ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439

ANL/MCS-TM-175

A Summary of Block Schemes for Reducing a General Matrix to Hessenberg Form

by

Christian Bischof

Mathematics and Computer Science Division

Technical Memorandum No. 175

February 1993

This work was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

Contents

Abstract 1

1 Introduction 1

2 Block Representations for Representing Products of Householder Matrices 1

3 Block Schemes for Reducing a General Matrix to Hessenberg Form 2

4 Discussion of the Various Update Schemes 4

References 6

A Summary of Block Schemes for Reducing a General Matrix to Hessenberg Form

by

Christian H. Bischof

Abstract. Various strategies have been proposed for arriving at block algorithms for reducing a general matrix to Hessenberg form by means of orthogonal similarity transformations. This paper reviews and systematically categorizes the various strategies and discusses their computational characteristics.

1 Introduction

Let A be an $n \times n$ symmetric matrix. Our goal is to compute an orthogonal matrix Q, $Q^TQ = I$ such that $Q^TAQ = H$ where H is of upper Hessenberg form. The standard algorithm [10] reduces A one column at a time through Householder transformation at a cost of $O(4/3n^3)$ flops. It mainly employs matrix-vector multiplications and symmetric rank-one updates, which require more memory references than matrix-matrix operations [5,4,7].

On the other hand, so-called block reduction methods allow the formulation of most of the computation in terms of matrix-matrix operations [8], which, at a small increase in the number of floating-point operations, dramatically reduce the number of memory accesses. As a result, the block approach is preferable for machines employing a memory hierarchy, as do (in one form or another) most high-performance computers, from workstations to massively parallel machines. Thus, block algorithms play a prominent role in the LAPACK library of portable linear algebra codes for high-performance architectures [1].

The orthogonal reduction of a matrix to Hessenberg form involves a sequence of $m-2$ Householder reductions

$$P_i = I - 2u_iu_i^T \ , u_i^Tu_i = 2.$$

The key to finding a block formulation is the expression of

$$B = P_p \cdots P_1 A P_1 \cdots P_p$$

in a form that is easy to compute and requires as little access to A as possible. Dongarra, Hammarling, and Sorensen [8] suggested $B = A - UV^T - WU^T$. Dubrulle [9], on the other hand, uses a block formulation of $Q_p = P_1 \cdots P_p$ based on the WY representation [6] to express $B = (I - UY^T)^T(A - ZY^T)$. U, V, W, Y, and Z are all $n \times p$ matrices.

The paper is structured as follows: In § 2 we review the possibilities for expressing a series $P_1 \cdots P_p$ of Householder transformations in block form. In § 3 we then discuss the various schemes that has been proposed and suggest some new ones. The relative merits of the various approaches are discussed in § 4.

2 Block Representations for Representing Products of Householder Matrices

There are several ways for representing a product

$$Q_i = P_1 \cdots P_i$$

of Householder transformations.

WY1: [6]

$$Q_i = I - U_i Y_i^T \tag{1}$$

WY2: [6]

$$Q_i = I - W_i U_i^T \tag{2}$$

compactWY: [12]

$$Q_i = I - U_i S_i U_i^T \tag{3}$$

In all those schemes, U_i is the matrix of Householder vectors, and extra storage is required for the other matrices. In the above formulas, U_i, Y_i, and W_i are $m \times i$ matrices. U_i and Y_i are lower trapezoidal and S_i is an upper triangular $i \times i$ matrix. Since typically $n \gg i$, the "compact" scheme requires less work space, at the expense of slightly more work in applying Q_i. These matrices are accumulated as follows ($k = 1, \ldots, i-1$). For all schemes, we have

$$U_{k+1} = [U_k \, , u_{k+1}]; \tag{4}$$

that is, U just collects the Householder vectors used in the transformation. The other matrices get updated as follows:

$$\begin{aligned} Y_{k+1} &= [P_k Y_k \, , u_{k+1}] && (5)\\ W_{k+1} &= [W_k \, , Q_k u_{k+1}] && (6)\\ S_{k+1} &= \begin{pmatrix} S_k & -S_k U_k^T u_{k+1} \\ 0 & 1 \end{pmatrix} && (7) \end{aligned}$$

With these block formulations, the one-sided application of Q_i to a matrix now involves two or three matrix-matrix multiplications instead of a series of i rank-one updates and matrix-vector multiplications. The resulting decrease in memory traffic usually more than makes up for the additional $O(i^2 n)$ floating-point operations. Examples of the use of block orthogonal transformations in the computation of the QR factorization $A = QR$, where Q is orthogonal and R upper triangular, can be found in [6,5,3].

We also mention that Walker [13] and Puglisi [11] suggested a variant of the compact WY representation which represents

$$Q_i = I - U_i (T_i)^{-1} U_i^T. \tag{8}$$

Computationally, the formulation (8) behaves very much like (3) and can be easily substituted in formulae based on (3). Hence, for the sake of brevity, we will not consider it further.

3 Block Schemes for Reducing a General Matrix to Hessenberg Form

The two main steps in the block algorithm are the generation of a block transformation and its application to the remaining part of the matrix. As a shorthand, let HH(x) be the Householder vector that reduces x to a multiple of e_1. Let

$$A_i = P_i \cdots P_1 A P_1 \cdots P_i$$

be the matrix obtained after the ith symmetric Householder update. Further, let $A^{(i:n,j:n)}$ be the $(n-i+1) \times (n-j+1)$ submatrix beginning at location (i,j) of A with analogous notation for vectors.

Having partitioned A into M block columns $\tilde{A}_j$ of width p, $m = Mp, j = 1, \ldots, M$, the "blueprint" for a block algorithm is shown in Figure 1. In the "rightlooking" variant we immediately apply Q_j to all remaining block columns, whereas in the "leftlooking" algorithm we apply

$\tilde{Q} \leftarrow I$;
for $j = 1$ **to** M **do**
Find an orthogonal matrix $\tilde{Q}_p$ such that
$(\tilde{Q}_p^T A(p(j-1)+1 : pM, p(j-1)+1 : pM)\tilde{Q}_p)(p(j-1)+1 : pM, p(j-1)+1 : pj)$
is upper Hessenberg.
Either **rightlooking algorithm:**
Update block columns $j+1, \ldots, M$ with $\tilde{Q}_p$.
or **leftlooking algorithm:**
Update block column $j+1$ with $\tilde{Q}_1, \ldots, \tilde{Q}_j$.
end for

Figure 1:

all previous block orthogonal transformations to the next block column only. In particular, the leftlooking algorithm requires storage of the block transformations $Q_1, \ldots, Q_j$. Hence the rightlooking algorithm is usually preferred (see [2] for a thorough discussion of the leftlooking vs. rightlooking issue). In order to develop an efficient block algorithm, we must be to find $\tilde{Q}_j$ and update $\tilde{A}_j$ in a fashion that requires as little as possible access to block columns $j+1, \ldots, M$. We consider the following alternatives:

implicit_Q1: ([8])

$$A_p = A - U_p V_p^T - W_p U_p^T \tag{9}$$

implicit_Q2: ([9])

$$A_p = (I - U_p Y_p^T)^T A (I - U_p Y_p^T) = A - U_p \tilde{V}_p^T - \tilde{W}_p U_p^T \tag{10}$$

Q_WY: (see [9])

$$A_p = (I - U_p S_p U_p^T)^T A (I - U_p S_p U_p^T) = (I - U_p Y_p^T)^T (A - Z_p Y_p^T) \tag{11}$$

Q_compactWY:

$$A_p = (I - U_p S_p U_p^T)^T (A - Z_p S_p U_p^T) \tag{12}$$

These matrices are accumulated as shown below for $i = 1, \ldots, p-1$. For all cases we have

$$U_{i+1} = [U_i \, , u_{i+1}] \, .$$

implicit_Q1:

$$\begin{gathered} u_{i+1} = \mathrm{HH}(\, a_i^{(i:m)} - (U_i V_i^T + W_i U_i^T) e_i^{(i:m)} \,) \\ v = (A^T - V_i U_i^T - U_i W_i^T) u_{i+1} \, , \; x = (A - U_i V_i^T - W_i U_i^T) u_{i+1} \\ V_{i+1} = [V_i \, , v - \frac{x^T u_{i+1}}{2} u_{i+1}] \, , \; W_{i+1} = [W_i \, , x - \frac{v^T u_{i+1}}{2} u_{i+1}] \end{gathered} \tag{13}$$

implicit_Q2:

$$\begin{gathered} u_{i+1} = \mathrm{HH}(\, a_i^{(i:m)} - U_i \tilde{V}_i^T - \tilde{W}_i U_i^T) e_i^{(i:m)} \,) \\ x = (A^T - U_i \tilde{V}_i{}^T - \tilde{W}_i U_i^T - u_{i+1} u_{i+1}^T (A^T - U_i \tilde{V}_i^T - \tilde{W}_i U_i^T)) u_{i+1} \\ \tilde{V}_{i+1} = [\tilde{V}_i \, , x] \, , \; \tilde{W}_{i+1} = [\tilde{W}_i \, , A u_{i+1}] \end{gathered} \tag{14}$$

Q_WY;

$$u_{i+1} = \mathrm{HH}(\,(I - U_p Y_p^T)^T (a_i^{(i:m)} - Z_i Y_i^T e_i^{(i:m)})\,)$$
$$Z_{i+1} = [Z_i\,,\, Au_{i+1}] \tag{15}$$
$$Y_{i+1} \text{ is updated as in (5).}$$

Q_compactWY;

$$u_{i+1} = \mathrm{HH}(\,(I - U_i S_i U_i^T)^T (a_i^{(i:m)} - Z_i S_i U_i^T e_i^{(i:m)})\,)$$
$$Z_{i+1} = [Z_i\,,\, Au_{i+1}] \tag{16}$$
$$S_{i+1} \text{ is updated as in (7).}$$

While the latter two schemes are reformulations of similarity transformations of Q in block form, the first two schemes do not explicitly involve a block orthogonal transformation. Thus, for the 'Q_WY' and 'Q_compactWY' schemes, the orthogonal transformations $\tilde{Q}_j$ are readily available in a convenient block form, whereas there is no block form of $\tilde{Q}_j$ readily accessible in the "implicit" schemes. Hence, the "implicit" schemes allow only for a rightlooking algorithm, whereas the other schemes allow for both a leftlooking and rightlooking formulation.

4 Discussion of the Various Update Schemes

To assess the work required for a block update, let us assume that A is $m \times m$ and that we choose a block size of p. Then at step $i = 1, \ldots, p$ the length of u_i is $m - i + 1$ and the work required to perform an update step can be expressed in the following units:

T_{mv}: Work required to multiply an $m \times (i-1)$ matrix with an $(i-1)$ vector or an $(i-1) \times m$ matrix by an m-vector. This requires $2m(i-1)$ flops.

T_{sv}: Work required to multiply an $(i-1) \times (i-1)$ upper triangular matrix by an $(i-1)$-vector. This requires $(i-1)^2$ flops.

T_{Au}: Work required to multiply an $(m-i+1) \times (m-i+1)$ matrix by an $(m-i+1)$-vector. This requires $2(m-i)^2$ flops.

After a block update of width p is computed, this update will have to be applied to an $m \times k$ submatrix of A, where k is determined primarily by our choice of a left- or rightlooking algorithm. If we choose a leftlooking algorithm, then $k = p$ (assuming fixed block sizes), whereas for the rightlooking algorithm $k = m - p$. This work can be expressed in terms of

T_{rk_p}: Work required to apply a rank-p update to an $m \times k$ matrix. This requires $2mkp$ flops.

T_{mm1}: Work required to multiply a $p \times m$ by an $m \times k$ matrix. This requires $2mkp$ flops.

T_{mm2}: Work required to multiply a $p \times p$ upper triangular matrix by a $p \times k$ matrix. This requires kp^2 flops.

Another point to consider is whether the block formulation of A also produces a block formulation for Q. The latter may be important if Q has to be applied to the eigenvectors of the Hessenberg matrix in a back-transform step. Using the above criteria, we summarize the differences between the different blocking schemes in Table 1. The first row (labeled "Gen. Step i") shows the work to be performed at the ith step of generating the block transformation, the second row (labeled "Application") shows the work required in applying the block update, and the last row (labeled "Block Q?") indicates whether this formulation also results in a block formulation for Q.

Table 1: Work required at Step i

	implicit_Q1	implicit_Q2	Q_WY	Q_compactWY
Gen. Step i	$10T_{mv} + 2T_{Au}$	$6T_{mv} + 2T_{Au}$	$4T_{mv} + T_{Au}$	$4T_{mv} + 3T_{sv} + T_{Au}$
Application	$2T_{rk_p}$	$2T_{rk_p}$	$2T_{rk_p} + T_{mm1}$	$2T_{rk_p} + T_{mm1} + 2T_{mm2}$
Block Q ?	no	no	yes	yes

Table 2: Storage Requirements

	implicit_Q1	implicit_Q2	Q_WY	Q_compactWY
Representing blocks	$2mp$	$2mp$	$2mp$	$mp + \frac{p^2}{2}$
Extra storage discarding blocks	none	none	none	kp
Extra storage preserving blocks	none	none	kp	kp

We compare in Table 2 the amount of storage required for the different schemes in a rightlooking algorithm. Assuming that U is stored in the lower triangular part of A, we consider the amount of storage required to represent the blocking factors, and the *extra* amount of storage required to apply an $m \times p$ block update to an $m \times k$ matrix with and without discarding of block orthogonal transformations.

Dubrulle [9] compared a rightlooking algorithm using formulations "implicit_Q2" and "Q_WY" on the IBM 3090/VF using straight Fortran. In these experiments he found the "Q_WY" formulation to be superior to the "implicit_Q2" formulation.

These numbers suggest that the "Q_WY" scheme is superior to the "implicit_Q2" scheme even if we do not need the block factors defining Q later on. The reason seems to be that in the "implicit_Q2" scheme we have to compute Au and $A^T u$ at every step of accumulating a block transform, whereas "Q_WY" gets by with only computing Au and with two less matrix-vector multiplications. On the other hand "Q_WY" requires an extra matrix-matrix product (the work needed is denoted by T_{mm1} in Table 1) when the block transformation is applied. Thus it seems as if "Q_WY" profits from the usual BLAS 3 to BLAS 2 tradeoff in that matrix-matrix operations execute more efficiently than a series of matrix-vector operations. Furthermore the "Q_WY" scheme requires less data traffic since we touch A only once at every update step, whereas "implicit_Q2" touches A twice.*

The scheme "implicit_Q2" scheme is similar to "implicit_Q1" which was implemented in the second test release of LAPACK, and it is reasonable to assume that it would perform similarly. The "Q_compactWY" scheme is similar to scheme "Q_WY" but requires less storage, in particular if a left-looking scheme is used (i.e., $k \ll m$ in Table 2) or when block transformations are stored for successive back transformations. Since the "implicit" schemes do not require less storage, do not give us any block formulation for Q, and do not seem to perform any better, it seems advantageous to use one of the WY-based schemes. In particular, the one based on the compact WY representation seems to be appropriate, since it allows economical storage of the block transforms for subsequent back transformations of the eigenvectors of the Hessenberg matrix. Hence, the "Q_compactWY" scheme was chosen for the final release of LAPACK [1].

*We assume here that Au and $A^T u$ are computed separately using BLAS 2 calls. We can get by with touching A only once, but then we cannot use a BLAS 2 call.

Acknowledgments

I thank Jeremy Du Croz and Sven Hammarling for the many stimulating discussions that eventually prompted me to methodologically evaluate the various possibilities.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK User's Guide*. SIAM, Philadelphia, Penna., 1992.

[2] E. Anderson and J. Dongarra. Evaluating block algorithm variants in LAPACK. Technical Report CS–90–103, Computer Science Department, The University of Tennessee, April 1990.

[3] C. H. Bischof. A block QR factorization algorithm using restricted pivoting. In *Proceedings SUPERCOMPUTING '89*, pages 248–256, Baltimore, Md., 1989. ACM Press.

[4] C. H. Bischof. *Fundamental linear algebra computations on high-performance computers*, in volume 250 of *Informatik Fachberichte*, pages 167–182. Springer-Verlag, Berlin, 1990.

[5] C. H. Bischof and J. J. Dongarra. A project for developing a linear algebra library for high-performance computers. In Graham Carey, editor, *Parallel and Vector Supercomputing: Methods and Algorithms*, pages 45–56. John Wiley & Sons, Somerset, N.J., 1989.

[6] C. H. Bischof and C. F. Van Loan. The WY representation for products of Householder matrices. *SIAM Journal on Scientific and Statistical Computing*, 8:s2–s13, 1987.

[7] J. Dongarra and S. Hammarling. *Evolution of Numerical Software for Dense Linear Algebra*, pages 297–327. Oxford University Press, Oxford, U.K., 1989.

[8] J. J. Dongarra, S. J. Hammarling, and D. C. Sorensen. Block reduction of matrices to condensed form for eigenvalue computations. Technical Report MCS–TM–99, Mathematics and Computer Science Division, Argonne National Laboratory, September 1987.

[9] A. A. Dubrulle. On block Householder algorithms for the reduction of a marix to Hessenberg form. In Joanne L. Martin and Stephen F. Lundstrom, editors, *Supercomputing '88: Volume II, Science and Applications*, Washington, DC, 1989. IEEE Computer Society Press.

[10] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1983.

[11] C. Puglisi. Modification of the Householder method based on the compact WY representation. CERFACS Report TR/PA/90/29, 1990.

[12] R. Schreiber and C. Van Loan. A storage efficient WY representation for products of Householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 10(1):53–57, 1989.

[13] H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 9(1):152–163, 1988.