EDUCATIONAL AND PSYCHOLOGICAL MEASUREMENT
1984, 44

AN APL PROCEDURE FOR COMPUTING THE EIGENVECTORS AND EIGENVALUES OF A REAL SYMMETRIC MATRIX[1]

JAMES L. ADAMS AND J. ARTHUR WOODWARD
University of California, Los Angeles

An APL procedure is presented for rapid, efficient computation of the eigenvalues and eigenvectors of a real symmetric matrix. Since APL has an interpreter, but not a compiler, many iterative procedures are very slow when executed in APL. In this paper, the writers develop a highly compact way of coding an iterative procedure in APL and then apply it to Kaiser's JK method of computing eigenvectors. The resulting APL program is presented in the Appendix.

Many methods of multivariate statistical analysis reduce to calculation of eigenvectors and eigenvalues of a real symmetric matrix. Principal components analysis, multiple discriminant analysis, multivariate analysis of variance, and certain approaches to factor analysis involve computation of eigenvectors and eigenvalues of real symmetric matrices. Although in many respects APL (A Programming Language) is an ideal computer language for handling complexities of these types of multivariate analyses, the calculation of eigenvectors and eigenvalues poses practical problems. Because APL is interpretive, it does not appear to lend itself to efficient execution of iterative algorithms (such as the eigen problem). Thus, the potential of APL as a tool for teaching multivariate analysis in applied settings (Wilkinson and Huesmann, 1973) and as a medium for conducting applied multivariate data analysis (Margolin, 1976) must remain to some extent hypothetical unless principles of APL programming can be devised for iterative algorithms.

[1]This research was supported in part by a UCLA Biomedical Research Support Grant and by a grant (DA01070) from the U.S. Public Health Service.

The purpose of the present paper was to develop an efficient APL procedure for iteratively calculating eigenvectors and eigenvalues of a real symmetric matrix by using a highly compact, parallel-processing code. Although this approach to APL coding has been developed in the context of the eigenproblem, it may be applicable to other iterative numerical methods used in multivariate analysis as well.

The APL Approach

The advantage of APL lies in its power and convenience as a high-level interactive language. It lends itself especially well to parallel processing kinds of operations such as adding or multiplying matrices. The primary disadvantage of APL is that, in most implementations of the language, it has an interpreter but no compiler. This circumstance means that every time a line of APL code is executed, it first must be interpreted. With the iterative nature of eigenvector programs, the percentage of time that the computer spends interpreting the code becomes a very large percentage of the total solution time. This condition leads one naturally to look for highly compact and efficient ways of coding.

The first step involved identification of a superior algorithm for computing eigenvectors and eigenvalues. The writers found such a promising algorithm in Kaiser's method (1972), as it is known to require only three-fourths of the computing time needed by the Jacobi method while still guaranteeing orthogonal eigenvectors even in the case of close or multiple eigenvalues. The first version of the JK method coded in APL by Adams was an almost direct conversion of the FORTRAN program reported by Kaiser in his 1972 article in *The Computer Journal.* The exceptions were to use APL operators in place of DO loops whenever possible. Subsequently, the program was systematically transformed from serial representation of the FORTRAN program into a highly compressed, parallel-processing code. The result is a four-fold increase in execution speed for the 16th order matrix of full rank. In the development of the efficient APL version, the basic JK method of rotating pairs of column vectors has been preserved; however, the approach has departed from that described by Kaiser. Instead of performing all the single plane rotations serially on columns 1 with 2, 1 with 3, . . ., 1 with n, 2 with 3, 2 with 4, . . ., 2 with n up to (n-l) with n, a scheme was developed to perform several rotations simultaneously through most of each iteration. For example, in a 5th degree matrix, the rotations can be grouped as follows: (1 with 2), (1 with 3), (1 with 4; 2

with 3), (1 with 5; 2 with 4), (2 with 5; 3 with 4), (1 with 2; 3 with 5), (1 with 3; 4 with 5), (1 with 4; 2 with 3), and so on. Notice that rotation of 2 with 3 was permissible at the same time as 1 with 4 because vectors 2 and 3 had already reached the values they would have had in a completely serial scheme. It also should be noted that it is possible to start the second sequence of rotations with vector 1 before completing the rotations on the higher numbered columns. ‘‘Bookkeeping‘‘ for these ‘‘parallelizations’’ becomes complicated but it pays off in improved efficiency. For an *n*th order matrix, as many as *n*/2 rotations can be performed simultaneously. Also, instead of having the same APL code interpreted *n*(*n*-1)/2 times for every iteration, it is interpreted only *n* times.

Highlights of the APL Eigenvector Program

In referring to the APL program JKAEIG shown subsequently, one will notice a number of diamond symbols. Diamonds are used as statement separators in the APL*PLUS[2] implementation of APL. Whenever a diamond is encountered, that which follows the diamond is treated exactly as if it were on a separate line immediately after the preceding statement. For example, in line [7], EPS receives a value, then T, next U, then JA, finally execution proceeds with line [8]. If this program is entered into an APL system not having the diamond separator, or an equivalent feature, it will require about twice as many lines of code. Its operation, however, should not otherwise be affected. Other than the diamond, the code is completely standard APL.

In line [7], EPS is the tolerance used to determine convergence. The remaining lines [7], [8], and [9] are concerned with setting up the ‘‘bookkeeping’’for the scheme of parallel operations.

Lines [11] through [17] perform the planar rotations in a fashion equivalent to that given in Kaiser’s paper, with one exception. It was found that with the parallel scheme of rotations it was more expensive to exclude certain columns from rotation than just to rotate all selected columns simultaneously. Although some of the rotations are infinitesimally small, there is no harm in completing them. Overall convergence is still detected in the same fashion as in the JK method. This program returns the eigenvectors arranged columnwise in descending order of the magnitude of the eigenvalues. With a slight modification of line [17] of the program, the

[2]The writers wish to express their sincere gratitude to Mr. Roy A. Sykes, Jr. of Scientific Time Sharing Corporation for his generous and valuable assistance in helping them to develop efficient, ‘‘non-pornographic’’ APL code.

```
  ∇ R←JKAEIG A;AJ;AK;C;CM;COS;D;EPS;HFP;I;J;JA;K;KA;L;N;NCT;NN;SIN;T;TAN;U
[1]  ⍝
[2]  ⍝ ... PROGRAM FOR FINDING EIGENVECTORS OF A REAL, SYMMETRIC, POSITIVE DEFINITE MATRIX.  REQUIRES ORIGIN 1.
[3]  ⍝ ... AUTHOR:  JAMES ADAMS, UCLA.  BASED ON JK METHOD OF H.F. KAISER, UCB.
[4]  ⍝ ... RETURNS UNIT-LENGTH COLUMN EIGENVECTORS ARRANGED IN DESCENDING ORDER OF MAGNITUDES OF ASSOCIATED EIGENVALUES
[5]  ⍝
[6]  ⍝ ... INITIALIZE A FEW VARIABLES AND SET UP THE PARALLELIZATION CONTROL VARIABLES.
[7]   EPS←1E¯10×(+/,A×A)÷N←(⍴A)[2] ⋄ T←2-2×I←⍳J←N-C←1 ⋄ U←(CM←J⍴ 2 ¯1),J ⋄ JA←(U⍴I)⌊KA←T⊖U↑(J× 1 2)⍴CM↑I+1
[8]   T←N+I←⍳0⌈J-2 ⋄ JA[T;]←JA[T;]+JA[I;] ⋄ KA[T;]←KA[T;]+KA[I;]
[9]   I←CM⍴0,+\L←+/J←KA≠0 ⋄ NCT←⍴JA←(J←,J)/,JA ⋄ KA←J/,KA ⋄ CM←(N=2)+(1↓⍳CM),N-2 ⋄ NN←(NCT-(0⌈N-3)⍴I),N⍴2!N ⋄ →B
[10] ⍝ ... THE ITERATIVE LOOP FOLLOWS:
[11] A:→NCT↓C ⋄ C←CM[C]
[12] B:AJ←A[;J←JA[U←I[C]+⍳D←L[C]]] ⋄ AK←A[;K←KA[U]]
[13]  T←0>TAN←+⌿(AJ+AK)×AJ-AK ⋄ HFP←+⌿AJ×AK ⋄ NCT←NCT-U←+/T<EPS>SIN←|HFP+HFP ⋄ →(U-D)↓A
[14]  U←SIN>TAN←|TAN ⋄ SIN←÷4○TAN←÷⌿U⊖SIN,[0.5] TAN ⋄ COS←(0.5×1+SIN×TAN*U)*0.5
[15]  SIN←(SIN×TAN*~U)÷COS+COS ⋄ U←N,D ⋄ SIN←U⍴(D⍴COS←T⊖SIN,[0.5] COS)×¯1*HFP<0 ⋄ TAN←HFP←0 ⋄ COS←U⍴ 0 1 ⌿COS
[16]  A[;J]←(AJ×COS)+AK×SIN ⋄ A[;K]←(AK×COS)-AJ×SIN ⋄ NCT←NN[C←CM[C]] ⋄ →SIN←COS←B
[17] C:R←A÷(⍴A)⍴(+⌿A×A)*0.5
  ∇
```

eigenvalues could be returned as well. The writers have used eigenvalues only to normalize the eigenvectors. The eigenvalues are represented in line [17] by (+ ⌿ A × A)* 0.5. A listing of the program appears in the Appendix.

Performance of the APL Program

The APL program described in this article, when tested on the Amdahl 470/V6 at Scientific Time Sharing Corporation (STSC), was more than twice as fast, for a 32 × 32 matrix of full rank, as the general EIG routine provided by STSC in the public workspace 20 EIGPAK. This saving in time is important because the EIG routine provided by STSC uses the methods currently regarded as best, namely Householder transformations to tridiagonal form followed by the QL algorithm. The program exhibited even greater gain when compared to the program EIGVECT available in the public workspace 6 LINALG on the IBM 360/91 at the UCLA Campus Computing Network.

From another viewpoint, the APL program described in this paper takes ten times as much CPU time as that required to compile and to execute Kaiser's FORTRAN program with the FORT (H extended) compiler on the 360/91 (using the same 32 × 32 matrix under both systems). When one of the general eigenvector subroutines at

UCLA is employed—namely, HOGIV, the ratio is only four times faster for FORTRAN.

Under APL, eigenvector solutions for very large matrices will probably be prohibitively expensive for most users. An attempt has been made to provide an eigenvector program which is practical for educators and for experimental statisticians.

Future developments in APL systems (such as an APL compiler) may eventually make the program JKAEIG competitive in any environment.

REFERENCES

Kaiser, H. F. (1972). The JK method: A procedure for finding the eigenvectors and eigenvalues of a real symmetric matrix. *The Computer Journal*, 15, 271–273.

Margolin, B. H. (1976). Design and analysis of factorial experiments via interactive computing in APL. *Technometrics*, 18, 135–150.

Wilkinson, L. and Huesmann, L. R. (1973). The use of APL in teaching multivariate data analysis. *Behavior Research Methodology and Instrumentation*, 5, 209–211.