally—as a long-term plan—other high-level modifications (e. g., for convergence acceleration) could be made. These issues, however, are *not* part of the current report.

In the current stage it was the goal to achieve performance improvements based on *existing* software (e. g., using LAPACK routines with only minor code modifications wherever possible), and not to do a lot of coding from scratch. Future work will necessarily attack the problem at a higher level and will have to look into substantially different algorithmic approaches. This will most likely open up a variety of opportunities for further improvement. But at the same time it will require significantly higher coding and implementation effort.

As already mentioned, there is a second, dense matrix eigensolver in WIEN 97. At the moment, it is approximately five times faster than the condensed matrix eigensolver, which is based on tridiagonalization. Trading convergence for performance, WIEN 97 switches back and forth between the two methods: Whenever the convergence of the dense matrix eigensolver slows down or the iterates proceed in a wrong direction, WIEN 97 switches back to the slower, but much more accurate condensed matrix eigensolver.

The basic principle of the dense matrix eigensolver, which is implemented in WIEN 97, is described in Singh [37]. The method is based on the observation that the exact eigenvectors in the initial iterations (far from self-consistency) are of little interest and approximate eigenvectors would serve as well, while in the final iterations the eigenvectors from the previous iteration are a very close approximation to the true eigenvectors. Thus the eigenvectors can be refined as self-consistency is approached, yielding a substantial reduction in computing time. The advantage of this method over the conventional approach lies in the fact that it solves the diagonalization problem simultaneously with the self-consistency problem.

Provided with accurate starting values computed by the condensed matrix eigensolver, several steps of the dense matrix eigensolver can be performed. The accuracy of the dense matrix eigensolver could be increased by computing a larger subspace, i. e., a larger number of eigenpairs. This, however, would require more time for the whole self-consistency cycle than switching between the two eigensolvers.

Parallel Code Version

Hierarchical blocking has been used to optimize the algorithms for modern cache based sequential computer architecture. A parallel code version, which is under development, will substantially benefit from this blocking, as the ratio of computation to communication can be held fairly high.

Chapter 3

Mathematical Background

This chapter briefly summarizes some of the mathematical background relevant to the solution of symmetric eigenvalue problems. A good coverage of the relevant background material can be found in the standard reference book by Golub and Van Loan [26], and many practical issues related to numerical software for eigenvalue problems are discussed in Demmel [17]. Very comprehensive treatment of methods for solving eigenproblems and for computing eigenvectors is also given in the more specialized books Wilkinson [42] and Parlett [34].

3.1 Generalized Symmetric Eigenproblems

Definition 3.1.1 (Generalized Symmetric Definite Eigenproblem) *Given a symmetric matrix A and a symmetric positive definite matrix B, one version*[1] *of the* generalized symmetric definite eigenvalue problem (GSEP) *is to solve*

$$Ax = \lambda Bx \tag{3.1}$$

for eigenvalues λ and corresponding eigenvectors $x \neq 0$.

Generalized *Hermitian* definite eigenproblems are defined analogously.

3.1.1 Reduction of a GSEP to a Standard Eigenproblem

Setting $B = I$ in (3.1) yields a *standard symmetric/Hermitian eigenproblem* (SEP). In this case the eigenvalues are known to be real, and there exists an

[1] Problems of the form $ABx = \lambda x$ or $BAx = \lambda x$ are different versions of the generalized eigenproblem.

orthonormal/unitary basis of eigenvectors (the geometric multiplicity of each eigenvalue equals its algebraic multiplicity).

A GSEP can be reduced to a SEP: Given the Cholesky factorization $B = LL^\top$ (L being a lower triangular matrix) the GSEP (3.1) can be rewritten as

$$(L^{-1}AL^{-\top})(L^\top x) = \lambda(L^\top x). \tag{3.2}$$

The eigenvalues of $Ax = \lambda Bx$ are identical to those of $Cy = \lambda y$, where C is the symmetric matrix $C := L^{-1}AL^{-\top}$ and the eigenvectors are transformed according to $y := L^\top x$. Analogously, the factorization $B = U^\top U$ (U being an upper triangular matrix) can be used to rewrite (3.1) as

$$(U^{-\top}AU^{-1})(Ux) = \lambda(Ux). \tag{3.3}$$

Summarizing, the standard way for solving a GSEP proceeds in the following steps.

1. Perform a Cholesky factorization of B,
2. Reduce the GSEP to the corresponding SEP $Cy = \lambda y$ by applying one of the similarity transformations (3.2), (3.3) to A,
3. Tridiagonalize C,
4. Compute eigenvalues and eigenvectors of the standard SEP,
5. Transform the computed eigenvectors.

In the WIEN 97 package, Steps 1 and 2 have been implemented quite efficiently by using the concept of hierarchical blocking (Ueberhuber [41]) in connection with a new storage format for symmetric matrices. This work is described in detail in Haunschmid and Kvasnicka [27].

Efficient implementations of Steps 3, 4, and 5 require some algorithmic modifications of standard techniques which are described in this report.

3.2 The Tridiagonalization Process

An important preprocessing step after the reduction of a GSEP to a SEP is the *tridiagonalization* of the symmetric/Hermitian matrix A.[2] This is usually done by a (finite) series of *similarity transformations* which introduce zeroes below the first subdiagonal.

[2] In the non-symmetric case only a transformation to *Hessenberg* form can be achieved.

Definition 3.2.1 (Similarity) *The matrices A and B are said to be* similar*, if $B = X^{-1}AX$ with a non-singular matrix X.*

Similar matrices have the same spectrum, and closely related eigenvectors: multiplying the eigenvectors of B by X from the left gives the eigenvectors of A.

3.2.1 Unitary/Orthogonal Similarity

For the numerically stable solution of eigenproblems, similarity transformations

$$B = Q^{-1}AQ$$

are performed using *unitary* or *orthogonal* matrices Q. These are matrices, for which

$$Q^{-1} = Q^H \qquad \text{or} \qquad Q^{-1} = Q^\top$$

holds, i. e., their inverse can be obtained without further calculation.

Moreover, a unitary (orthogonal) matrix Q has unit Euclidean norm ($\|Q\|_2 = 1$). Consequently, multiplying a vector x by a unitary (orthogonal) matrix ($y = Qx$) does not change its Euclidean norm:[3]

$$\|y\|_2 = \sqrt{\langle y, y\rangle} = \sqrt{y^H y} = \sqrt{x^H Q^H Q x} = \sqrt{x^H I x} = \sqrt{x^H x} = \sqrt{\langle x, x\rangle} = \|x\|_2.$$

A unitary similarity transformation represents (like any other similarity transformation) a change from one basis used for a matrix representation of the underlying linear operator to another basis. In the case of unitary transformations, however, *orthonormal* bases are exchanged.

Two well known unitary (orthogonal) transformations are *plane rotations* (also known as *Givens* or *Jacobi rotations*) and *(Householder) reflections*. Many numerically stable algorithms for various kinds of elimination processes (one example being the tridiagonalization process for solving eigenproblems) are based on the successive application of such transformations.

Plane Rotations (Givens Rotations)

It is a result from analytic geometry in $\mathbb{R}^2$ that under a rotation of the coordinate axes through an angle ϕ (see Fig. 3.1) the coordinates of a point are transformed as follows.

[3] $\langle \cdot\,, \cdot \rangle$ denotes the scalar product.

$$\begin{pmatrix} x_i \\ x_j \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \hat{x}_i \\ \hat{x}_j \end{pmatrix}, \qquad \begin{pmatrix} \hat{x}_i \\ \hat{x}_j \end{pmatrix} = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} x_i \\ x_j \end{pmatrix}.$$

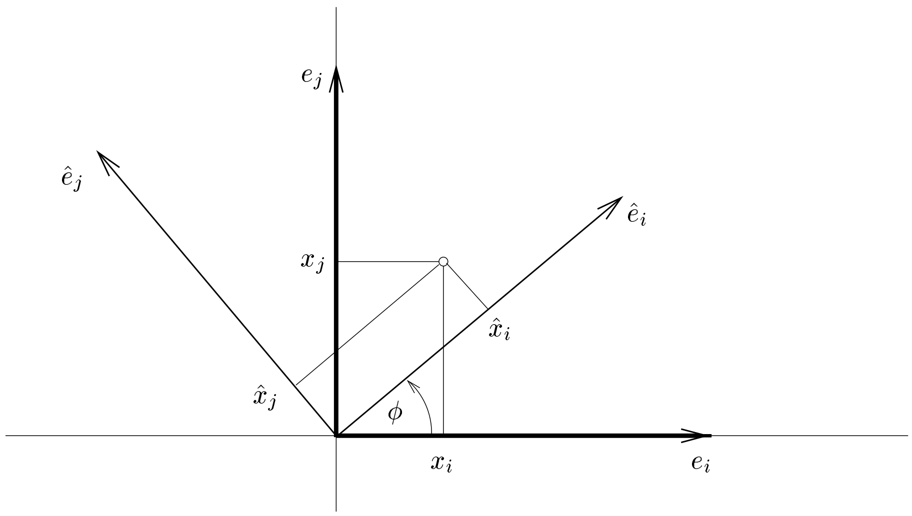


Figure 3.1: Plane rotation (in the i-j plane) through an angle ϕ.

In $\mathbb{R}^n$ a rotation in the i-j plane, spanned by the unit vectors $\hat{e}_i$ and $\hat{e}_j$ is obtained by a premultiplication with the transposed *rotation matrix* (*Givens matrix*, *Jacobi matrix*) $G(\phi; i, j)$. This matrix is defined as the identity matrix subject to the following alterations:

$$g_{ii} = g_{jj} = \cos\phi, \quad -g_{ij} = g_{ji} = \sin\phi \qquad \text{where} \quad |\phi| \leq \pi$$

$$G(\phi; i, j) \; := \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & \mathbf{0} & \\ & & 1 & & & & & & & & \\ & & & \cos\phi & & & & -\sin\phi & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & \sin\phi & & & & \cos\phi & & & \\ & & & & & & & & 1 & & \\ & \mathbf{0} & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix}.$$

The premultiplication $A := G^\top(\phi; i, j)A$ affects only the ith and jth *row* of the matrix A. On the other hand, the postmultiplication $A := A\, G(\phi; i, j)$ affects only the ith and jth *column* of A. Accordingly, the similar matrices A and

$$B := G^{\top}(\phi; i, j)\, A\, G(\phi; i, j)$$

differ only in their ith and jth rows and columns.

Householder Reflections

Given the *Householder vector* $u \in \mathbb{C}^n$, a *Householder reflector*

$$H_n(u) := I_n - \tau u u^H$$

can be defined. By construction, $H_n(u)$ is a Hermitian matrix. Additionally requiring $H_n(u)$ to be a unitary matrix yields the following condition:

$$2\mathrm{Re}(\tau) = |\tau|^2 \|u\|_2^2.$$

Householder reflections can be represented in several ways (Lehoucq [33]). In this report it will be required that $\tau \in \mathbb{R}$ and that u is normalized ($\|u\|_2 = 1$), which implies $\tau = 2$.

Application of $H_n(u)$ to a vector $x \in \mathbb{C}^n$ has the following effect:

$$y := H_n(u)x = (I_n - \frac{2}{u^H u} u u^H)x = x - \frac{2}{u^H u}\langle u, x\rangle u = x - \left(\frac{2}{u^H u}\|x\|_2 \cos\langle u, x\rangle\right) u.$$

Interpreted geometrically this means that the vector y is obtained by a reflection of x on the hyperplane whose normal vector is u (see Fig. 3.2).

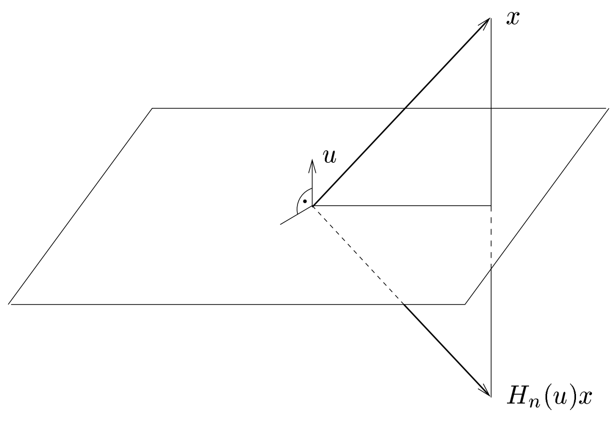


Figure 3.2: Reflection of a vector x on the plane with the normal vector u.

In particular, u can be chosen such that y is a scalar multiple of the unit vector e_1:

$$u = x \pm \|x\|_2 e_1 \quad \Rightarrow \quad H_n(u)x = \mp \|x\|_2 e_1.$$

Comparison

Generating a single Givens rotation which eliminates one element in a matrix requires five floating-point operations (*flops*) and one additional square root (Golub and Van Loan [26]). Tridiagonalization requires

$$\frac{5}{2}n^2 + O(n) \quad \text{flops} \quad \text{and} \quad \frac{n^2}{2} + O(n) \quad \text{square roots}$$

for constructing the $(n-2)+(n-3)+\ldots+1 = (n-2)(n-1)/2$ Givens rotations needed to transform a symmetric $n \times n$ matrix A into tridiagonal form in one sweep.

The construction of Householder transformations requires *fewer* operations: generating a single Householder transformation which transforms a vector $x \in \mathbb{R}^k$ into a multiple of e_1 involves about $3k$ flops and one square root (Golub and Van Loan [26]). This leads to a total amount of

$$\sum_{k=2}^{n} 3k = \frac{3}{2}n^2 + O(n) \quad \text{flops} \quad \text{and} \quad n-1 \quad \text{square roots}$$

in the standard tridiagonalization process of a symmetric matrix in one sweep. The increased number of square roots for Givens rotations ($O(n^2)$ instead of $O(n)$) is one of the reasons why Householder transformations are often preferred to Givens rotations.

Most of the work in the tridiagonalization process, however, is not dedicated to generating the transformations but to *applying* them to the unreduced part of the matrix. In summary, Householder tridiagonalization requires $4n^3/3 + O(n^2)$ flops. If the transformation matrix Q is explicitly required then it can be formed with an additional $4n^3/3 + O(n^2)$ flops (Golub and Van Loan [26]).

Usually Householder reflectors are usually used in elimination processes. Only in certain cases, for example, when eliminating single elements in a matrix, Givens rotations are more suitable.

3.3 Methods for Computing Eigenvectors

A more detailed discussion of the methods described in this section can be found in Wilkinson [42], Parlett [34], and Golub and Van Loan [26].

3.3.1 The Power Method

Algorithm 3.3.1 shows the structure of the *power method*, an iterative process for computing eigenvectors (and eigenvalues) of a matrix A. Under suitable conditions $\lambda^{(k)}$ converges to the *dominant*[4] eigenvalue and $x^{(k)}$ converges to the corresponding eigenvector.

Algorithm 3.3.1 Power Method

do $k = 0, 1, 2, \ldots$
$y^{(k+1)} := Ax^{(k)}$
$x^{(k+1)} := y^{(k+1)}/\|y^{(k+1)}\|_\infty$
$\lambda^{(k+1)} := (x^{(k+1)})^H Ax^{(k+1)}$
end do

3.3.2 The Inverse Power Method

If all eigenvalues $\lambda(A)$ of the matrix $A \in \mathbb{C}^{n\times n}$ are non-zero, then there exists an inverse matrix A^{-1} whose spectrum is given by

$$\lambda(A^{-1}) = \{1/\lambda_1, 1/\lambda_2, \ldots, 1/\lambda_n\}.$$

The eigenvectors of A and A^{-1} are identical. If $1/\lambda_n$ is the dominant eigenvalue of A^{-1}, this means

$$1/|\lambda_n| > 1/|\lambda_{n-1}| \geq \cdots \geq 1/|\lambda_1| > 0,$$

then application of the power method to A^{-1} yields the reciprocal of the eigenvalue of A with the *smallest* magnitude.

It is intuitively clear that the process shown in Algorithm 3.3.2 is equivalent to Algorithm 3.3.1 with the matrix $\bar{A} = A^{-1}$ and that $\lambda^{(k)}$ will (under suitable conditions) converge to the eigenvalue of A of smallest modulus and $x^{(k)}$ to the corresponding eigenvector. This process is also called *inverse iteration*.

Spectral Shift

The inverse power method converges particularly fast if $|\lambda_n|$ is significantly smaller than all other eigenvalues' absolute values. This fact can be taken advantage of

[4] λ_1 is called the *dominant* eigenvalue of A if $|\lambda_1| > |\lambda_i| \quad \forall\, i = 2, \ldots, n$. In the following, eigenvalues are assumed to be ordered according to $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_{n-1}| \geq |\lambda_n|$.

Algorithm 3.3.2 Inverse Iteration *(basic form)*

do $k = 0, 1, 2, \ldots$
  solve $Ay^{(k+1)} = x^{(k)}$
  $x^{(k+1)} := y^{(k+1)}/\|y^{(k+1)}\|_\infty$
  $\lambda^{(k+1)} := (x^{(k+1)})^H Ax^{(k+1)}$
end do

when an approximation μ for an eigenvalue of A is already known by using this approximation for a *spectral shift*:

$$\lambda(A - \mu I_n) = \{\lambda_1 - \mu, \lambda_2 - \mu, \ldots, \lambda_n - \mu\}.$$

In this case the matrix $A - \mu I_n$ has an eigenvalue which is very close to zero, and the power method applied to $(A - \mu I_n)^{-1}$ (the inverse power method as outlined in Algorithm 3.3.3), converges particularly fast.

Algorithm 3.3.3 Inverse Iteration *(with spectral shift)*

do $k = 0, 1, 2, \ldots$
  solve $(A - \mu I_n)y^{(k+1)} = x^{(k)}$
  $x^{(k+1)} := y^{(k+1)}/\|y^{(k+1)}\|_\infty$
  $\lambda^{(k+1)} := (x^{(k+1)})^H Ax^{(k+1)}$
end do

Eventually, $x^{(k)}$ will be dominated by the eigenvector x_k whose corresponding eigenvalue λ_k is closest to the point μ in the complex plane.

As mentioned before, the matrix $A - \mu I_n$ has an eigenvalue $\lambda_i - \mu$ very close to zero, if μ is a good approximation to an eigenvalue of A—in other words, the iteration matrix is almost singular. Since each iteration requires the solution of a linear system with the matrix $A - \mu I_n$, doubts about the stability of the method might arise. It can be shown, however, that the inverse power method produces good results despite the almost-singularity of $A - \mu I_n$ (Wilkinson [43]).

The practical implementation of the inverse iteration method requires certain modifications to ensure a reasonable level of efficiency. In particular, the system of linear equations needs not be factorized at every step (because of the $O(n^3)$ computational effort). Instead, an LU decomposition outside the k-loop would be executed. The computational effort per iteration step is thus reduced to $O(n^2)$ operations for the back substitution and is therefore of the same order of magnitude as in the power method. The computational effort of the single LU factorization of $A - \mu I_n$ outside the loop has to be added only once.

In the special case of A being a symmetric band matrix with upper and lower bandwidth b, which will be of interest in Chapters 4 and 6, Algorithm 3.3.3 requires about $2nb^2 + 8nb$ flops (assuming that $n \gg b$ and that not more than 2 iterations suffice to reach the desired accuracy, see Ipsen [30]).

Rayleigh Quotient Iteration

The rate of convergence of the inverse power method can be further improved if the translation parameter μ is updated to the *current* eigenvalue approximation $\lambda^{(k+1)}$ after every step. The method obtained this way—also called *Rayleigh quotient iteration*—converges quadratically, whereas the inverse power method converges only linearly. If A is symmetric, Rayleigh quotient iteration almost always converges cubically (Golub and Van Loan [26]).

The main drawback of this approach is an increased computational effort since now an LU decomposition has to be carried out at every iteration step. There are efforts to replace the LU decomposition by an iterative solver in order to reduce the computational effort (Gansterer and Ueberhuber [25]).

3.4 Blocking

The memory system of modern computer systems is organized *hierarchically*. It consists of registers, several cache levels, main memory, and external memory. Lower levels in this hierarchy have larger capacity but longer access times. Therefore it is crucial for the performance of numerical algorithms to maintain a high ratio of floating-point operations to data transfers between memory layers. This can only be achieved when algorithms and codes exhibit good *data locality*.

In particular, it is important to have currently needed data available as close as possible to the processor. With this goal in mind, *blocking* matrix operations was found to be very successful given the memory architectures of most of today's high performance computers (Dongarra et al. [19], Ueberhuber [41]). In addition to algorithmic blocking several other blocking techniques to improve data-locality have been identified (Gansterer et al. [22]).

In the context of matrix-updates, a *Level 3 algorithm with localized data access patterns* usually performs the following basic steps (the order might differ from case to case):

1. Compute a block of the matrix (for example, by performing b single operations for b successive columns of the matrix) *without read or write references to the rest of the matrix*. If the corresponding column block is chosen such that it fits into the cache memory, then no lower levels of the memory hierarchy have to be addressed, and no data transfers are required (all memory accesses are *local*).

2. Accumulate all b single column operations into Level 3 operations and update the rest of the matrix by using Level 3 BLAS routines. There is room for further improvement: Block updates can themselves be blocked (this leads to *hierarchical blocking*, see Ueberhuber [41]), or the update of a part of the matrix which is not accessed immediately can be further delayed.

Obviously this method is only applicable if the dependency structure in the operation being performed allows for such a decoupling of memory accesses. A standard example of an operation for which localized data access patterns can be achieved in a blocked algorithm is matrix-matrix multiplication (Ueberhuber [41]).

As described in Haunschmid and Kvasnicka [27], hierarchical blocking in connection with a new storage format was applied successfully to Cholesky factorization and to the reduction of a generalized to a standard eigenvalue problem in the WIEN 97 package: On average performance improvements by factors of about five could be achieved, on some architectures up to ten.

As will be discussed in Section 4.3, the tridiagonalization process does not allow for a full decoupling of memory references. Consequently, the blocking done in the corresponding LAPACK routines does not yield satisfactory performance. For the same reason it does not parallelize very well, either. Since parallelization is one of the next steps in the development of WIEN 97 (see Chapter 6), potential algorithmic alternatives had to be investigated.

Chapter 4

Blocking the Tridiagonalization Process

The blocking approach for tridiagonalization used by LAPACK will be described in Section 4.2. Its drawbacks will lead to a detailed examination of the dependencies in the tridiagonalization process in Section 4.3. As a result, it will become clear that similarity transformations do not allow for a complete decoupling of memory accesses. Although localized data access patterns as defined in the beginning of Section 3.4 cannot be achieved for one-sweep tridiagonalization, a way to extract the maximum blocking potential will be considered in Sections 4.4 and 4.5.

4.1 Unblocked LAPACK Tridiagonalization

LAPACK provides several routines to reduce a real symmetric matrix $A \in \mathbb{R}^{n \times n}$ to a real symmetric tridiagonal matrix T with an orthogonal similarity transformation $T := Q^\top AQ$.[1]

For illustration purposes *unblocked* tridiagonalization, which is implemented in the LAPACK routine `LAPACK/*sytd2` is described first. It proceeds as shown in Algorithm 4.1.1.

[1] For purpose of simplicity and clarity the *real symmetric* case will be described. Generalizations to the *complex Hermitian* case are mostly straightforward—if special care has to be taken the complex case will be dealt with explicitly.

Algorithm 4.1.1 Unblocked Tridiagonalization

$A^{(0)} := A$

do $i = 1, 2, \ldots, n-2$

1. *Construct the Householder vector* $u_i \in \mathbb{R}^{n-i}$ *such that* $A^{(i-1)}(i+2:n, i)$ *is eliminated.*

2. *Setting* $H := I_{n-i} - 2u_i u_i^\top$ *the similarity transformation*

$$A^{(i)} := \begin{pmatrix} I_i & 0 \\ 0 & H \end{pmatrix} A^{(i-1)} \begin{pmatrix} I_i & 0 \\ 0 & H \end{pmatrix}$$

is performed. Properly partitioning

$$A^{(i-1)} = \begin{pmatrix} A_{11} & A_{21}^\top \\ A_{21} & A_{22} \end{pmatrix},$$

where $A_{11} \in \mathbb{R}^{i \times i}$, $A_{21} \in \mathbb{R}^{(n-i)\times i}$, *and* $A_{22} \in \mathbb{R}^{(n-i)\times(n-i)}$ *yields*

$$\begin{pmatrix} I_i & 0 \\ 0 & H \end{pmatrix} A^{(i-1)} \begin{pmatrix} I_i & 0 \\ 0 & H \end{pmatrix} = \begin{pmatrix} A_{11} & A_{21}^\top H \\ HA_{21} & HA_{22}H \end{pmatrix}.$$

This shows that in addition to the elimination of $A^{(i-1)}(i+2:n, i)$ *(by applying* H *to* A_{21} *from the left) the submatrix* $\bar{A}^{(i-1)} := A^{(i-1)}(i+1:n, i+1:n)$ $(= A_{22})$ *is affected.*

3. *Due to the special structure of* H *the update* $H\bar{A}^{(i-1)}H$ *does not involve* $O(n^3)$ *operations. It can be performed in* $O(n^2)$ *operations in the following way.*

$y_i := 2\bar{A}^{(i-1)}u_i$ *(3a)*

$v_i := y_i - (y_i^\top u_i)u_i$ *(3b)*

$A^{(i)}(i+1:n, i+1:n) := \bar{A}^{(i-1)} - u_i v_i^\top - v_i u_i^\top$ *(3c)*

end do

$T := A^{(n-2)}$

4.2 Blocked LAPACK Tridiagonalization

The LAPACK routine `LAPACK/*sytrd` implements a blocked algorithm which is described in Dongarra et al. [20].

In order to find a *blocked* algorithm, Dongarra et al. [20] proposed the aggregation of a sequence of b Householder transformations, followed by the application of a symmetric *rank*-$2b$ update to the unreduced part of the matrix. This approach is shown in Algorithm 4.2.1.

The strength of this method lies in the potential for using Level 3 BLAS operations in the update of the remaining matrix: A series of b Level 2 BLAS operations is replaced by one Level 3 BLAS operation.

By using the routine `LAPACK/dsytrd`, which is based on this method of blocking the tridiagonalization process in WIEN 97, the performance of the code could be improved compared to the previous version WIEN 95. For example, on an SGI Power Challenge with MIPS R10000 processor, matrix size $n = 2000$, a runtime of 142 seconds compared to 344 seconds was achieved.[2] Nevertheless, a call to the Level 2 matrix-vector routine `BLAS/dsymv` in order to compute y_{b+1} in Step 6 of Algorithm 4.2.1 consumes nearly all the computation time for large matrices in the blocking approach of Dongarra et al. [20], as experiments with the routines `LAPACK/dsytrd` and `LAPACK/dsptrd` and with the block packed version `dshtrd`[3] showed.

This shows that there is still a serious drawback in terms of data-locality in the blocked tridiagonalization process. The application of Householder reflectors to a matrix from the left and from the right involves a matrix-vector product, as shown in Step 3 of Algorithm 4.1.1. This operation remains in the innermost loop of the blocked algorithm, as shown in Step 6 of Algorithm 4.2.1. Because the matrix involved is basically the whole unreduced submatrix this operation repeatedly causes a lot of data transfer, which obviously has a negative effect on performance.

In summary, the memory access patterns of this approach for blocking the tridiagonalization process are not favorable. Algorithm 4.2.1 is blocked, but it does not have localized data access patterns: As discussed in Section 3.4, it would be desirable to update an entire block of A without *any* references to other parts of A. This is not the case here, because it is necessary to reference the whole submatrix to the right of every single column eliminated.[4]

---

[2] Detailed results are given in Haunschmid and Kvasnicka [27].

[3] `dshtrd` is a newly developed routine which is based on a new storage format for symmetric matrices. More details are given in Haunschmid and Kvasnicka [27].

[4] This assumes that the algorithm operates on the lower triangular part of the symmetric

Algorithm 4.2.1 Blocked LAPACK Tridiagonalization

1. *The Householder vector* u_1 *is constructed to eliminate* $A^{(0)}(3:n,1)$.

2. $\bar{A}^{(0)}$ *has to be updated. Instead of performing a full update (see Step 3 of Algorithm 4.1.1) only the first column is updated:*
$$A^{(1)}(2:n,2) := A^{(0)}(2:n,2) - v_1(2)u_1 - u_1(2)v_1.$$

3. *The Householder vector* u_2 *is constructed to eliminate* $A^{(1)}(4:n,2)$.

4. $\bar{A}^{(1)}$ *has to be updated, which would require the calculation of* $y_2 := \bar{A}^{(1)}u_2$ *(see Step 3a of Algorithm 4.1.1). However, at this point* $\bar{A}^{(1)}$ *is not yet fully computed. But by dropping the first components of* u_1 *and* v_1 *(and thus obtaining* $\bar{u}_1$ *and* $\bar{v}_1$*)* y_2 *can be constructed without completing the update of* $\bar{A}^{(1)}$*:*
$$\begin{aligned} y_2 := \bar{A}^{(1)}u_2 &= \left(A^{(0)}(3:n,3:n) - \bar{u}_1\bar{v}_1^\top - \bar{v}_1\bar{u}_1^\top\right)u_2 = \\ &= A^{(0)}(3:n,3:n)u_2 - (\bar{v}_1^\top u_2)\bar{u}_1 - (\bar{u}_1^\top u_2)\bar{v}_1. \end{aligned}$$

5. *Now the first two transformations could be applied as a single symmetric rank-4 update:*
$$\begin{aligned} A^{(2)}(3:n,3:n) &:= \bar{A}^{(1)} - u_2v_2^\top - v_2u_2^\top \\ &= A^{(0)}(3:n,3:n) - u_1v_1^\top - v_1u_1^\top - u_2v_2^\top - v_2u_2^\top. \end{aligned}$$

6. *Continuing this process up to Householder vector* u_b *allows to form* $A^{(b)}$ *from* $A^{(0)}$ *with a symmetric rank-2b update:*
$$A^{(b)}(b+1:n,b+1:n) := A^{(0)}(b+1:n,b+1:n) - UV^\top - VU^\top,$$
where
$$U := (u_1, u_2, \dots, u_b) \in \mathbb{R}^{(n-1)\times b},\ V := (v_1, v_2, \dots, v_b) \in \mathbb{R}^{(n-1)\times b}$$
$$v_b := y_b - \tfrac{1}{2}(y_b^\top u_b)u_b$$
$$y_1 := \bar{A}^{(0)}u_1;\ y_{b+1} := \left(A^{(0)}(b+1:n,b+1:n) - UV^\top - VU^\top\right)u_{b+1}$$
$$\begin{aligned} &A^{(b+1)}(b+1:n,b+1) \\ &\qquad := A^{(b)}(b+1:n,b+1) - \textstyle\sum_{i=1}^{b}\left(v_1(b+1)u_i + u_i(b+1)v_i\right). \end{aligned}$$

Implementation

The blocked Algorithm 4.2.1 is implemented in the routine `LAPACK/*sytrd`. Its structure is shown in Algorithm 4.2.2.

Algorithm 4.2.2 `LAPACK/*sytrd`

do *for all blocks of columns*

1. *call* `LAPACK/*latrd` *to reduce the block to tridiagonal form, and construct matrices* U *and* V *needed to update the unreduced part of the matrix.*

2. *call the Level 3 routine* `BLAS/*syr2k` *to update the unreduced submatrix* S *with a symmetric rank-2b update*

$$S := S - UV^\top - VU^\top.$$

end do

3. *call the unblocked routine* `LAPACK/*sytd2` *to reduce the last (or the only) block.*

The operations with poor data-locality are contained in the routine `LAPACK/*latrd`. For computing the matrix V, this routine performs the matrix-vector product `BLAS/*symv` (see Step 6 in Algorithm 4.2.1) with basically the whole unreduced part of the matrix A inside a loop over the columns of the current block. This Level 2 operation consumes most of the computation time for large matrices: Having to access the entire unreduced part of A for each single column causes very bad memory access patterns, a lot of transfers between different levels of the memory hierarchy, and consequently poor performance (Haunschmid and Kvasnicka [27]). The exact details depend on whether the upper or lower triangular part of the symmetric matrix A is referenced.

The following section will illustrate that poor data-locality cannot be avoided if the goal is to reduce A to tridiagonal form in *one sweep*. In this case one Householder vector u_j is constructed for each column j of A such that *all* the elements in rows $j+2, j+3, \dots, n$ are eliminated. In total, $n-2$ different Householder vectors are required for the full one-sweep tridiagonalization process.

matrix A. An algorithm operating on the upper triangular part of A has similar data access patterns.

4.3 Limitations of Blocking

Having investigated the drawbacks of the LAPACK approach to blocked tridiagonalization, the following questions are to be asked:

- Is blocking with localized data access patterns in the sense of Section 3.4 possible in the standard tridiagonalization process?
- If it is not, why?
- What is the best achievable data-locality?
- How can an algorithm with better data-locality than the one used in LAPACK be designed?

In this section it will become clear that the answer to the first question is "No" – it is *not* possible to fully block tridiagonalization *in one sweep*. The reason for this will be given in this section. The third and fourth question will be addressed in Section 4.4.

4.3.1 General Similarity Transformations

As a starting point, the general form of a similarity transformation

$$A \to X^{-1}AX =: A''$$

with a nonsingular matrix X be considered. This transformation can be split into two operations:

1. In the first operation, $A' := X^{-1}A$, the computation of the element $A'(i,j)$ references the ith row of X^{-1}, $X^{-1}(i,:)$, and the jth column of A, $A(:,j)$.
2. In the second operation, $A'' := A'X$, the computation of the element $A''(i,j)$ references the row $A'(i,:)$ and the column $X(:,j)$.

Fig. 4.1 illustrates the reference patterns in the three matrices involved. The important aspect here is that the calculation of *any* element $A''(i,j)$ references a row $A'(i,:)$. The computation of this row in turn references *all* the elements of A as can be seen in Step 1 above. Hence it is not possible to update a block of A separately without referencing all of A. This shows that a similarity transformation cannot be blocked with localized data access patterns (as introduced in Section 3.4).[5]

[5]Actually, only *one-sided* matrix updates can be formulated with localized data access patterns. In this case, parts of A can be fully updated without having to access the entire matrix.

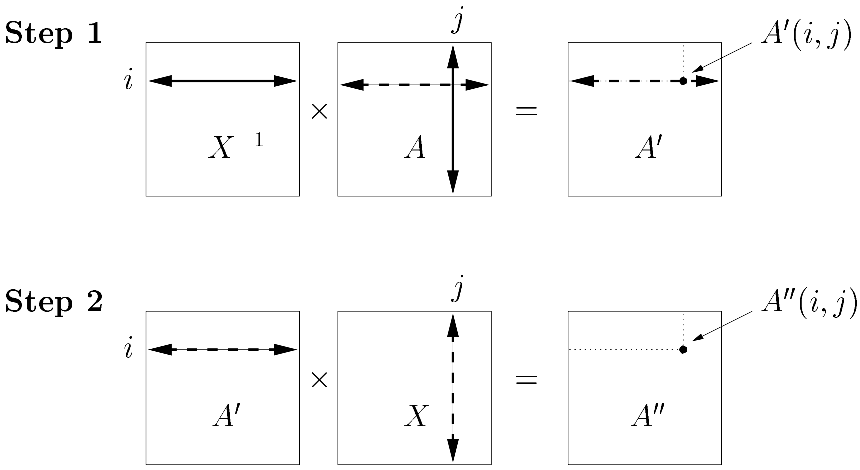


Figure 4.1: Data access patterns for a general similarity transformation. Solid lines indicate data accesses during the first operation, dashed lines show data accesses during the second operation.

4.3.2 Similarity Transformations Using Householder Transformations

The tridiagonalization process consists of a sequence of special cases of similarity transformations—Householder reflections. In the following the special structure of Householder reflectors will be taken into account to investigate the potential for blocking.

Partitioning A into column blocks $A = (A_1 \mid A_2)$ $(A_1 \in \mathbb{R}^{n\times b}, A_2 \in \mathbb{R}^{n\times(n-b)})$, writing the Householder reflector as $H = I_n - 2uu^\top$, and partitioning the Householder vector $u^\top = (u_1^\top \mid u_2^\top)$ $(u_1 \in \mathbb{R}^b, u_2 \in \mathbb{R}^{n-b})$ implies

$$H^{-1}AH = (HA_1 \mid HA_2)(I_n - 2uu^\top) = (HA_1 \mid HA_2) - 2(HA_1u_1u^\top + HA_2u_2u^\top)$$

$$= (HA_1 \mid HA_2) - 2(HA_1u_1u_1^\top + HA_2u_2u_1^\top \mid HA_1u_1u_2^\top + HA_2u_2u_2^\top).$$

Consequently, the two column blocks of A have to be updated according to

$$\begin{aligned} A_1 &:= HA_1 - 2(HA_1u_1 + HA_2u_2)u_1^\top \\ A_2 &:= HA_2 - 2(HA_1u_1 + HA_2u_2)u_2^\top \end{aligned}$$

In order to keep references local, A_1 would have to be updated *without accessing* A_2. Such local updating can only be done if

---

If $A := XA$ columns can be blocked, and if $A := AX$ rows can be blocked.

either u_1 has only zero components, which means that only rows and columns $b+1$ through n of A are combined,

or u_2 has only zero components, which means that only rows and columns 1 through b of A are combined.

Otherwise A_2 will contribute to the update of A_1 and A_1 will contribute to the update of A_2.

In general, none of these two requirements can be satisfied, and hence in general the special structure of Householder reflectors will not enable local references for updating a column block. It is clear now that the process of tridiagonalizing a matrix A cannot entirely avoid these dependencies and thus has inherent drawbacks in terms of data-locality.

Given the fact that tridiagonalization *in one sweep* does not allow for localized data access patterns and that consequently the blocked LAPACK version does not exhibit satisfactory data-locality, alternative ways of blocking the tridiagonalization process had to be investigated.

4.4 Tridiagonalizing in Multiple Sweeps

An alternative to the blocked LAPACK version is not to tridiagonalize in one sweep, but to allow for *multiple* (two or more) sweeps. This means that the Householder vector u_j $(j = 1, 2, \ldots, n-2)$ of the first sweep does *not* eliminate all the elements $j+2, j+3, \ldots, n$ in column j, but only elements $j+b+1, j+b+2, \ldots, n$. Therefore, a first sweep of $n-2$ Householder reflections transforms matrix A into a *banded* matrix. At least one more sweep of eliminations (based on Householder reflections and/or Givens rotations) is required to reach the tridiagonal form.

Some work has recently been done on the blocked reduction of the bandwidth of banded matrices (Bischof and Sun [9], Lang [32], Bischof et al. [5], Bischof et al. [6, 7]). The method described here is very similar, but it is more extensively related to the computation of eigenvectors.

The basic idea is as follows: Matrix A is partitioned into submatrices based on the generic partitioning shown in Fig. 4.2, which is applied recursively. By "tridiagonalizing" the resulting blockmatrix at the submatrix level (thereby eliminating blocks below the subdiagonal blocks) transformation to a band matrix B is achieved.

This "tridiagonalization" at the submatrix level has localized data access patterns. The dependencies between two neighboring blocks do not lead to unfavorable data references. Only the subsequent reduction of the banded matrix to a tridiagonal matrix (at the element level) by a later sweep makes it necessary to deal with data dependencies between block-columns. The reduction from banded to tridiagonal form can be done in *one* sweep or by gradually reducing the bandwidth in *several* sweeps, eventually ending up with a tridiagonal matrix (Bischof et al. [6, 7]).

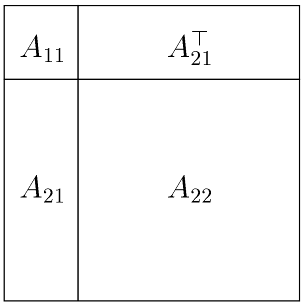


Figure 4.2: Generic partitioning of a symmetric matrix for blocking: $A_{11} \in \mathbb{R}^{b\times b}$, $A_{21} \in \mathbb{R}^{(n-b)\times b}$, $A_{22} \in \mathbb{R}^{(n-b)\times(n-b)}$.

The advantage of this approach is fairly clear. The main part of the work (the first sweep) is fully blocked, has localized data access patterns, and therefore can be done very efficiently. After the first sweep the band matrix B of bandwidth $w = 2b + 1$ is obtained.[6] In particular for small values of b memory requirements for storing this band matrix are negligible and the work required to reduce such a matrix to tridiagonal form is small compared with the work needed for the first sweep.

4.4.1 Computing Eigenvectors

For a large enough block size b the multi-sweep approach is advantageous if only eigenvalues have to be calculated (Bischof et al. [6]). If eigenvectors have to be computed as well there is a tradeoff: Every new sweep represents additional work in the backtransformation of the eigenvectors from the tridiagonal matrix T.

In the following it will be assumed that k eigenvalues and the corresponding k eigenvectors ($k < n$, in the case of WIEN 97 $k \approx n/10$) have to be computed. The eigenvectors are collected in a matrix $Y \in \mathbb{R}^{n\times k}$. The following workload comparison for the backtransformation of eigenvectors is obtained.

[6] Since all the matrices are symmetric, b = lower bandwidth = upper bandwidth := number of subdiagonals. At the same time, b turns out to be the maximum possible block size for the first sweep.

1. For **single-sweep tridiagonalization** (full $\to$ tridiagonal), the backtransformation can be applied to the eigenvectors Y computed from the tridiagonal form in $5n^2k/2 + O(nk)$ flops (starting with the Householder vectors and taking advantage of their sparsity structure).

2. For **two-sweep tridiagonalization** (full $\to$ banded $\to$ tridiagonal) the best way to perform the backtransformation of the eigenvectors Y would be to accumulate all the transformation data in one single matrix Q.

   The explicit construction of Q after the first reduction sweep requires $4(n-b)^3/3+O(n^2)$ flops. During the second reduction sweep, Q has to be updated after every single transformation (Givens rotation or Householder reflection) performed on the band matrix. Eventually, the transformation matrix Q needs to be multiplied with the eigenvector matrix Y, which requires $2n^2k+O(nk)$ flops.

It is clear that by introducing a second sweep better memory access patterns are traded for more work. Despite the improved data-locality of the first sweep, the additional work in the backtransformation process for the eigenvectors causes any multi-sweep eigensolver to be less efficient than the one-sweep eigensolver on modern computer systems, if the eigenvectors are computed from the tridiagonal matrix T (Bischof et al. [6]).

In order to avoid this slow-down problem it was decided to calculate the eigenvectors *from the band matrix B* directly (see Fig. 4.3). Then only *one* backtransformation step has to be stored and applied to the eigenvectors of B.

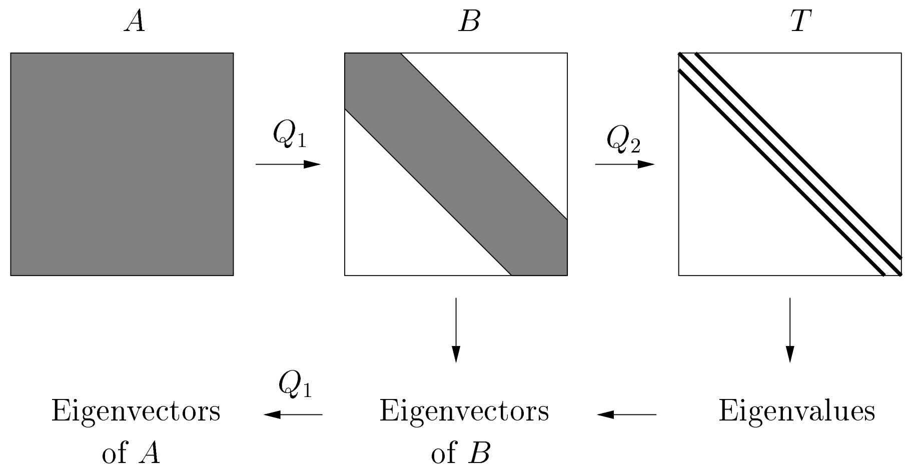


Figure 4.3: The principle of the new two-sweep eigensolver.

It is necessary to keep in mind that the extra work introduced (compared to single-sweep tridiagonalization) is proportional to the number k of eigenvectors needed, because inverse iteration on B involves more flops than inverse iteration on T (see Section 3.3.2).

For solving a SEP when only a few of the eigenvectors are needed (like, for example, in the context of WIEN 97) Algorithm 4.4.1 can be recommended as very efficient algorithm (for experimental results, see Gansterer et al. [23]).

Algorithm 4.4.1 Generic Two-Sweep Algorithm for SEP

1. **Eliminate** *elements in the current block column of* A *such that localized updating is possible.*
2. **Accumulate** *all the transformations done in Step 1 by computing a corresponding transformation matrix* Q.
   Q *can be represented in different ways:*
   $Q = I_n - WY^\top$ *or* $Q = I_n - YTY^\top$ *or* $Q = I_n - GG^\top$
   (see Sections 5.1, 5.2, 5.3).
3. **Update** *the remaining parts of* A *using* Q.

Steps 1 to 3 are iterated from left to right over block columns of A *(each consisting of* b *columns). When all the block columns are processed,* A *has been transformed into a banded matrix* B *with bandwidth* $w = 2b + 1$.
All the transformation matrices Q *computed in Step 2 can be accumulated into a single orthogonal transformation* Q_1 *such that* B *can be expressed as* $B = Q_1^\top A Q_1$. *(The matrix* Q_1 *need not be constructed explicitly for this update.)*

4. **Calculate eigenvalues** *by further transforming* B *into a tridiagonal matrix* $T = Q_2^\top B Q_2$ *(with an orthogonal transformation* Q_2*) and then using, for example, a bisection method.*
5. **Calculate eigenvectors** y *of* B *via inverse iteration on the banded matrix* B *using the eigenvalues computed in Step 4.*
6. **Transform** *the eigenvectors* y *of* B *calculated in Step 5 to eigenvectors* x *of* A *according to* $x := Q_1 y$.

In the following section and in the next chapter different ways to perform Steps 1, 2, and 3 of Algorithm 4.4.1, that is, to eliminate blocks of A, to represent the corresponding transformations, and to perform the update operations will be described. Current research is concentrating on improved versions of the elimination phase (Step 1). In the future another important area will be the implementation of efficient algorithms for Step 5 (see Section 6.1, and also Gansterer and Ueberhuber [25]).

4.5 A Variant based on QR Factorization

One possibility for performing Steps 1 to 3 of Algorithm 4.4.1 (the reduction to band form) is shown in Algorithm 4.5.1 (see also Bischof et al. [6]). The resulting matrix B is banded with bandwidth $w = 2b + 1$, as shown in Fig. 4.4. One big advantage of this approach is that good software (LAPACK [1]) is readily available for all the building blocks.

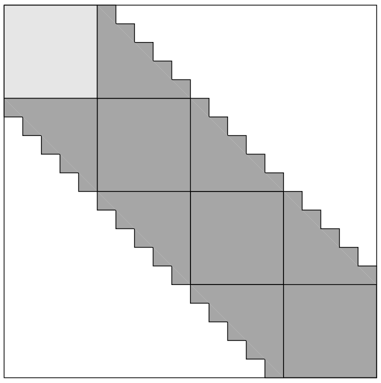


Figure 4.4: Resulting band matrix B after a first elimination sweep using QR factorizations ($n = 20$, $b = 5$).

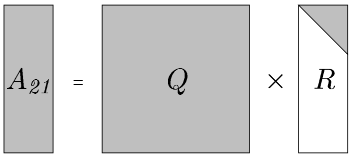
The description of Algorithm 4.5.1 does not go into all details. Note, several important issues, in particular, how to accumulate a sequence of Householder reductions into a single transformation Q, are "hidden" in the "black box" QR factorization (Step 2 in Algorithm 4.5.1). In Chapter 5 several ways to accumulate the transformation information will be investigated in more detail.

Algorithm 4.5.1 QR-Based Reduction to Band Structure

do $i = 0, 1, 2, \ldots$

1. partition $A(ib+1:n, ib+1:n)$ *according to Fig. 4.2.*

2. factorize $A_{21} = QR$,



where $Q := (I_{n-(i+1)b} - 2u_{i1}u_{i1}^\top) \cdots (I_{n-(i+1)b} - 2u_{ib}u_{ib}^\top)$
is the product of b *Householder reflections,* $R \in \mathbb{R}^{(n-(i+1)b)\times b}$.
Various ways to represent the matrix $Q \in \mathbb{R}^{(n-(i+1)b)\times(n-(i+1)b)}$ *are outlined in Chapter 5.*

3. update

$$\begin{pmatrix} I_b & 0 \\ 0 & Q^\top \end{pmatrix} \begin{pmatrix} A_{11} & A_{21}^\top \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} I_b & 0 \\ 0 & Q \end{pmatrix} = \begin{pmatrix} A_{11} & R^\top \\ R & Q^\top A_{22} Q \end{pmatrix},$$

hence $A_{21} := R$, $A_{22} := Q^\top A_{22} Q$.

if A_{22} *is a* $l \times l$ *block with* $l \leq b$ **then exit**

end do

Chapter 5

Accumulating Householder Transformations

A crucial step for the performance of a blocked tridiagonalization algorithm is the accumulation of b single Householder reflectors into one matrix in order to update unreduced parts of the remaining matrix.

There are three well known ways to represent a product of Householder transformations:

1. the $\mathbf{WY}^\top$**representation** (Bischof and Van Loan [10]);
2. a variant thereof, the **compact** $\mathbf{WY}^\top$ **representation** or $\mathbf{YTY}^\top$ **representation** (Schreiber and Van Loan [36]); and
3. the $\mathbf{GG}^\top$ **representation** (**block reflector** approach, Schreiber and Parlett [35]).

Computing the $GG^\top$ representation requires $O(b^3)$ more flops than the two other options. Since normally $n \gg b$ in the application considered this is not a serious disadvantage. Therefore, all three approaches for accumulating Householder transformations will be discussed in Sections 5.1, 5.2, and 5.3.

Section 5.4 will briefly address the task of applying the accumulated transformations to a matrix. A more detailed comparison of various representations of combined Householder transformations is given in Bischof [4].

5.1 WY$^\top$ Representation

The product $Q = H_1 \cdots H_b$ of b Householder matrices generated by the Householder vectors $u_1, \ldots, u_b \in \mathbb{R}^n$ can be represented as $Q = I_n - WY^\top$, where both $Y \in \mathbb{R}^{n\times b}$ and $W \in \mathbb{R}^{n\times b}$ are upper trapezoidal ($Y(i,j) = W(i,j) = 0$ for $i < j$, see Fig. 5.1). The matrices W and Y can be constructed iteratively, as shown in Algorithm 5.1.1. Y is simply the aggregation of the b Householder vectors. The computation of W requires *extra* work, which is not required in unblocked tridiagonalization, where each Householder transformation is applied to the unreduced part immediately (see Algorithm 4.1.1).

Algorithm 5.1.1 Construction of W and Y

$W := (2u_1)$; $Y := (u_1)$
do $j = 2, 3, \ldots, b$
  $z := 2(I_n - WY^\top)u_j$
  $Y := (Y \mid u_j)$
  $W := (W \mid z)$
end do

Because of the sparsity structure in the Householder vectors of the reduction to banded form, Y and W are essentially $(n - (i+1)b) \times b$ matrices at iteration number i of Algorithm 4.5.1. All information needed to recover the transformations is contained in the Y matrices, which can be stored in the eliminated parts of the array A. Since the overhead for storing the W matrices is considerable, they are usually not stored after the updating operation, but *recomputed* at the time of backtransformation of the eigenvectors. This, however, implies some computational overhead.

5.2 YTY$^\top$ Representation

A slight modification of the $WY^\top$ representation makes it possible to write Q as $Q = I_n - YTY^\top$ with an upper triangular matrix $T \in \mathbb{R}^{b\times b}$, and Y being the same matrix as in Section 5.1 (see Fig. 5.1). Y and T are easily constructed in a way similar to W and Y in Section 5.1 (see Algorithm 5.2.1).

The main advantage of this representation is that its storage requirements are much lower than those of the $WY^\top$ representation. It is only necessary to store an upper triangular $b\times b$ matrix T instead of an $n\times b$ matrix W. In practical situations

Algorithm 5.2.1 Construction of Y and T

$Y := (u_1);\ T := (2)$
do $j = 2, 3, \ldots, b$
  $z := -2(TY^\top)u_j$
  $Y := (Y \mid u_j)$
  $T := \begin{pmatrix} T & z \\ 0 & 2 \end{pmatrix}$
end do

typical parameter values could be $n = 4000$ and $b = 32$ (see Gansterer et al. [23]). Thus storage efficiency is increased significantly. Moreover, the construction of the matrix T requires slightly fewer operations than the accumulation of the W matrix.

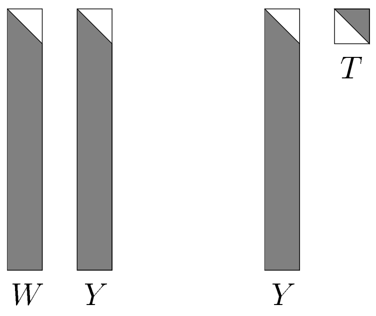


Figure 5.1: Storage requirements for $WY^\top$ and $YTY^\top$ representations. (Y can be stored in unused parts of the array A.)

There is, however, a slight disadvantage of this representation when the combined transformations need to be applied to a matrix, which occurs in the update operation of the tridiagonalization process and also during backtransformation of the eigenvector matrix (see Section 5.4).

5.3 $\mathbf{GG}^\top$ Representation

In the blocked tridiagonalization process blocks of A are eliminated instead of one column at a time. The task of *block elimination* can be performed in a very compact way by using the concept of a *block reflector*. This is a natural generalization of Householder reflectors (see Section 3.2.1), which are used to perform vectorwise elimination. Schreiber and Parlett [35] introduce block reflectors as orthogonal, symmetric matrices with possibly more than one negative eigenvalue. They develop a theory of block reflectors and also give algorithms to compute them.

Definition 5.3.1 (Block Elimination) *Given a matrix* $E = \begin{pmatrix} E_1 \\ E_2 \end{pmatrix} \in \mathbb{R}^{n\times b}$*, where* $E_1 \in \mathbb{R}^{b\times b}$ *and* $n > b$*, a* block reflector *is an orthogonal symmetric* $n \times n$ *matrix* H*, such that*

$$HE = \begin{pmatrix} F \\ 0 \end{pmatrix}, \quad F \in \mathbb{R}^{b\times b}. \tag{5.1}$$

(5.1) is the general formulation of a task to be performed repeatedly in the framework of blocked elimination processes. Schreiber and Parlett [35] give certain conditions on the matrix F in (5.1) which are necessary and sufficient for the existence of a block reflector H. They also give an expression for H if a suitable F is known. Since this is usually not the case, this representation of a block reflector is in general not useful for practical purposes.

One obvious approach for constructing a block reflector H is the accumulation of b Householder (vector-)reflectors as discussed in Sections 5.1 and 5.2. QR factorization (Golub and Van Loan [26]) of E, i. e.,

$$E = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} F \\ 0 \end{pmatrix}$$

yields $H = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}^\top$. In this case the matrix F is not a full matrix but it has upper triangular form, and a reduction sweep based on block reflectors constructed this way leads to a band matrix of bandwidth $2b+1$ (see Algorithm 4.5.1 in Section 4.5 and Fig. 4.4). In general, however, block elimination according to Definition 5.3.1 does not yield an upper triangular F.

5.3.1 Compact Representation of Block Reflectors

In addition to the $WY^\top$ and the $YTY^\top$ representation of a block reflector as discussed in Sections 5.1 and 5.2 there is a third representation, which might be of practical interest: Assuming that E has rank $r \leq b$, it is possible to find a matrix $G \in \mathbb{R}^{n\times r}$, such that $H = I_n - GG^\top$ with $G^\top G = 2I_r$. This representation allows to take advantage of a situation where E happens not to have full column rank. But also if E has rank b, it has advantages in terms of storage efficiency compared to $WY^\top$ representation, and also in terms of the backtransformation of eigenvectors compared to the $YTY^\top$ representation.

Obviously, the central question is how to construct the matrix G. Algorithm 5.3.1 is given in Schreiber and Parlett [35] and also in Schreiber and Van Loan [36].

Algorithm 5.3.1 Construction of G

1. *Compute Householder reflections* $H_1, \ldots, H_r$, *such that*
$$H_r \cdots H_1 E = \begin{pmatrix} T \\ 0 \end{pmatrix}$$
with $T \in \mathbb{R}^{r\times b}$ *and lower trapezoidal* ($T(i,j) = 0$ *for* $i > j$).
2. *Let* $\begin{pmatrix} U_1 \\ U_2 \end{pmatrix}$ *consist of the first* r *columns of the product*
$$Q := H_1 \cdots H_r \text{ with } U_1 \in \mathbb{R}^{r\times r}, U_2 \in \mathbb{R}^{(n-r)\times r}.$$
3. *Compute the* polar decomposition $U_1 = ZM$
with orthogonal $Z \in \mathbb{R}^{r\times r}$ *and symmetric*
nonnegative definite $M \in \mathbb{R}^{r\times r}$.
4. *Then* $F := \begin{pmatrix} -ZT \\ 0 \end{pmatrix}$.
5. *Compute the Cholesky factorization* $I_r + M = LL^T, L \in \mathbb{R}^{r\times r}$.
6. *Then* $G \in \mathbb{R}^{n\times r}$ *can be computed as*
$$G := \begin{pmatrix} ZL \\ U_2 L^{-\top} \end{pmatrix}.$$

The *polar decomposition* of U_1 can be computed using an SVD-based method (Golub and Van Loan [26]); or, alternatively, using an iterative, quadratically converging algorithm suggested by Higham [28]. This algorithm involves matrix inversions, which can themselves be approximated by an inner iteration.

Algorithm 5.3.1 for generating the matrix G requires more work than the Algorithms 5.1.1 and 5.2.1 for computing $WY^\top$ or $YTY^\top$ representations since it analogously requires the explicit construction of r single transformations in Step 1 and accumulation of the first r columns of the combined transformation in Step 2. Additional work is required in Steps 3 to 6 to find the more compact representation.

The $GG^\top$ representation is still interesting for several reasons: On the one hand—although the additional work involves $O(r^3)$ operations—the computational overhead is small, since usually $r \ll n$. On the other hand, as mentioned before, an advantage of the $GG^\top$ representation is the low storage requirements for keeping all the transformation information (compared to the $WY^\top$ representation). Moreover, the effort of computing W matrices anew at the time of backtransformation (see Sections 5.1, 5.2) can be saved.

One of the future goals is to further pursue this approach and to find out how efficient (and numerically stable) implementations can be achieved. Some (error-prone) coding effort will be required. This is probably the reason why this method has not been used so far.

5.4 Comparison

Bischof [4] compares the methods for representing block transformations illustrated in Sections 5.1, 5.2, and 5.3. Of particular interest are storage requirements and the work required when applying them to a matrix.

As already mentioned in the previous sections, the $YTY^\top$ representation and the $GG^\top$ representation are most storage efficient. The former is also most efficient in terms of the work required for actually generating the factors of the representation.

In all three representations ($WY^\top$, $YTY^\top$, $GG^\top$), about $2nkb$ flops are required for applying a block transformation to k eigenvectors (assuming that G has rank b). There is, however, a slight drawback in applying a block transformation in the $YTY^\top$ representation, where these operations occur as *three* separate matrix-matrix operations instead of two (Schreiber and Van Loan [36]).

Chapter 6

Potential for Future Development

6.1 Eigenvectors of Band Matrices

The standard procedure for computing an eigenvector of a symmetric matrix using a known eigenvalue is inverse iteration (see Section 3.3).

As explained in Section 4.4 in the context of two-sweep tridiagonalization it is preferable to perform inverse iteration on the banded matrix B directly, and *not* on the tridiagonal matrix T.

In experiments (Gansterer et al. [23]) the routine `EISPACK/bandv` from the EISPACK library (Smith et al. [40]) has been used for performing inverse iteration on the band matrix B. The block size b_2^* which yields the best performance for the two-sweep reduction is *larger* than the block size b_1^* which yields the best performance for a complete eigensolver, which includes the computation of eigenvectors. This is due to the fact that larger block sizes increase the workload for doing inverse iteration on the band matrix B. In order to be able to use block sizes as large as possible in the complete eigensolver, it is important to develop very efficient algorithms for computing eigenvectors of a band matrix.

Avoiding Tridiagonalization. A very interesting approach might be to compute both eigenvalues and eigenvectors from the band matrix B directly and thus to completely avoid the reduction to tridiagonal form T and its data dependencies. Attempts will be made to develop an efficient and stable method for generalizing Cuppen's divide-and-conquer algorithm (Cuppen [15]) to a band matrix (see Arbenz and Golub [3] and Arbenz et al. [2]).

Alternatively, generalization of a new and embarassingly parallel method for computing eigenvectors of a symmetric tridiagonal matrix (Dhillon [18]) to band matrices will be investigated.

6.2 Alternative Methods

One of the future tasks will be to look carefully at the block Davidson method (see Wood and Zunger [44]) currently used in WIEN 97 (Singh [37]). The goal will be to accelerate convergence of this method. Recently, a lot of progress has been made on the *Jacobi-Davidson method* (Sleijpen and van der Vorst [39]), which is a variant of the original Davidson method (Davidson [16]). It will be interesting to compare the block Davidson method currently in use to the Jacobi-Davidson method and to take advantage of recent developments.

If the convergence of the block Davidson method can be improved it will not be necessary to use the more time-consuming tridiagonalization process as often. This might lead to significant performance improvements.

Another possible development to be investigated could be an attempt to modify the underlying LAPW method (see Chapter 2). In particular, there seems to be some flexibility in the choice of the basis functions $\Phi_\alpha(r)$ (see Equation (2.10) in Section 2.2). If they can be chosen with limited support then the overlap matrix S would not be a full matrix, but banded (see Section 2.2.1). This can lead to a significant work reduction in the Cholesky factorization of S which is required in the transformation of the generalized eigenproblem to a standard eigenproblem (see Section 3.1.1).

6.3 Parallelization

Significant performance improvements of the sequential WIEN 97 code have been achieved. The next goal is to develop an efficient parallel version of the code. Parallel eigensolvers have been developed based on single-sweep tridiagonalization (Dongarra and van de Geijn [21], Choi et al. [14], ScaLapack [11]).

A lot of work has also been done on parallel band reduction and on parallel multi-sweep tridiagonalization (Lang [32], Bischof et al. [8], Bischof et al. [5], Wu et al. [45]). The integration of the new approach described in this report into this work is the topic of current and future research activities.

Chapter 7

Conclusion

A Level 3 algorithm with localized data access patterns for the tridiagonalization of symmetric matrices has been discussed.

Analysis of the dependencies in the tridiagonalization process shows that a blocked single-sweep reduction of a dense matrix to tridiagonal form requires global data accesses in every iteration step. Multi-sweep methods reduce a full matrix to banded form in a fully blocked first sweep with better data-locality. At least one more sweep is required to reduce the band matrix to tridiagonal form.

This approach was already proposed by Bischof et al. ([6]). In addition to this multi-sweep approach the implications of having to compute eigenvectors along with eigenvalues have been discussed in detail, and a method to do this efficiently has been proposed.

The implementation of a two-sweep approach not only showed significant performance gains on various sequential computer systems (see Gansterer et al. [23]), preliminary experiments on parallel machines also show major performance improvements compared to already existing algorithms and codes.

Acknowledgement

We would like to thank Karlheinz Schwarz and Peter Blaha for providing us with information on the program package WIEN 97.

In addition, we would like to acknowledge the financial support of the Austrian Science Fund FWF.

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. C. Sorensen: LAPACK *User's Guide*. 2nd edn. SIAM Press, Philadelphia 1995.

[2] P. Arbenz, W. Gander, G. H. Golub: *Restricted Rank Modification of the Symmetric Eigenvalue Problem: Theoretical Considerations*. Linear Algebra Appl. 104 (1988), pp. 75–95.

[3] P. Arbenz, G. H. Golub: *On the Spectral Decomposition of Hermitian Matrices Modified By Low Rank Perturbations With Applications*. SIAM J. Matrix Anal. Appl. 9 (1988), pp. 40–58.

[4] C. H. Bischof: *A Summary of Block Schemes for Reducing a General Matrix to Hessenberg Form*. Report ANL/MCS-TM-175, Argonne National Laboratory, 1993.

[5] C. H. Bischof, B. Lang, X. Sun: *Parallel Tridiagonalization through Two-Step Band Reduction*. Preprint ANL/MCS-P412-0194, Math. & Comp. Sci. Div., Argonne National Laboratory, 1994.

[6] C. H. Bischof, B. Lang, X. Sun: *A Framework for Symmetric Band Reduction*. Preprint ANL/MCS-P586-0496, Math. & Comp. Sci. Div., Argonne National Laboratory, 1996.

[7] C. H. Bischof, B. Lang, X. Sun: *The SBR Toolbox—Software for Successive Band Reduction*. Preprint ANL/MCS-P587-0496, Math. & Comp. Sci. Div., Argonne National Laboratory, 1996.

[8] C. H. Bischof, M. Marques, X. Sun: *Parallel Bandreduction and Tridiagonalization. Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing* (edited by R. Sincovec). SIAM Press, 1993, pp. 383–390.

[9] C. H. Bischof, X. Sun: *A Framework for Band Reduction and Tridiagonalization of Symmetric Matrices*. Preprint ANL/MCS-P298-0392, Math. & Comp. Sci. Div., Argonne National Laboratory, 1992.

[10] C. H. Bischof, C. F. Van Loan: *The WY Representation for Products of Householder Matrices*. SIAM J. Sci. Comput. 8 (1987), pp. s2–s13.

[11] L. S. Blackford et al.: ScaLapack *Users' Guide*. SIAM Press, Philadelphia 1997.

[12] P. Blaha, K. Schwarz, J. Luitz: Wien 97. Improved and updated UNIX version of the original copyrighted Wien code [13], 1997.

[13] P. Blaha, K. Schwarz, P. Sorantin, S. B. Trickey: *Full-Potential, Linearized Augmented Plane Wave Programs for Crystalline Systems*. Comput. Phys. Comm. 59 (1990), pp. 399–415.

[14] J. Choi, J. J. Dongarra, D. Walker: *The Design of a Parallel, Dense Linear Algebra Software Library: Reduction to Hessenberg, Tridiagonal, and Bidiagonal Form*. Numer. Algorithms (1995), pp. 379–400.

[15] J. J. M. Cuppen: *A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem*. Numer. Math. 36 (1981), pp. 177–195.

[16] E. R. Davidson: *The Iterative Calculation of a Few of the Lowest Eigenvalues and Corresponding Eigenvectors of Large Real Symmetric Matrices*. J. Comput. Phys. 17 (1975), pp. 87–94.

[17] J. W. Demmel: *Applied Numerical Linear Algebra*. SIAM Press, Philadelphia 1997.

[18] I.S. Dhillon: *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*. Ph.D. thesis, Computer Science Division (EECS), University of California at Berkeley, 1997.

[19] J. J. Dongarra, I. S. Duff, D. C. Sorensen, H. A. van der Vorst: *Numerical Linear Algebra for High-Performance Computers*. SIAM Press, Philadelphia 1998.

[20] J. J. Dongarra, S. J. Hammarling, D. C. Sorensen: *Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations*. J. Comput. Appl. Math. 27 (1989), pp. 215–227.

[21] J. J. Dongarra, R. A. van de Geijn: *Reduction to Condensed Form for the Eigenvalue Problem on Distributed-Memory Architectures*. Parallel Comput. 18 (1992), pp. 973–982.

[22] W. N. Gansterer, D. F. Kvasnicka, C. W. Ueberhuber: *Efficiency Enhancement by Blocking*. Technical Report AURORA TR1998-10, Vienna University of Technology, 1998.

[23] W. N. Gansterer, D. F. Kvasnicka, C. W. Ueberhuber: *High Performance Computing in Material Sciences. Numerical Experiments with Symmetric Eigensolvers*. Technical Report AURORA TR1998-19, Vienna University of Technology, 1998.

[24] W. N. Gansterer, D. F. Kvasnicka, C. W. Ueberhuber: *Multi-Sweep Algorithms for the Symmetric Eigenproblem*. *Proceedings of the Third International Meeting on Vector and Parallel Processing (VECPAR 98)* (edited by J. M. L. M. Palma, J. J. Dongarra, V. Hernandez), Lecture Notes in Computer Science, 1998. to appear.

[25] W. N. Gansterer, C. W. Ueberhuber: *Adaptive Inner-Outer Inverse Iteration for Eigenvector Computations*. Technical Report AURORA TR1998-04, Institute for Applied and Numerical Mathematics, Vienna University of Technology, 1998.

[26] G. H. Golub, C. F. Van Loan: *Matrix Computations*. 3rd edn. Johns Hopkins University Press, Baltimore London 1996.

[27] E. Haunschmid, D. F. Kvasnicka: *High Performance Computing in Material Sciences. Maximizing Cache Utilization Without Increasing Memory Requirements*. Technical Report AURORA TR1998-17, Vienna University of Technology, 1998.

[28] N. J. Higham: *Computing the Polar Decomposition with Applications*. SIAM J. Sci. Comput. 7 (1986), pp. 1160–1174.

[29] P. Hohenberg, W. Kohn: *Inhomogeneous Electron Gas*. Phys. Rev. B 136 (1964), pp. 864–871.

[30] I. C. F. Ipsen: *Computing an Eigenvector With Inverse Iteration*. SIAM Rev. 39 (1997), pp. 254–291.

[31] W. Kohn, L. J. Sham: *Self-Consistent Equations Including Exchange and Correlation Effects*. Phys. Rev. A 140 (1965), pp. 1133–1138.

[32] B. Lang: *A Parallel Algorithm for Reducing Symmetric Banded Matrices to Tridiagonal Form*. SIAM J. Sci. Comput. (1993), pp. 1320–1338.

[33] R. B. Lehoucq: *The Computation of Elementary Unitary Matrices*. ACM Trans. Math. Software 22 (1996), pp. 393–400.

[34] B. N. Parlett: *The Symmetric Eigenvalue Problem*. Prentice Hall, Englewood Cliffs 1980.

[35] R. Schreiber, B. N. Parlett: *Block Reflectors: Theory and Computation*. SIAM J. Numer. Anal. 25 (1988), pp. 189–205.

[36] R. Schreiber, C. F. Van Loan: *A Storage-efficient WY Representation for Products of Householder Transformations*. SIAM J. Sci. Comput. 10 (1989), pp. 53–57.

[37] D. J. Singh: *Simultaneous Solution of Diagonalization and Self-consistency Problems for Transition-Metal Systems*. Phys. Rev. B 40 (1989), pp. 5428–5431.

[38] D. J. Singh: *Planewaves, Pseudopotentials and the LAPW Method*. Kluwer Academic Publishers, Boston Dordrecht London 1994.

[39] G. L. G. Sleijpen, H. A. van der Vorst: *A Jacobi-Davidson Iteration Method for Linear Eigenvalue Problems*. SIAM J. Matrix Anal. Appl. 17 (1996), pp. 401–425.

[40] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, C. B. Moler: *Matrix Eigensystem Routines*—EISPACK *Guide*, Vol. 6 of *Lecture Notes in Computer Science*. 2nd edn. Springer-Verlag, Berlin Heidelberg New York Tokyo 1976.

[41] C. W. Ueberhuber: *Numerical Computation*. Springer-Verlag, Berlin Heidelberg New York Tokyo 1997.

[42] J. H. Wilkinson: *The Algebraic Eigenvalue Problem*. Oxford University Press, Oxford 1965.

[43] J. H. Wilkinson: *Inverse Iteration in Theory and Practice*. Sympos. Math. 10 (1972), pp. 361–379.

[44] D. M. Wood, A. Zunger: *A New Method for Diagonalizing Large Matrices*. J. Phys. A 18 (1985), pp. 1343–1359.

[45] Y.-J. J. Wu, P. A. Alpatov, C. H. Bischof, R. A. van de Geijn: *A Parallel Implementation of Symmetric Band Reduction Using* PLAPACK. *Proceedings of the Scalable Parallel Libraries Conference, Starkville, MS*, 1996. to appear.