A. A. Dubrulle

Solution of the Complete Symmetric Eigenproblem in a Virtual Memory Environment

Abstract: Algorithms for the complete symmetric eigenproblem are studied from the viewpoint of performance in a virtual memory environment. The preferred algorithm is based on tridiagonalization and the implicit QR algorithms. Implementation factors that affect execution time are discussed.

Introduction

Virtual memory and paging devices have considerably simplified the development of programs operating on large data arrays. Storage management problems that were left to the programmer's ingenuity are now automatically dealt with by the system.

However, some precautions must be taken in order to implement algorithms that are efficient in the context of virtual memory with paging. This question is of particular importance in the solution of matrix problems [1]. When designing a program involving matrix operations, the choice of the algorithms is guided not only by numerical considerations, such as stability, accuracy, and count of arithmetic operations, but also by other factors more intimately related to the virtual memory. A "good" algorithm should result in a program that minimizes the number and the extent of the virtual memory sweeps required in referring to the elements of the matrices involved.

In this paper, a few methods for the solution of the complete symmetric eigenproblem are examined in a virtual memory context, and the implementation of the preferred algorithm is studied. Additional considerations are also taken into account:

- that performance of the algorithm is also optimal in a real memory environment.
- that extra programming effort required by virtual memory considerations is minimal, and
- that the overall legibility of the resulting program is good in the sense that the code closely reflects the numerical algorithm.

Matrices are assumed to be stored row-wise as is the case in PL/I and BASIC. However, techniques similar to those described here apply as well to FORTRAN [1] (matrices stored column-wise). Experiments were run on an IBM System/3, Model 6 using the BASIC software paging facilities.

Choice of method

Two methods are considered for their numerical properties of stability, accuracy, and reliability [2]: Jacobi's method with threshold, and a combination of Householder's tridiagonalization with the implicit QR algorithm. These methods are briefly described below (detailed information is provided in the references).

- *Jacobi's method with threshold*

The matrix is diagonalized by means of similarity transformations [3, 4].

Given a positive threshold, each off-diagonal element of the matrix that is larger than the threshold in absolute value is annihilated in turn by a similarity transformation based on a plane rotation. For an off-diagonal subscripted element (i, j), this implies the transformation of rows and columns i and j of the matrix. Once all off-diagonal

elements are smaller than the threshold, this threshold is decreased and the above procedure is repeated. The whole process is iterated until the threshold is negligible in terms of machine precision.

The eigenvectors are obtained by forming the product of the plane rotations involved in the diagonalization. The ultimate convergence of the process is quadratic.

For a matrix of order n, the minimum storage requirements are:

One one-dimensional array of $n(n+1)/2$ locations for the symmetric matrix, and

One $(n \times n)$ array, where the matrix of eigenvectors is constructed.

This widely known method is quite attractive for its elegance and its compactness. Yet, one can see the problems it raises in a virtual memory environment: there are extensive sweeps of the memory for each plane rotation, that is, the transformation of two rows and two columns of the matrix. When forming the matrix of the eigenvectors, extra sweeps are required from the array of the matrix to the array of the eigenvectors and back, and also within the array of the eigenvectors (transformation of two rows for each plane rotation). These sweeps are at the kernel of the iterative process.

• *Householder's tridiagonalization and implicit QR*

The original matrix is first reduced to a symmetric tridiagonal matrix by orthogonal similarity transformations based on Householder's matrices [5, 2]. The resulting tridiagonal matrix is then diagonalized by using the QR (or QL) algorithm with implicit shift [6, 7]. The ultimate convergence of the QR algorithm is cubic.

The matrix of eigenvectors is built by forming the product of the Householder matrices of the tridiagonalization stage and the plane rotations of the QR algorithm.

In a real memory environment, this method is the most efficient for the treatment of the complete eigenproblem.

For a matrix of order n, the storage requirements are:

One two-dimensional array of n^2 locations for the matrix of eigenvectors, which initially contains the symmetric matrix (see Ref. 5, procedure tred2), and

Two one-dimensional arrays of n locations for the diagonal and subdiagonal elements of the symmetric tridiagonal matrix.

In addition to being more economical of storage, this method presents the substantial advantage that only one large array is involved in the iterative process (QR) for the computation of the eigenvectors. This means that the number and the extent of the virtual memory sweeps is more limited than in the Jacobi algorithm. One more factor in favor of the QR algorithm is better convergence properties.

• *Remarks on inverse iteration for the eigenvectors*

A third possibility was considered because no operation on large arrays was imbedded in an iterative process: a combination of tridiagonalization and QR for the eigenvalues, and Varah's inverse iteration for the eigenvectors [8].

Although this approach leads to a program sometimes as efficient as one based on the QR algorithm, it presents serious drawbacks: The resulting program is about twice as voluminous as the program based on tridiagonalization and QR, and the storage requirements are higher than for the Jacobi method. Furthermore, the orthogonality of the eigenvectors is not guaranteed, the eigenvectors corresponding to clustered eigenvalues may be poorly determined.

The preferred algorithm is thus based on tridiagonalization and the QR algorithm for the computation of the eigenvalues and eigenvectors.

Implementation of the tridiagonalization–QR method

• *Tridiagonalization*

Let $\mathbf{A}$ be a symmetric matrix. Its reduction to tridiagonal form can be expressed by the following:

$$\mathbf{A}^{(0)} = \mathbf{A},$$

$$\mathbf{A}^{(k+1)} = \mathbf{P}^{(k)}\mathbf{A}^{(k)}\mathbf{P}^{(k)}, \qquad k = 0, \cdots, n-3, \tag{1}$$

where $\mathbf{A}^{(k)}$ is a symmetric matrix that is tridiagonal in its last k rows and where $\mathbf{P}^{(k)}$ is the Householder matrix that reduces the $(n-k)$th row of $\mathbf{A}^{(k)}$ to tridiagonal form. $\mathbf{A}^{(n-2)}$ is the resulting tridiagonal matrix similar to $\mathbf{A}$.

$\mathbf{P}^{(k)}$ is orthogonal and of the form

$$\mathbf{P}^{(k)} = \mathbf{I} + \alpha^{(k)}\mathbf{v}^{(k)}\mathbf{v}^{(k)T},$$

where $\alpha^{(k)}$ is a scalar and $\mathbf{v}^{(k)}$ a vector chosen to annihilate the pertinent elements of row $(n-k)$ of $\mathbf{A}^{(k)}$

Equation (1) is equivalent to

$$\mathbf{A}^{(k+1)} = \mathbf{A}^{(k)} + \mathbf{w}\,\mathbf{h}^T + \mathbf{h}\,\mathbf{w}^T, \tag{2}$$

where

$$\mathbf{w} = \alpha^{(k)}\mathbf{v}^{(k)} \text{ and}$$

$$\mathbf{h} = \mathbf{A}^{(k)}\mathbf{v}^{(k)} + [(\mathbf{v}^{(k)T}\mathbf{A}^{(k)}\mathbf{v}^{(k)})/2]\mathbf{w}. \tag{3}$$

(Superscripts applicable to $\mathbf{w}$ and $\mathbf{h}$ have been omitted for clarity.)

Since the successive $\mathbf{A}^{(k)}$'s are symmetric, only their lower triangular part is formed, including the diagonal.

Equation (2), rewritten as

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} + w_i h_j + w_j h_i,$$

shows that, once vectors $\mathbf{w}$ and $\mathbf{h}$ are computed, the elements of the array containing $\mathbf{A}^{(k)}$ can be modified to produce $\mathbf{A}^{(k+1)}$ in the order that best suits the structure of the virtual memory, that is, row-wise. The main computational part of Eq. (3) is the calculation of $\mathbf{A}^{(k)}\mathbf{v}^{(k)}$.

$\mathbf{v}^{(k)}$ is defined by

$$v_i^{(k)} = a_{n-k,i}^{(k)}, \qquad 1 \le i < n-k-1,$$

$$v_i^{(k)} = a_{n-k,i}^{(k)} - p^{(k)}, \qquad i = n-k-1,$$

$$v_i^{(k)} = 0 \text{ otherwise,}$$

with

$$p^{(k)} = -\operatorname{sign}\,(a_{n-k,n-k-1}^{(k)}) \left[\sum_{i=1}^{n-k-1} (a_{n-k,i}^{(k)})^2\right]^{1/2}.$$

Since the last $(k+1)$ components of $\mathbf{v}^{(k)}$ are zero, the multiplication $\mathbf{A}^{(k)}\mathbf{v}^{(k)}$ involves the leading principal submatrix of $\mathbf{A}^{(k)}$ of order $(n-k-1)$ only. To efficiently perform the multiplication, one would like to address each pertinent element of the lower triangular part of $\mathbf{A}^{(k)}$ once and only once, the matrix being swept row-wise.

This can be achieved as shown in the sequence of BASIC statements below, where N is the order of the matrix, A, the array of $\mathbf{A}^{(k)}$, V, the array of $\mathbf{v}^{(k)}$, X, the array of $\mathbf{A}^{(k)}\mathbf{v}^{(k)}$, and M represents $(n-k-1)$.

```
 1  FOR I = 1 TO M
 2  S = 0
 3  T = V(I)
 4  FOR J = 1 TO I - 1
 5  U = A(I, J)
 6  X(J) = X(J) + U*T
 7  S = S + U*V(J)
 8  NEXT J
 9  X(I) = S + A(I, I)*T
10  NEXT I
```

In the implementation of the tridiagonalization process, it is essential for good performance that a separate one-dimensional array be allocated for vector $\mathbf{v}^{(k)}$, although its elements can be retrieved from the $(n-k)$th row of the array containing $\mathbf{A}^{(k)}$. The nonzero parts of $\mathbf{w}$ and $\mathbf{h}$ are stored in one-dimensional arrays, where the diagonal and subdiagonal elements of the resulting tridiagonal matrix are stored as they are obtained. Once the matrix has been reduced to tridiagonal form, the product of the Householder transformations is formed as a first step in the construction of the matrix of eigenvectors. This product is computed in the array used for the $\mathbf{A}^{(k)}$'s. In order to allow this product to be expressed in terms of row operations, the product $\mathbf{P}^{(0)}\mathbf{P}^{(1)} \cdots \mathbf{P}^{(n-3)}$ (premultiplying matrices) is formed, which eventually leads to the matrix of left eigenvectors (transpose of the matrix of eigenvectors).

When the off-tridiagonal part of the $(n-k-1)$ row of $\mathbf{A}^{(k)}$ is negligible in terms of machine precision, it is advantageous to skip a transformation and to write $\mathbf{A}^{(k+1)} \equiv \mathbf{A}^{(k)}$. The following simple test can be used for this purpose. The transformation is skipped when

$$\text{fl}\left\{\text{fl}\left[\sum_{i=1}^{n-k-2} (a_{n-k,i}^{(k)})^2\right] + \text{fl}\,\{(a_{n-k,n-k-1}^{(k)})^2\}\right\} \equiv \text{fl}\,\{(a_{n-k,n-k-1}^{(k)})^2\},$$

where fl { } denotes values obtained by machine floating-point computation.

• *QR algorithm for a symmetric tridiagonal matrix*

The following is the QR form of the implicit QL algorithm of [6, 7]. The shift at the kth iteration is $\sigma^{(k)}$, $\{e_i^{(k)}; i = 2, \cdots, n\}$ are the subdiagonal elements of the tridiagonal matrix, and $\{d_i^{(k)}; i = 1, \cdots, n\}$ are the diagonal elements. At the kth iteration, the shift is chosen to be the eigenvalue of the bottom 2×2 principal submatrix that is closer to $d_n^{(k)}$. When $e_n^{(k)}$ can be considered as negligible, $d_n^{(k)}$ is taken as an eigenvalue and the order n of the matrix is reduced by one (by neglecting the last row and column). This process is repeated on the successive reduced matrices until the whole matrix is diagonalized. The formulas reflecting one iteration are

$$c_1 = s_1 = 1,\ q_1 = d_1^{(k)} - \sigma^{(k)},\ u_1 = 0.$$

$$h_i = c_{i-1} e_i^{(k)},$$

$$p_i = s_{i-1} e_i^{(k)},$$

$$e_{i=1}^{(k+1)} = (p_i^2 + q_{i-1}^2)^{1/2},$$

$$s_i = p_i / e_{i-1}^{(k+1)},$$

$$c_i = q_i / e_{i-1}^{(k+1)},$$

$$g_i = d_{i-1}^{(k)} - u_{i-1},$$

$$t_i = (d_i^{(k)} - g_i) s_i + 2 c_i h_i,$$

$$u_i = s_i t_i,$$

$$d_{i-1}^{(k+1)} = g_i + u_i,$$

$$q_i = c_i t_i - h_i,$$

$$d_n^{(k+1)} = d_n^{(k)} - u_n,\ e_n^{(k+1)} = q_n;$$

where $i = 2, \cdots, n$.

This QR transformation consists of performing on the tridiagonal matrix $(n-1)$ similarity transformations based on plane rotations. The plane rotations apply to pairs of consecutive rows and columns of the matrix, taken in their natural order: (1, 2), (2, 3), · · ·. The ith plane rotation is defined by the parameters c_{i+1} and s_{i+1}.

The matrix of left eigenvectors is obtained by premultiplying the matrix of accumulated transformations

generated in the tridiagonalization step by plane rotations such as

$$
\begin{array}{c} \\ (i-1) \\ (i) \\ \\ \\ \\ \\ \\ \\ \\ \end{array}
\begin{bmatrix}
1 & & & & & & \\
 & \ddots & & & & & \\
 & & 1 & & & & \\
 & & c_i & s_i & & & \\
 & & -s_i & c_i & & & \\
 & & & & 1 & & \\
 & & & & & \ddots & \\
 & & & & & & 1
\end{bmatrix}
\quad i = 2, \cdots, n .
$$

This is a favorable situation from a virtual memory viewpoint: The array of eigenvectors is modified row wise and the virtual memory sweeps are quite limited in number since the rows transformed by each plane rotation are adjacent.

The formulas describing the QR iteration show that practically no performance improvement is to be expected from a special formulation of the algorithm for virtual memory.

• *Remarks on implementation*

It should be noted that no algorithmic modification of the method was necessary for an adaptation to a virtual memory environment. The critical points are:

- The use of the same array for the original matrix and the matrix of eigenvectors,
- All operations involved in the tridiagonalization being performed row-wise,
- The construction of the matrix of left eigenvectors, and
- A special implementation of the matrix-vector product.

These implementation details do not affect the performance of the method in a real memory environment.

Computation of the eigenvalues only

For the eigenvalues only, the advantage of the tridiagonalization-QR is substantially increased, since no operation on a large array is imbedded in the iterative process. Furthermore, there is no need to form the product of the transformations involved in the tridiagonalization stage.

Since no $(n \times n)$ array is needed for the eigenvectors, program efficiency is increased if the symmetric matrix is represented by the element of its lower triangular part packed row-wise in a one-dimensional array of $n(n+1)/2$ locations.

The matrix-vector multiplication then becomes

```
 1  K = 0
 2  FOR I = 1 TO M
 3  S = 0
 4  T = V(I)
 5  FOR J = 1 TO I - 1
 6  K = K + 1
 7  U = A(K)
 8  X(J) = X(J) + U*T
 9  S = S + U*V(J)
10  NEXT J
11  K = K + 1
12  X(I) = S + A(K)*T
13  NEXT I
```

Note on FORTRAN implementation

In this case, matrices are stored column-wise. The approach described for the tridiagonalization is still valid if all the operations formulated in terms of the rows of the lower part of the $\mathbf{A}^{(k)}$'s are now thought of in terms of corresponding columns of the upper part of these matrices.

The matrix of *right* eigenvectors is constructed by forming the product $\mathbf{P}^{(n-3)}\,\mathbf{P}^{(n-2)} \cdots \mathbf{P}^{(0)}$ and then by postmultiplying this product by the transposes of the above QR plane rotations. All these operations are performed column-wise.

Experimental results

In general, the time savings obtained by using the combination of tridiagonalization-QR rather than Jacobi's method are very substantial. These savings are highly dependent on the size of the arrays, the size of the virtual memory page and other characteristics of the system used.

Test runs were performed on an IBM System/3 Model 6 with a main memory of 16,000 bytes. Each page of the virtual memory has a capacity of 51 floating-point short precision words.

With arrays allowing for a maximum matrix order of 25, the combination tridiagonalization-QR was about five times as fast as Jacobi's method for random matrices of order 20.

References

1. C. B. Moler, "Matrix Computations with Fortran and Paging," *Communications ACM* **15,** No. 4 (April, 1972).
2. J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.
3. J. Greenstadt, "The Determination of the Characteristic Roots of a Matrix by the Jacobi Method," *Mathematical Methods for Digital Computers*, Vol. I, Editors Ralston and Wilf, John Wiley & Sons, New York, 1960.
4. H. Rutishauser, "The Jacobi Method for Real Symmetric Matrices," *Handbook for Automatic Computation, Vol. II, Linear Algebra*, Editors Wilkinson and Reinsch, Springer, Berlin, 1971.

5. R. S. Martin, C. Reinsch, and J. H. Wilkinson, "Householder Tridiagonalization of a Symmetric Matrix," *Handbook for Automatic Computation, Vol. II, Linear Algebra*, Editors Wilkinson and Reinsch, Springer, Berlin, 1971.
6. A. Dubrulle, "A Short Note on the Implicit QL Algorithm for Symmetric Tridiagonal Matrices," *Numerische Mathematik*, **15**, p. 450 (1970).
7. A. Dubrulle, R. S. Martin, and J. H. Wilkinson, "The Implicit QL Algorithm," *Handbook for Automatic Computation, Vol. II, Linear Algebra*, Editors Wilkinson and Reinsch, Springer, Berlin, 1971.
8. G. Peters, and J. H. Wilkinson, "The Calculation of Specified Eigenvectors by Inverse Iteration," *Handbook for Automatic Computation, Vol. II, Linear Algebra*, Editors Wilkinson and Reinsch, Springer, Berlin, 1971.

Received June 2, 1972

The author is located at the IBM Data Processing Division Development Center, Palo Alto, California. 94301