A Level 3 Algorithm for the Symmetric Eigenproblem

Dieter F. Kvasnicka[1], Wilfried N. Gansterer[2], and Christoph W. Ueberhuber[3]

[1] Institute for Technical Electrochemistry,
University of Technology, Vienna
dieter@titania.tuwien.ac.at

[2] Institute for Applied and Numerical Mathematics,
University of Technology, Vienna
ganst@aurora.tuwien.ac.at

[3] Institute for Applied and Numerical Mathematics,
University of Technology, Vienna
christof@uranus.tuwien.ac.at

Abstract. This paper shows how the symmetric eigenproblem, which is the computationally most demanding part of numerous scientific and industrial applications, can be solved much more efficiently than by using algorithms currently implemented in LAPACK routines.
The main techniques used in the algorithm presented in this paper are (i) sophisticated blocking in the tridiagonalization, which leads to a two-sweep algorithm; and (ii) the computation of the eigenvectors of a band matrix instead of a tridiagonal matrix.
This new algorithm improves the locality of data references and leads to a significant improvement in the floating-point performance of symmetric eigensolvers on modern computer systems. Speedup factors of up to four (depending on the computer architecture and the matrix size) have been observed.

Keywords: Numerical Linear Algebra, Symmetric Eigenproblem, Tridiagonalization, Performance Oriented Numerical Algorithm, Blocked Algorithm

1 Introduction

Reducing a dense symmetric matrix A to tridiagonal form T is an important preprocessing step in the solution of the symmetric eigenproblem. LAPACK (Anderson et al. [1]) provides a blocked tridiagonalization routine whose memory reference patterns are not optimal on modern computer architectures. In this LAPACK routine a significant part of the computation is performed by calls to Level 2 BLAS routines. Unfortunately, Level 2 BLAS do not have a ratio of floating-point operations to data movement that is high enough to enable efficient reuse of data that reside in cache or local memory (see Table 1). Thus,

This work was supported by the Austrian Science Foundation (FWF).

software construction based on calls to Level 2 routines is not well suited to computers with a memory hierarchy and multiprocessor machines.

Table 1. Ratio of floating-point operations to data movement for three closely related operations from the Level 1, 2, and 3 BLAS (Dongarra et al. [6])

BLAS	Routine	Memory Accesses	Flops	Flops per Memory Access
Level 1	`daxpy`	$3n$	$2n$	$2/3$
Level 2	`dgemv`	n^2	$2n^2$	2
Level 3	`dgemm`	$4n^2$	$2n^3$	$n/2$

Bischof et al. [4, 5] recently developed a general framework for reducing the bandwidth of symmetric matrices, which improves data locality and allows for the use of Level 3 BLAS instead of Level 2 BLAS.

In this paper we introduce an important modification to this framework in case eigenvectors have to be computed, too: Accumulating the transformation information for the eigenvectors incurs an overhead which can outweigh the benefits of a multisweep reduction (as remarked in Bischof et al. [4, 5]). Therefore we compute the required eigenvectors directly from the intermediate band matrix. Analyses and experimental results show that our approach to improving memory access patterns is superior to established algorithms in many cases.

2 Eigensolver with Improved Memory Access

A real symmetric $n \times n$ matrix A can be factorized as

$$A = V^T B V = Q^T T Q = Z^T \Lambda Z \tag{1}$$

where V, Q, and Z are orthogonal matrices. B is a symmetric band matrix with band width $2b+1$, T is a symmetric tridiagonal matrix, and Λ is a diagonal matrix whose diagonal elements are the eigenvalues of the (similar) matrices A, B and T. The column vectors of Z are the eigenvectors of A.

LAPACK reduces a given matrix A to tridiagonal form T by applying Householder similarity transformations. A bisection algorithm[1] is used to compute selected eigenvalues of T, which are also eigenvalues of A. The eigenvectors of T are found by inverse iteration on T. These eigenvectors have to be transformed into the eigenvectors of A using the transformation matrix Q.

The new method proposed does not compute the tridiagonal matrix T from A directly, but derives a band matrix B as an intermediate result. This band reduction can be organized with good data locality, which is critical for high performance on modern computer architectures. Using a block size b in the first

[1] If *all* eigenvalues are required, LAPACK also provides other algorithms.

reduction sweep results in a banded matrix B with a semibandwidth of at least b. This relationship leads to a tradeoff:

- If smaller values of b are chosen, then a larger number of elements of A are eliminated. This decrease in the number of non-zero elements leads to a smaller amount of data to be processed in later steps.
- If larger values of b are chosen, then better performance improvements are obtained in the first reduction sweep.

Using appropriate values of b the two-sweep tridiagonal reduction achieves speedups of up to ten as compared with the LAPACK tridiagonal reduction (see Gansterer, Kvasnicka [7, 8]).

The eigenvectors can be computed by inverse iteration either from A, B, or T. In numerical experiments inverse iteration on B turned out to be the most effective. The eigenvectors of A have to be computed from the eigenvectors of B using the orthogonal transformation matrix V.

The new algorithm includes two special cases:

1. If $V = Q$ then $B = T$, and the inverse iteration is performed on the tridiagonal matrix T. This variant coincides with the LAPACK algorithm.
2. If $V = I$ then $B = A$, and the inverse iteration is performed on the original matrix A. This variant is to be preferred if only a few eigenvectors have to be computed and the corresponding eigenvalues are known.

3 The New Level 3 Eigensolver

The blocked LAPACK approach puts the main emphasis on accumulating several elimination steps. b rank-2 updates (each of them a Level 2 BLAS operation) are aggregated to form one rank-$2b$ update and hence one Level 3 BLAS operation. However, this approach does not take into account memory access patterns in the updating process. We tried to introduce blocking in a much stricter sense, namely by using *blocked access patterns.*

Partitioning matrix A into block columns, and further partitioning each block column into quadratic submatrices makes it possible to process the block-columns and their submatrices the same way that the single columns and their elements are processed in the original unblocked method: all *blocks*[2] below the subdiagonal *block* are eliminated, which leaves a *block* tridiagonal matrix, i. e., a *band* matrix (when considered elementwise). At first sight this matrix has bandwidth $4b-1$. Further examination shows that the first elimination sweep can be organized such that the subdiagonal blocks in the block tridiagonal matrix are of upper triangular form. Hence the band width of B can be reduced to $2b+1$.

There are two characteristics which distinguish the newly developed Level 3 algorithm from standard algorithms (see Fig. 1):

[2] Replace *block* by *element* (or *elementwise*) to get the original, unblocked, algorithm.

Two-sweep tridiagonalization: The first sweep reduces the matrix A to a band matrix B. The second sweep reduces B to a tridiagonal matrix T. No fill-in occurs in the first reduction sweep. In the second sweep, however, additional operations are needed to remove fill-in.

Inverse iteration on the band matrix: Calculating the eigenvectors of B avoids the large overhead entailed by the backtransformation of the eigenvectors of T. On the other hand, the overhead caused by inverse iteration on B does not outweigh the benefits of this approach.

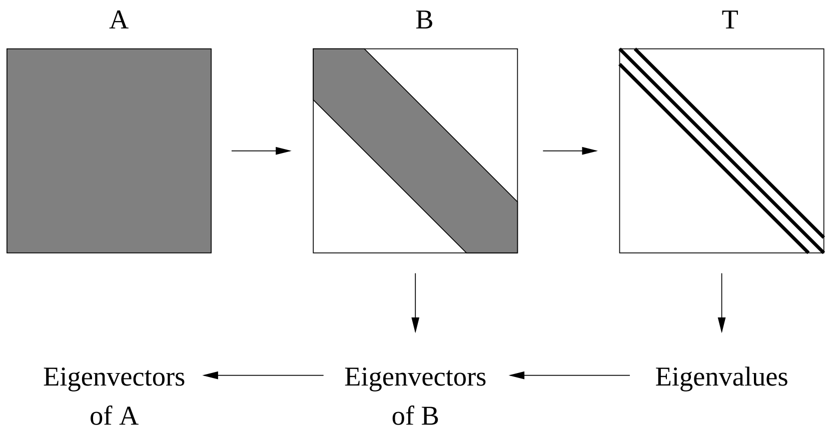

Fig. 1. Basic concept of the new eigensolver

4 Implementation Details

The resulting algorithm has five steps.

1. Reduce the matrix A to the band matrix B.
   (a) Compute the transformation

   $$V_i^T A V_i = (I - u_b u_b^T) \cdots (I - u_1 u_1^T) A (I - u_1 u_1^T) \cdots (I - u_b u_b^T)$$

   for a properly chosen subblock of A. The Householder vectors for the transformation of the columns of this subblock do not depend on each other. This independence enables a new way of blocking in Step 1c.
   (b) Collect the Householder vectors as column vectors in an $n \times b$ matrix Y and compute the $n \times b$ matrix W such that the transformation matrix V_i is represented as $(I - WY^T)$. Matrix Y can be stored in those parts of A which have just been eliminated. Matrix W requires separate storage of order $O(n^2)$; therefore it is overwritten and has to be computed again in the backtransformation step.

(c) Perform a rank-$2b$ update using the update matrix $(I - WY^T)$.
(d) Iterate Steps 1a to 1c over the entire matrix A.

2. Reduce the matrix B to the tridiagonal matrix T. Fill-in increases the total number of operations needed for the tridiagonalization.
3. Compute the desired eigenvalues of the tridiagonal matrix T.
4. Compute the corresponding eigenvectors of B by inverse iteration. The computation of the eigenvectors of B requires a higher amount of operations than the computation of the eigenvectors of T.
5. Transform the eigenvectors of B to the eigenvectors of the input matrix A. The update matrices W have to be computed anew. The transformation matrix V, which occurs in (1), is not computed explicitly.

Variations of the representation of V_i are (see Bischof [3])

- $V_i = (I + WY^T)$. The matrix Y holds a set of Householder vectors whereas the matrix W is computed explicitly. This version is actually used in the current implementation.
- $V_i = (I - GG^T)$ (see Schreiber, Parlett [9]). This version needs higher effort for computing G as well as for coding. However, memory for storing W and the redundant effort to compute W twice is saved.
- $V_i = (I - YUY^T)$ (see Schreiber, Van Loan [10]) with a $b \times b$ triangular matrix U. The storage requirement for U is nearly negligible.

The new algorithm and its variants make it possible to use Level 3 BLAS in all computations involving the original matrix, in contrast to the LAPACK routine `dsyevx`. This routine performs 50 % of the operations needed for the tridiagonalization in Level 2 BLAS.

5 Complexity

The total execution time T of our algorithm consists of five parts:

$T_1 \sim c_1(b)n^3$ for reducing the symmetric matrix A to a band matrix B,
$T_2 \sim c_2(b)n^2$ for reducing the band matrix B to a tridiagonal matrix T,
$T_3 \sim c_3(k)n^2$ for computing k eigenvalues of T,
$T_4 \sim c_4(b)kb^2n^2$ for computing the corresponding k eigenvectors of B, and
$T_5 \sim c_5kn^2$ for transforming the eigenvectors of B into the eigenvectors of A.

The parameters c_1, c_2, and c_4 depend on the semibandwidth b. The parameter c_1 decreases in b, whereas c_2 and c_4 increase in b due to an increasing number of operations to be performed. The parameter c_3 depends on the number of computed eigenvectors k; whereas c_5 is independent of the problem size.

The band reduction step is the only part of the algorithm requiring an $O(n^3)$ effort. Thus, a large semibandwidth b of B seems to be desirable. However, b should be chosen appropriately not only to speed up the band reduction (T_1), but also to make the tridiagonalization (T_2) and the eigenvector computation (T_4) as efficient as possible.

For example, on an SGI Power Challenge the calculation of $k = 200$ eigenvalues and eigenvectors of a symmetric 2000×2000 matrix requires a total execution time $T = 87\,\mathrm{s}$. T_1 is $55\,\mathrm{s}$, i. e., 63 % of the total time. T_4 is $16\,\mathrm{s}$, i. e., 18 % of T. The other steps of the algorithm require an insignificant part of the execution time.

6 Results

A first implementation of our algorithm uses routines from the symmetric band reduction toolbox (SBR; see Bischof et al. [2, 4, 5]), EISPACK routines (Smith et al. [11]), LAPACK routines (Anderson et al. [1]), and some of our own routines.

In numerical experiments we compare the well established LAPACK routine `dsyevx` with the new algorithm. On an SGI Power Challenge (with an R8000 processor running with 90 MHz), speedup factors of up to 4 were observed (see Table 2 and Fig. 5).

Table 2. Execution times (in seconds) on an SGI Power Challenge. $k = n/10$ of the n eigenvalues and eigenvectors were computed

n	k	b	LAPACK `dsyevx`	New Method	Speedup
500	50	6	1.5 s	2.1 s	0.7
1000	100	6	16.6 s	12.6 s	1.3
1500	150	6	74.4 s	39.2 s	1.9
2000	200	10	239.2 s	89.7 s	2.7
3000	300	12	945.5 s	286.4 s	3.3
4000	400	12	2432.4 s	660.5 s	3.7

Fig. 2 shows the normalized computing time $T(n)/n^3$. The significant speedup of the new algorithm when applied to large problems is striking. This speedup has nothing to do with complexity, which is nearly identical for both algorithms (see Fig. 3). The reason for the good performance of the new algorithm is its significantly improved utilization of the computer's potential peak performance (see Fig. 4). The deteriorating efficiency of the LAPACK routine (due to cache effects) causes the $O(n^4)$ behavior of its computation time between $n = 500$ and $n = 2000$ (see Fig. 2).

The new algorithm shows significant speedups compared to existing algorithms for large matrices ($n \geq 1500$) and a small subset of eigenvalues and eigenvectors ($k = n/10$) on all computers at our disposal, including workstations of DEC, HP, IBM, and SGI.

Choosing the block size b. Experiments show that the optimum block size b, which equals the smallest possible band width, increases slightly with larger

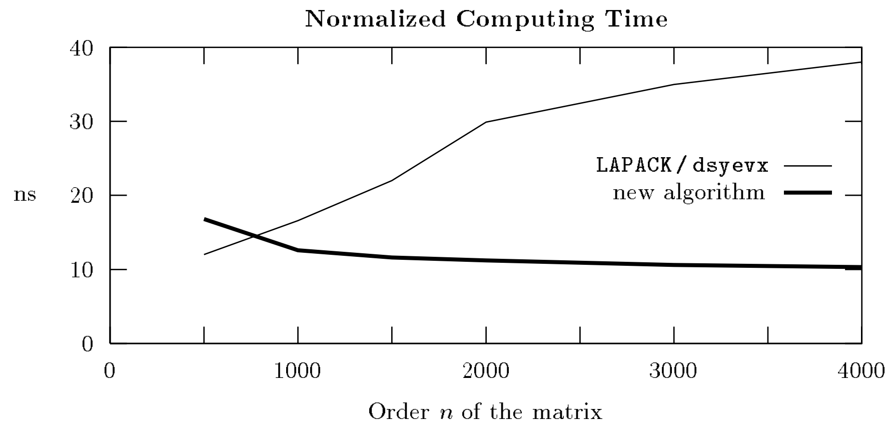

Fig. 2. Normalized computing time $T(n)/n^3$ in nanoseconds

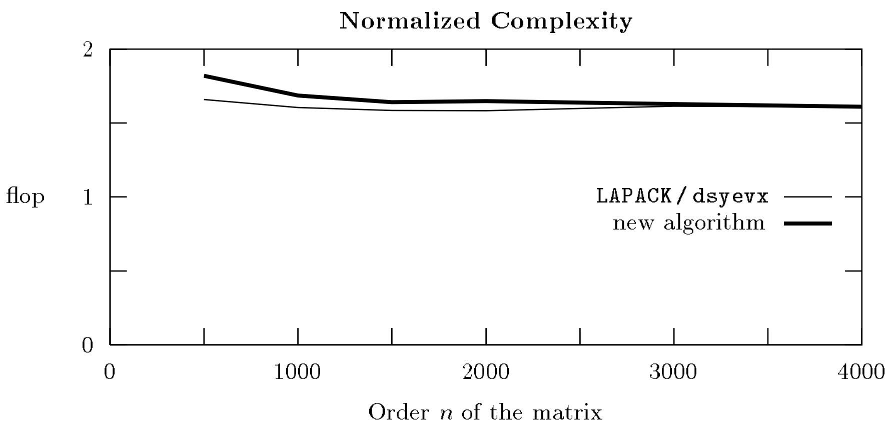

Fig. 3. Normalized number $\mathrm{op}(n)/n^3$ of floating-point operations

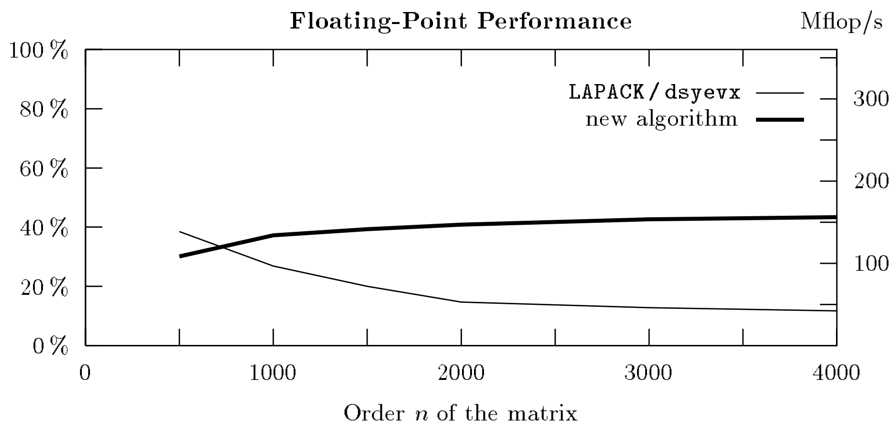

Fig. 4. Floating-point performance (MFlop/s) and efficiency (%)

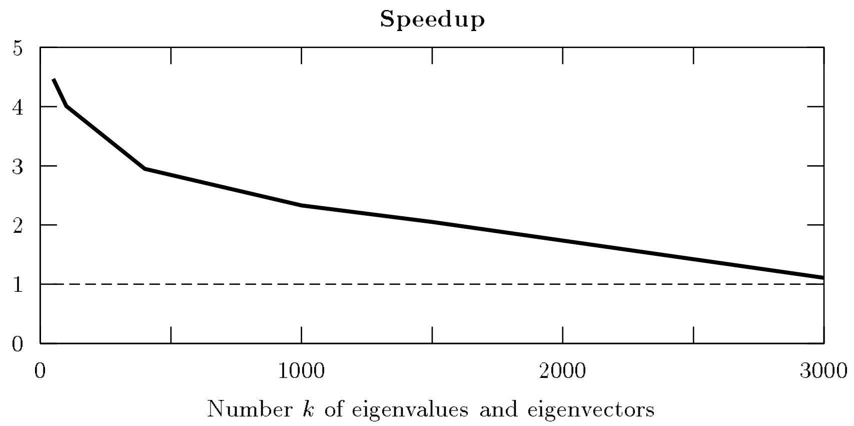

Fig. 5. Speedup of the new algorithm as compared with the LAPACK routine `dsyevx`. Performance improvements are all the better if k is only a small percentage of the $n = 3000$ eigenvalues and eigenvectors

matrix sizes (see Table 2). A matrix of order 500 requires a b of only 4 or 6 for optimum performance, depending on the architecture (and the size of cache lines). Band widths of 12 or 16 are optimum on most architectures when the matrix order is larger than 2000. A hierarchically blocked version which is currently under development allows increasing the block size without increasing the band width and therefore leads to even higher performance.

7 Conclusion

We presented an algorithm for the computation of selected eigenvalues and eigenvectors of symmetric matrices which is significantly faster than existing algorithms. This speedup is achieved by improved blocking in the tridiagonalization process, which significantly improves data locality.

LAPACK algorithms for the symmetric eigenproblem spend up to 80 % of their execution time in Level 2 BLAS, which do not perform well on cache-based and multiprocessor computers. In our algorithm all performance relevant steps make use of Level 3 BLAS.

The price that has to be paid for the improved performance in the tridiagonalization process is that the eigenvectors cannot be computed from the tridiagonal matrix because of prohibitive overhead in the backtransformation. They have to be computed from the intermediate band matrix.

When choosing the block size, a compromise has to be made: Larger block sizes improve the performance of the band reduction. Smaller block sizes have to be used to reduce the band width and, therefore, to speed up the inverse iteration on the band matrix. This property makes the new algorithm particularly attractive (on most architectures) if not all eigenvectors are required.

If the gap between processor speed and memory bandwidth further increases, our algorithm will be highly competitive also for solving problems where all eigenvectors are required.

Future Development. Routines dominated by Level 3 BLAS operations, like the eigensolver presented in this paper, have the potential of speeding up almost linearly on parallel machines. That is why we are currently developing a parallel version. Another promising possibility of development is the use of hierarchical blocking (see Ueberhuber [12]).

References

1. E. Anderson et al., LAPACK *Users' Guide, 2nd ed.*, SIAM Press, Philadelphia, 1995.
2. C. H. Bischof, B. Lang, X. Sun, *Parallel Tridiagonalization through Two-Step Band Reduction*, Proceedings of the Scalable High-Performance Computing Conference, IEEE Press, Washington, D. C., 1994, pp. 23–27.
3. C. H. Bischof, *A Summary of Block Schemes for Reducing a General Matrix to Hessenberg Form*, Technical Report ANL/MCS-TM-175, Argonne National Laboratory, 1993.
4. C. H. Bischof, B. Lang, X. Sun, *A Framework for Symmetric Band Reduction*, ACM Trans. Math. Software, Argonne Preprint ANL/MCS-P586-0496 (1996).
5. C. H. Bischof, B. Lang, X. Sun, *The SBR Toolbox – Software for Successive Band Reduction*, ACM Trans. Math. Software, Argonne Preprint ANL/MCS-P587-0496 (1996).
6. J. J. Dongarra et al., *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM Press, Philadelphia, 1991.
7. W. Gansterer, D. Kvasnicka, *High-Performance Computing in Material Sciences. The Standard Eigenproblem – Concepts*, Technical Report 3/97, AURORA-5, Technical University of Vienna, 1997.
8. W. Gansterer, D. Kvasnicka, *High-Performance Computing in Material Sciences. The Standard Eigenproblem – Experiments*, Technical Report 5/97, AURORA-5, Technical University of Vienna, 1997.
9. R. Schreiber, B. Parlett, *Block Reflectors: Theory and Computation*, SIAM J. Numer. Anal. 25 (1988), pp. 189–205.
10. R. Schreiber, C. Van Loan, *A Storage-Efficient WY Representation for Products of Householder Transformations*, SIAM J. Sci. Stat. Comput. 10–1 (1989), pp. 53–57.
11. B. T. Smith et al., *Matrix Eigensystem Routines –* EISPACK *Guide*, Lecture Notes in Computer Science, Vol. 6, Springer-Verlag, Berlin Heidelberg New York Tokyo, 1976.
12. C. W. Ueberhuber, *Numerical Computation*, Springer-Verlag, Berlin Heidelberg New York Tokyo, 1997.