LINEAL: A system for numerical linear algebra

H. M. Khalil and D. L. Ulery

College of Arts and Science, Department of Statistics and Computer Science, University of Delaware, Newark, Delaware 19711, USA

This paper describes a computing system that enables linear algebra problems to be simply programmed and efficiently solved. The main features of this language, its translator, and the two modules for the automatic solution of the eigenproblem and systems of linear equations are described. The power of the system lies in the in-depth analysis techniques employed by these modules and in the quantitative error information accompanying the resultant solutions.

(Received June 1973)

Many of the day-to-day problems which confront numerical mathematicians, statisticians, engineers and econometricians involve solution of linear systems and the general eigenproblem. The solution of these problems requires frequent computation with matrices. In response to this need, many problem-oriented languages have been developed over the past decade which provide special facilities for the manipulation of matrices: APL (Iverson, 1971), ASP (Kalman and Englar, 1965), Burley (Burley, 1967), MARI (Branin, *et al.*, 1965), MAP (Kaplow and Brackett, 1966), MATLAN (System/360), MATRIX (MATRIX), MM (Newbold and Agrawala, 1967), NAPSS (Rice, 1968), OMNITAB II (Agher, Pears and Varner, 1969), and POSE (Schlesinger and Sashkin, 1967). A survey of their capabilities can be found in Smith (1970) and Ulery and Khalil (1974).

It is our judgement that more than ease of matrix manipulation is needed in a special-purpose language for linear algebra. One feature we feel is of paramount importance is 'automatic' problem-solving, in which the system attempts to provide the method of solution for a stated problem. The problem of choosing an appropriate method of solution for problems in linear algebra is not at all simple. The types of matrices which appear in practice vary widely, and an algorithm perfectly suitable for solving one type of system may well be quite inappropriate for use with another. Much work has been done in this area of numerical computation, largely by Wilkinson (1965), and it has reached a very advanced state. The chance of an amateur selecting the algorithm best suited to his system from the large body of algorithms available to solve this class of problems is slim. An inappropriate choice can unfortunately lead to results with such a poor degree of accuracy as to make them totally without value, unbeknownst to the unsuspecting user.

Our aim is to relieve the user of this burden of selection by making the computer act as a professional trained in this area. The user can describe his problem quite simply in the LINEAL language, which resembles standard mathematical notation and is readily learned. The LINEAL system will then automatically select the algorithms, perform analysis, and output the results and statistics pertinent to that solution, thus providing the user with far better results than he would normally have been able to achieve. The system also serves the expert in this area by providing a convenient tool for the comparative study of algorithms, since it provides a measure of the accuracy of the solution.

A few of the existing systems listed above do provide facilities for automatic problem solving. In all cases, however, the number of algorithms and the logic employed to select them is quite limited. LINEAL represents an extension of these efforts. In designing the system our chief objectives were to provide

1. Simplicity and clearness of notation;
2. Concise and powerful operators for the computations of linear algebra;
3. 'Automatic' problem solving for systems of linear equations and the eigenproblem and a measure of goodness of the resulting solution;
4. Economisation of storage;
5. A formal description of the syntax and semantics of the language;
6. High machine independence.

The characteristics of the language, its formal syntax, a part of its formal semantic description, and a brief description of its translator and the two modules for the automatic solution of the eigenproblem and systems of equations are outlined here. APL was used for describing the semantics of LINEAL because of its operator richness and conciseness. It is the authors' belief that APL is the most natural way for describing a language whose data structures are scalars and arrays. Its syntax is described in terms of a modified BNF (Cocke and Schwartz, 1970). Repetitive concatenations of objects are indicated by the notation $\{\ldots\}_i^j$ where i is the minimum number of repetitions required and j is the maximum number of repetitions permitted. Where either index is represented by a variable, the domain of the variable is the metaexpression.

The LINEAL language

Some of the details of the language are formally described in Appendices 1 and 2. We will concentrate here instead on a more informal description of the main features of the language.

The two examples below illustrate the general format of LINEAL code.

Example 1:

```
COMMENT   THIS PROGRAM GIVES A SOLUTION TO THE
          GENERAL EIGENPROBLEM ABX = λX;
DECLARE   M, N;
READ      M, N;
DECLARE   A(M, N), B(N, M);
READ      A, B;
SOLVE     EIGENPROBLEM A * B;
END.
```

Example 2:

```
DECLARE   N, I, J, S;
READ      N;
DECLARE   RHS(N, 1): SYMMETRIC HILB(N, N);
COMMENT   GENERATE HILBERT MATRIX;
LOOP      I = 1 TO N DO
          S = 0;
          LOOP J = I to N DO
                HILB(I, J) = / ((I + J) − 1);
          ENDLOOP J;
          LOOP J = 1 TO N DO
                S = S + HILB(I, J);
          ENDLOOP J;
          RHS(I, 1) = S;
```

```
ENDLOOP I;
SOLVE        LINEQ HILB/RHS;
END.
```

The first example is a program to solve the general eigenproblem, given the coefficient matrices 'A' and 'B'. The second is a program to solve a system of linear equations whose coefficient matrix is an internally-generated Hilbert matrix of order 'N'.

The allowable data types in LINEAL are real, boolean, and alphanumeric. The latter type is used in composing strings. The permitted data structures are scalars and arrays which are treated as units of information and manipulated accordingly.

A LINEAL program is a sequence of statements which normally are executed sequentially. A feature of the language is the absence of labels and unconditional transfer statements, their place being taken by an iterative and a conditional statement. The former invokes an action using a succession of values of its looping index and terminating when the upper limit for the loop is exceeded. The latter invokes one of two alternate actions: the choice is based on the truth or falsity of its boolean expression.

The arithmetic expression permits direct manipulation of scalars and arrays with the basic operators (+, −, *, **) plus a large number of built-in numerical operators particularly suited to matrix computations. Included are operators for finding the absolute value, the trace, transpose, determinant, and inverse of a square array; element-wise reciprocals; the sum of elements of an array; a scaling operator; the maximum and Euclidean norms; and the spectral radius. In addition, built-in operators are included for the trigonometric functions sine, cosine, arctangent and logarithms.

An assignment statement is of the form

$$T = \tau$$

where T is an identifier and τ is an arithmetic expression. τ is evaluated using the current values assigned to the variables appearing within it. The value(s) obtained is stored in T. To avoid specifying precedence of operators, expressions have to be fully parenthesised.

For example,

$$T = \text{NORME}\ (A * B)$$

defines the scalar 'T' as the value of the Euclidean norm of the matrix product A · B. (See Appendix 2 for the semantic description).

LINEAL distinguishes segments of the program called define declarations. Intuitively, a define declaration is equivalent to a procedure: it is a sequence of statements which form a semi-independent unit within the body of the main program. When the procedure is invoked, a branch from the main program to the first statement of the procedure occurs. Exit occurs only from the last statement in the procedure and returns control to the next statement in sequence in the main program. The header of a define declaration includes the name of the procedure followed by a list of formal parameters. A procedure is invoked by a call statement consisting of the word CALL followed by the name of the procedure and a parenthesised list of actual parameters. The procedure call represents a copy of the procedure body in which the actual parameters have been substituted for the dummy variables.

An example of the define declaration is given below:

```
COMMENT    THE SAMPLE MEAN AND SAMPLE VARIANCE
           OF A RANDOM SAMPLE OF SIZE N ARE
           COMPUTED;
DEFINE     STAT(X, N, MEAN, VAR);
           MEAN = X(1);
           VAR = 0;
           IF N GTR 1 BEGIN
              MEAN = (SUM X)(/N);
              VAR = ((NORME(X − MEAN)) ** 2)
                 (/(N − 1));
           ENDCOND;
ENDSTAT;
```

LINEAL permits the assignment of attributes to arrays for efficient storage management and algorithm selection. There is a special statement to perform this task, i.e. the declare statement. The allowable attributes are: vector, general, symmetric, band, band symmetric, triangular, and sparse. These attributes are used to provide control structures for selection of appropriate algorithms and means of efficient storage management. On translating the declare statement, the LINEAL translator associates with each declared array identifier a five-element attribute vector in which the type of the array, its dimensions, and other relevant information are stored for subsequent use.

Finally, one of the most useful features of the system is the solve statement which has the form:

SOLVE TYPE VARS, (options);

where TYPE specifies the problem, i.e. linear equations, eigenvalues and/or eigenvectors; VARS are the equation labels; and options are either empty or represent directives to the system. On executing this statement, the system returns the results in reserved arrays (LAMBDA and/or VECTOR) and all the statistics regarding the solution in another system-reserved array STATUS. The user may exercise the options to:

1. Impose on the system the degree of accuracy required—if this option is not exercised the system sets the accuracy automatically.
2. Specify the algorithm to be used, i.e. control the method of solution. If no algorithm is specified, the system automatically selects the most appropriate algorithm for the particular problem.
3. Suppress output—as mentioned before, the system dynamically allocates arrays for the solution of the problem at hand. Once the problem is solved, the system outputs the results and the relevant statistics, and frees these dynamically allocated arrays. If the suppress option is exercised, the user must save his results by an assignment statement immediately following the solve statement.

For example, to have the system solve the eigenproblem

$$AX = \lambda BX$$

for all eigenvalues and eigenvectors, the user would simply have to write

SOLVE EIGENPROBLEM A/B;

and the system would automatically select an appropriate algorithm, find a solution with the best accuracy possible for the particular machine and matrices, and output the results.

Other features include format-free I/O and the option to free storage space previously reserved for identifiers specified in a declare statement.

The polyalgorithms

Computations performed during program execution are handled by a set of polyalgorithms. Polyalgorithms for arithmetic operations and element manipulation are straightforward. Of more interest are the two polyalgorithms invoked by the solve statement: the LEQ module and the EIGEN module. The design objectives of a module for the automatic solution of numerical analysis problems have been enumerated by Rice (1968) and will not be elaborated upon here. Instead, we will confine our attention to the strategy used.

Each of the modules comprises a number of algorithms together with the logic necessary to apply one or more of these to a given problem. The main reference sources of these algorithms are Wilkinson and Reinsch (1971) and Dekker (1970). The following information, where applicable, is returned to the user by each module:

1. Errors incurred in attempting to solve the problem;
2. The determinant and rank of the coefficient matrix;
3. An estimate of the condition number of the coefficient matrix;
4. The number of iterative improvements required to achieve desired accuracy;
5. An estimate of the number of correct digits in the solution;
6. The method(s) employed.

Each of the component algorithms may contain one or more of the following features: scaling, partial or complete pivoting and iterative improvement.

The strategy employed is based on system size, attributes of the system, and accuracy specified (Khalil and Ulery, 1973; Ulery, 1972). In addition, the EIGEN module takes into account whether both the latent roots and vectors are required or only one of them, as well as the number of required roots and/or vectors.

The LEQ module comprises four decomposition algorithms (Gauss, Crout, Cholesky and Band), two iterative algorithms (with or without a relaxation factor), and Golub and Reinsch's Singular Value Decomposition (SVD), (Golub and Reinsch, 1965). For small, square systems one of the decomposition methods is chosen initially; for large and/or non-dense systems, an iterative method is selected. If the initial method should fail, another approach, based upon the type of failure incurred, is tried. For example, if failure occurs due to rank deficiency, the SVD algorithm is applied. This method of recovery is continued until all applicable paths have been exhausted. The user is then provided with the best possible solution and relevant data concerning the methods tried and the results achieved.

The EIGEN module consists of three submodules: symmetric, general and reduction. The first of these is used to solve $Ax = \lambda x$ where A is symmetric. If A is of small order and the accuracy specified is not high, the Jacobi algorithm is applied. Otherwise A is reduced to tridiagonal form using Householder's

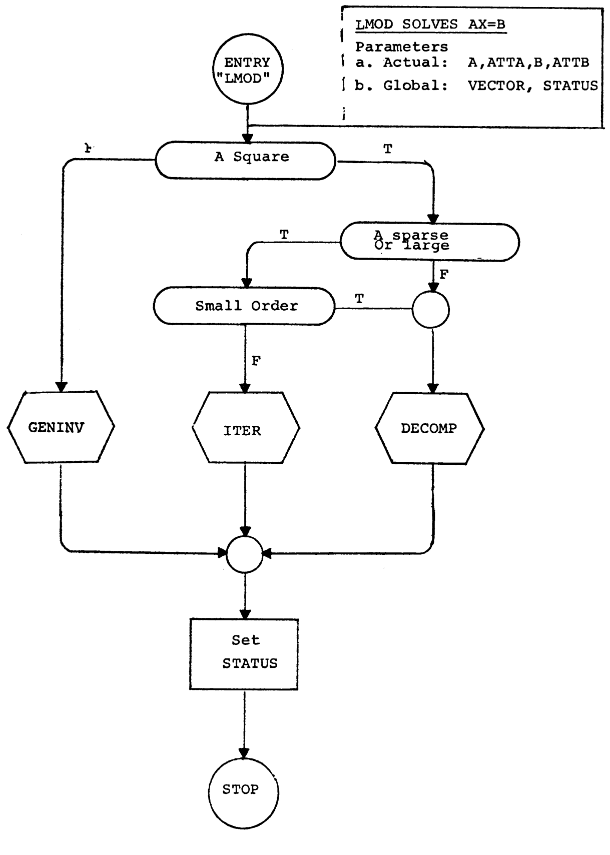


Fig. 1a

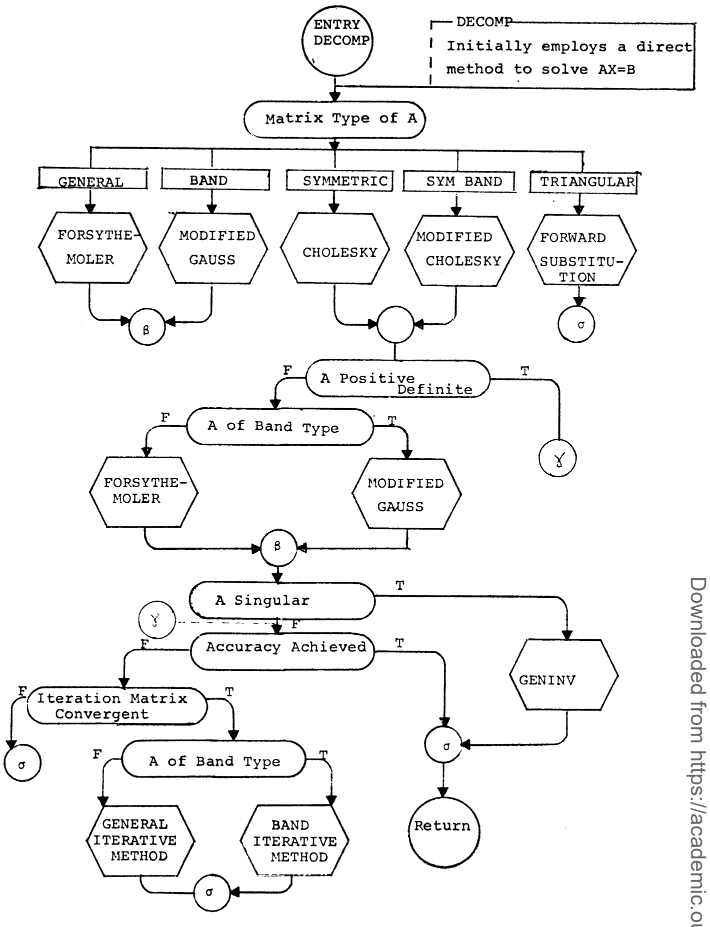


Fig. 1b

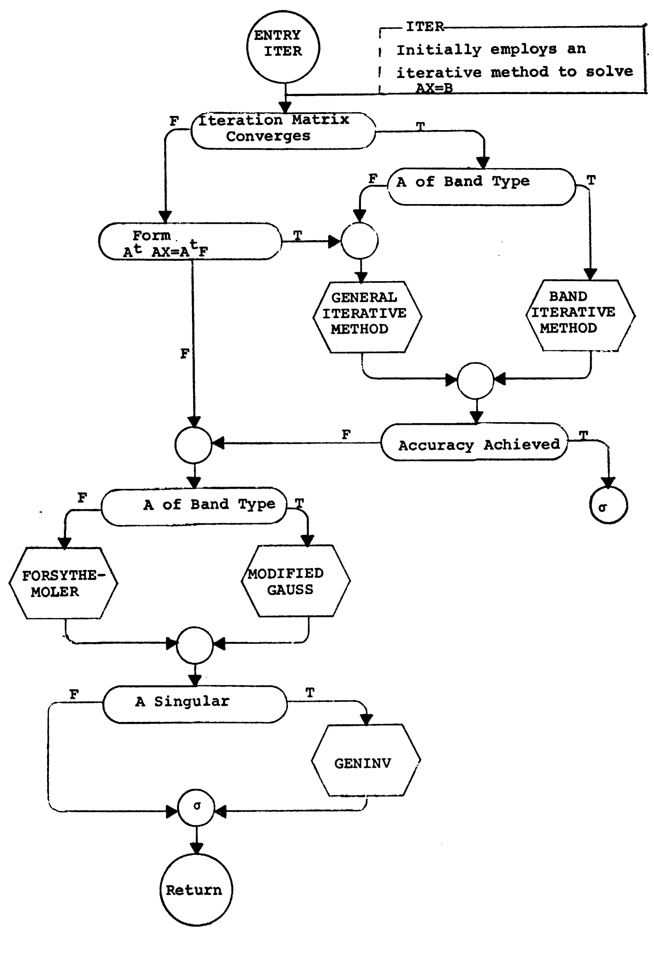


Fig. 1c

transformation. This is followed by one of the QR versions depending upon whether or not the latent vectors are required. On the other hand, if only few latent values and/or vectors are needed, a bisection algorithm followed by inverse iteration is applied. The general submodule comprises the following algorithms: a QR-algorithm for finding the latent roots, an algorithm based upon the norm-reducing Jacobi for computing both values and vectors (Ebelein, 1970), and an iterative algo-

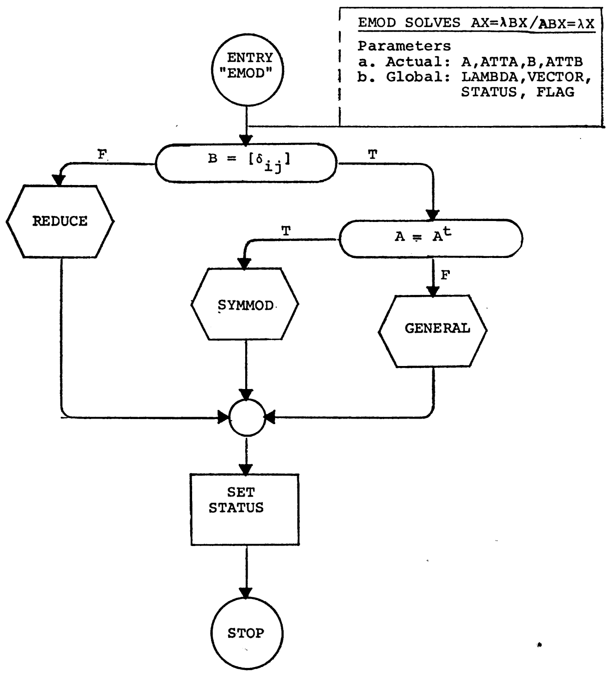
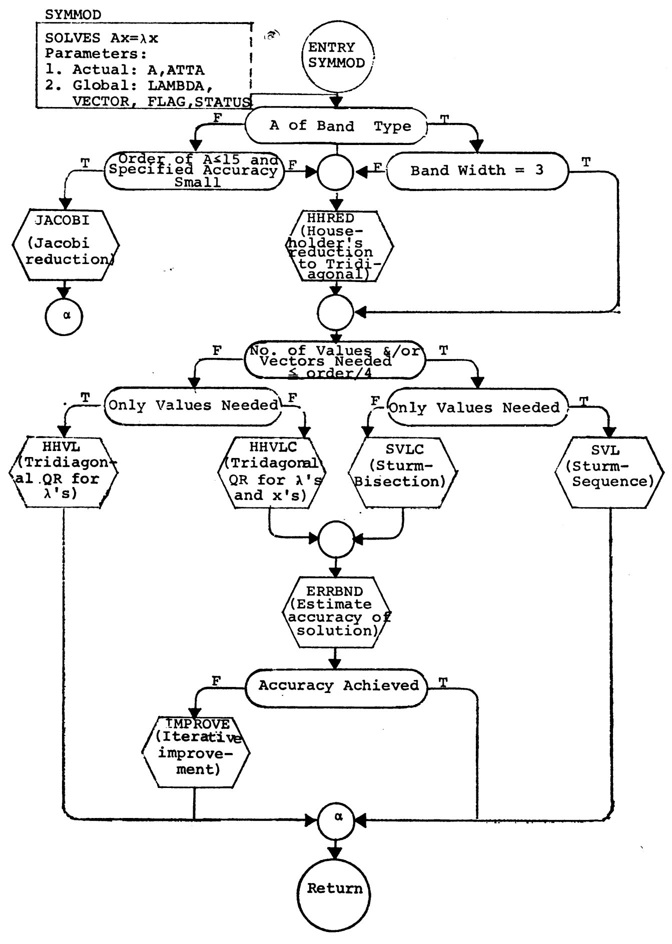
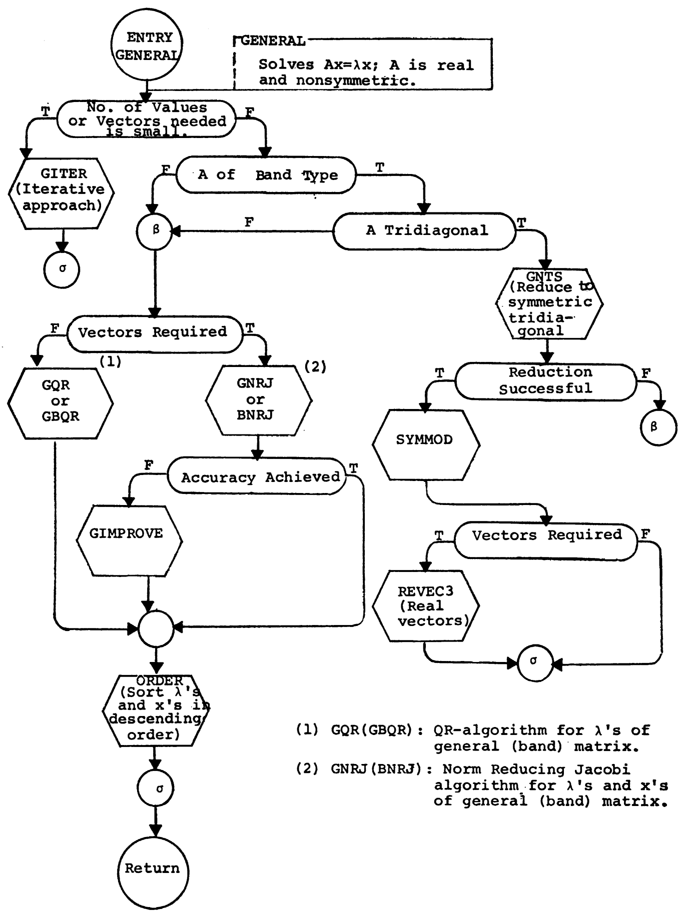


Fig. 2a



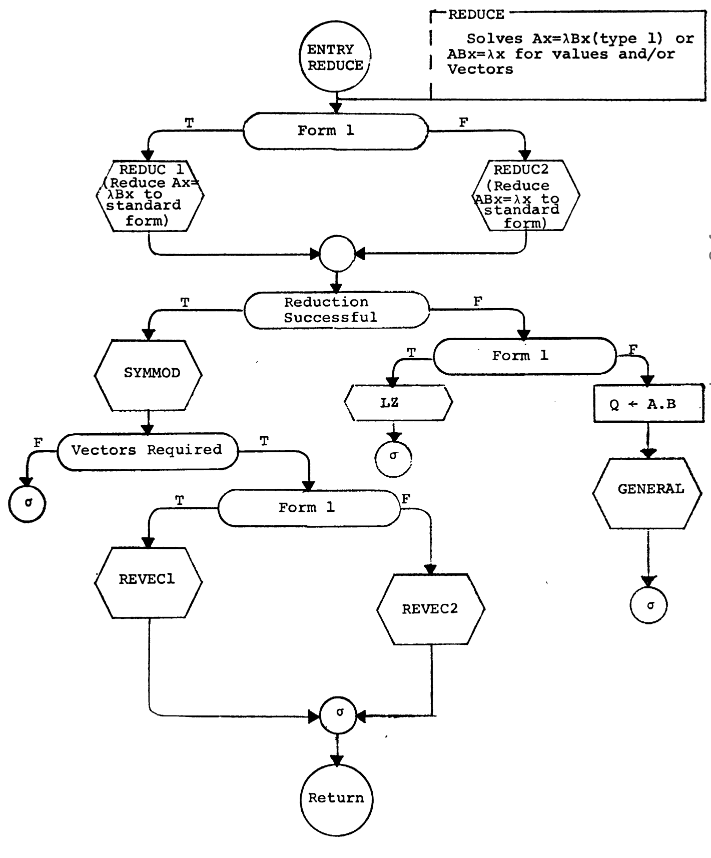
SYMMOD
SOLVES Ax=λx
Parameters:
1. Actual: A,ATTA
2. Global: LAMBDA, VECTOR, FLAG,STATUS
ENTRY SYMMOD
A of Band Type
Order of A≤15 and Specified Accuracy Small
Band Width = 3
JACOBI (Jacobi reduction)
HHRED (Householder's reduction to Tridiagonal)
α
No. of Values &/or Vectors Needed ≤ order/4
Only Values Needed
Only Values Needed
HHVL (Tridiagonal QR for λ's)
HHVLC (Tridagonal QR for λ's and x's)
SVLC (Sturm-Bisection)
SVL (Sturm-Sequence)
ERRBND (Estimate accuracy of solution)
Accuracy Achieved
IMPROVE (Iterative improvement)
α
Return

Fig. 2c



ENTRY GENERAL
GENERAL
Solves Ax=λx; A is real and nonsymmetric.
No. of Values or Vectors needed is small.
GITER (Iterative approach)
σ
A of Band Type
β
A Tridiagonal
GNTS (Reduce to symmetric tridiagonal
Vectors Required
(1)
(2)
GQR or GBQR
GNRJ or BNRJ
Reduction Successful
β
SYMMOD
Accuracy Achieved
GIMPROVE
Vectors Required
REVEC3 (Real vectors)
σ
ORDER (Sort λ's and x's in descending order)
σ
Return
(1) GQR(GBQR): QR-algorithm for λ's of general (band) matrix.
(2) GNRJ(BNRJ): Norm Reducing Jacobi algorithm for λ's and x's of general (band) matrix.

Fig. 2b



REDUCE
Solves Ax=λBx(type 1) or ABx=λx for values and/or Vectors
ENTRY REDUCE
Form 1
REDUC 1 (Reduce Ax=λBx to standard form)
REDUC2 (Reduce ABx=λx to standard form)
Reduction Successful
SYMMOD
Form 1
LZ
Q ← A.B
σ
Vectors Required
σ
GENERAL
Form 1
σ
REVEC1
REVEC2
σ
Return

Fig. 2d

rithm for finding a few dominant values and vectors. Each of these algorithms exists in two versions: one for the band case, and the other for the dense case. Finally, the last submodule is designed to solve the general eigenproblem ($Ax = \lambda Bx$ or $ABx = \lambda x$). If the matrices A and B are both symmetric, the submodule will try to reduce this general problem to a standard form, i.e. $Qy = \lambda y$ where Q is symmetric. Should the reduction succeed, the symmetric submodule is then applied. Otherwise, either the QZ (Moler and Stewart, 1971) or the LZ (Kaufman, 1972) algorithm is applied.

The logic flow employed in both these polyalgorithms is illustrated in **Figs. 1 and 2.**

Implementation of LINEAL

The LINEAL translator will eventually allow for both batch and interactive execution. Effort has been made to keep the translator as machine independent as possible. The interactive portion is currently being written as a FORTRAN IV program. Since the Burroughs 6700 is our available machine and ALGOL '60 has no I/O, the current running version of the translator, an ALGOL '60 program generator, is written in B6700 ALGOL. The polyalgorithms are written in ALGOL '60, however, and are therefore transferable to other machines.

The implemented generator is approximately 5700 cards long, excluding the polyalgorithms. It accepts a LINEAL source program and produces an efficient, semantically equivalent ALGOL '60 program (Carvin and Khalil, 1973). It is modularly organised to facilitate expansion of the language and updating of the underlying numerical algorithms.

The generator uses a modified form of the recursive descent algorithm (Gries, 1971; Leathrum and Fisher, 1970) to parse the input program. A demand lexical analyser is used to provide lexical units upon request from the parser. Code generation is done in parallel with the parse of the input program whenever possible. Although this approach proves to be efficient, it causes some problems in optimising the generated programs.

The generator provides many capabilities which we summarise here:

1. *Memory usage minimisation*
Memory usage is minimised by mapping arrays according to their declared attributes. For example, a band matrix with '1b' and 'ub' codiagonals below and above the main diagonal, respectively, will have the ALGOL declaration $[1{:}N;\ -1b{:}\ ub]$.

2. *Mapping of variable names from LINEAL to ALGOL*
If the LINEAL program contains an ALGOL reserved word as an identifier, the generator will change the name of the identifier.

3. *Recognition of unspecified attributes*
The attributes of a LINEAL array serve not only as a means of minimising storage, but also as a control structure once a polyalgorithm is involved. The generator investigates the declared LINEAL arrays to assure proper assignment of attributes.

4. *Code optimisation*
At this time, the generator optimises the evaluation of arithmetic expressions as well as the folding of expressions (Cocke and Schwartz, 1970).

5. *Documentation of generated programs*
The generator flags unusual constructs, explains the function of some of the polyalgorithms used, and correlates the statements in the ALGOL source with the corresponding statements in the LINEAL source.

6. *Inclusion of polyalgorithms*
The user has the option of either using a compiler version of a polyalgorithm at running time (on the B6700) or of copying the source version of it into his ALGOL program.

7. *Error checking and recovery*
The generator checks for syntactic, semantic and logic errors during both the recognition and generation phases. Once an error is detected, code generation ceases; however, scanning of the remaining source program continues to completion. At run-time numerous checks are made as well. Error messages regarding abnormal job termination indicate probable causes of difficulty. The statement number in the LINEAL source program attributed to the error is specified also.

8. *Options*
The user has available to him options to:

8.1 Display the LINEAL source and/or ALGOL source.
8.2 Compile and/or execute the generated program.
8.3 Copy the generated program onto disc and/or cards.
8.4 Copy the object program produced by the ALGOL compiler onto disc.
8.5 Include in the generated program the source code for required polyalgorithms or code to bind the generated program to a compiled polyalgorithm at run-time.
8.6 Specify maximum size of identifiers.
8.7 Specify maximum amount of processor time to be used for execution of the generated program.

The above characteristics are illustrated by examples in Appendix 3.

Thirty test programs were run on the B6700 to obtain a measure of machine requirements. The average size of each LINEAL program was twelve statements; corresponding to each source statement, the generator produced 5·3 ALGOL statements (excluding the polyalgorithms). The average processor time required for the translation was 1·8 seconds. (It is expected that the average time/statement will be reduced as the size of the source program increases.) The generated ALGOL program required an average of 1·9 seconds for compilation.

Summary

A specialised, high-level, language-oriented system for automatically providing optimal solutions to the common computational problems of linear algebra is presented. One important characteristic of the system is the assignment of array attributes to provide efficient storage management and appropriate algorithm selection. Another is the use of the LEQ module and the EIGEN module to automatically solve systems of linear equations and the eigenproblem and provide quantitative error information with respect to these solutions. The system is well-adapted for use by both the low-level user whose knowledge of programming and numerical mathematics is limited, and the sophisticated user interested in the comparative study of different algorithms as well as the accuracy of the solution.

Acknowledgement

This work was partially supported by two University of Delaware Research Foundation grants.

Appendix 1 Syntax of LINEAL

```
⟨program⟩            ::= {⟨sentence⟩;}_0^n END.
⟨sentence⟩           ::= ⟨declaration⟩ | ⟨stmt⟩
⟨declaration⟩        ::= ⟨sd decl⟩ | ⟨def decl⟩
⟨stmt⟩               ::= ⟨sd st⟩ | ⟨if st⟩
⟨sd st⟩              ::= ⟨assign st⟩ | ⟨comment st⟩ | ⟨call st⟩ | ⟨control st⟩ | ⟨erase st⟩ | ⟨IO st⟩ | ⟨solve st⟩
⟨d-sentence⟩         ::= ⟨sd decl⟩ | ⟨stmt⟩
⟨letter⟩             ::= A | B | . . . | Y | Z
⟨digit⟩              ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
⟨bi-op⟩              ::= + | − | * | **
⟨uni-op⟩             ::= SIN | COS | ATAN | LN | EXP | ABS | TRACE | DET | RANK | INVERSE | TRANSPOSE | SCALE | SUM |
                         NORM1 | NORM2 | NORME | − | /
⟨relation⟩           ::= GTR | GEQ | EQL | LEQ | LSS | NEQ
⟨eigen algorithm⟩    ::= JACOBI | HOUSEHOLDER | ITERATIVE | NONSYM
⟨leq algorithm⟩      ::= CROUT | GAUSS | CHOLESKY | SOR | LSQ
⟨attribute⟩          ::= SYMMETRIC | TRIANGULAR | IDENTITY | BAND | SYMBAND | SPARSE | GENERAL
⟨keyword⟩            ::= END | DECLARE | READ | WRITE | LOOP | ENDLOOP | FREE | SOLVE | TOLERANCE | USE | IF |
                         BEGIN | ENDCOND | DEFINE | CALL | EIGENPROBLEM | EIGENVALUES | EIGENVECTORS | LINEQ |
                         SUPPRESS | COMMENT
⟨special⟩            ::= LAMBDA | VECTOR | STATUS | FLAG
⟨integer⟩            ::= {⟨digit⟩}_1^11
⟨number⟩             ::= {+ | −}_0^1 {⟨digit⟩}_0^(i≤11) {.{⟨digit⟩}_0^(11−i)}
⟨string⟩             ::= {x^n | n > 0 and x ∊ symbol excluding; and"}
⟨scalar id⟩          ::= ⟨letter⟩{⟨letter⟩ | ⟨digit⟩}_0^5
⟨array id⟩           ::= ⟨letter⟩{⟨letter⟩ | ⟨digit⟩}_0^5
⟨def id⟩             ::= ⟨letter⟩{⟨letter⟩ | ⟨digit⟩}_0^5
⟨id⟩                 ::= ⟨scalar id⟩ | ⟨array id⟩
⟨id list⟩            ::= ⟨id⟩{, ⟨id⟩}_0^n
⟨subscripted id⟩     ::= ⟨array id⟩(⟨subscript list⟩)
⟨subscript list⟩     ::= ⟨sexpr⟩{, ⟨sexpr⟩}_0^1
⟨sexpr⟩              ::= ⟨digit⟩ | ⟨scalar id⟩
⟨simple id⟩          ::= ⟨elementary id⟩ | ⟨number⟩
⟨elementary id⟩      ::= ⟨scalar id⟩ | ⟨subscripted id⟩
⟨variable⟩           ::= ⟨elementary id⟩ | ⟨array id⟩
⟨variable list⟩      ::= ⟨variable⟩{, ⟨variable⟩}_0^n
⟨scalar expr⟩        ::= {⟨simple id⟩ ∪ ⟨bi-op⟩ ∪ | ⟨uni-op⟩ ∪ | ε}_1^1 ⟨simple id⟩
⟨arithmetic expr⟩    ::= {⟨primary⟩ ∪ ⟨bi-op⟩ ∪ | ⟨uni-op⟩ ∪ | ε}_1^1 ⟨primary⟩
⟨primary⟩            ::= ⟨number⟩ | ⟨variable⟩ | (⟨arithmetic expr⟩)
⟨boolean expr⟩       ::= ⟨scalar expr⟩ ⟨relation⟩ ⟨scalar expr⟩
⟨comment st⟩         ::= COMMENT ∪ ⟨string⟩
⟨IO st⟩              ::= {READ | WRITE}_1^1 {∪ ⟨paren⟩ ∪ ⟨variable list⟩ | ∪ '⟨string⟩'}_1^1
⟨paren⟩              ::= (⟨subscript list⟩) | ε
⟨assign st⟩          ::= ⟨variable⟩ = ⟨arithmetic expr⟩
⟨control st⟩         ::= LOOP ∪ ⟨*scalar id⟩ = ⟨sexpr⟩ ∪ TO ∪ ⟨sexpr⟩ ∪ {BY ∪ ⟨sexpr⟩ ∪}_0^1
                         DO ∪ {⟨d-sentence⟩;}_0^n ∪ ENDLOOP ∪ ⟨*scalar id⟩
⟨erase st⟩           ::= FREE ∪ ⟨id list⟩
⟨call st⟩            ::= CALL ∪ ⟨def id⟩ (⟨primary⟩ {, ⟨primary⟩}_0^n)
⟨solve st⟩           ::= SOLVE ∪ ⟨paren⟩ ∪ ⟨solve word⟩ ∪ ⟨id expr⟩ ⟨error expr⟩ ⟨use expr⟩ ⟨suppress expr⟩
⟨solve word⟩         ::= LINEQ | EIGENVALUES | EIGENVECTORS | EIGENPROBLEM
⟨id expr⟩            ::= ⟨array id⟩{{* | /}_1^1 ⟨array id⟩ | ε}_1^1
⟨error expr⟩         ::= ,TOLERANCE ⟨paren⟩ | ε
⟨use expr⟩           ::= ,USE ∪ {⟨leq algorithm⟩ | ⟨eigen algorithm⟩}_1^1 | ε
⟨suppress expr⟩      ::= ,SUPPRESS | ε
⟨if st⟩              ::= IF ∪ ⟨boolean expr⟩ ∪ BEGIN ∪ {⟨sd st⟩;}_0^n ∪ ENDCOND
⟨sd decl⟩            ::= DECLARE ∪ ⟨att spec⟩{:⟨att spec⟩}_0^n
⟨att spec⟩           ::= ⟨attrib⟩ ∪ ⟨id⟩⟨paren⟩{, ⟨id⟩⟨paren⟩}_0^n
⟨attrib⟩             ::= SYMMETRIC | TRIANGULAR | IDENTITY | GENERAL | SPARSE (⟨integer⟩) | BAND (⟨integer⟩,
                         ⟨integer⟩) | SYMBAND (⟨integer⟩) | ε
⟨def decl⟩           ::= DEFINE ∪ ⟨*def id⟩ (⟨id list⟩); {⟨d-sentence⟩;}_1^n END ∪ ⟨*def id⟩
```

Appendix 2 Semantic description

(*a*) ⟨assign st⟩ ::= ⟨var⟩ = {⟨primary$_1$⟩ ∪ ⟨bi-op⟩ ∪ | ⟨uni-op⟩ ∪ | ε} ⟨primary$_2$⟩
Let t: LINEAL → APL, then t(⟨var⟩) → T; t(⟨primary$_1$⟩) → A; t(⟨primary$_2$⟩) → B, and the semantics of the form T = A⟨bi-op⟩B is given by:

LINEAL Statement	ρA	ρB	ρT	*Conditions for which 't' is defined*	*Semantics t : LINEAL → APL*
T = A + B					T ← A + B
T = A − B					T ← A − B
T = A * B					T ← A × B
T = A + B		M	M	(ρT) ≥ M	T ← A + B

LINEAL Statement	*ρA*	*ρB*	*ρT*	*Conditions for which 't' is defined*	*Semantics t : LINEAL → APL*
T = A − B		M	M	$(\rho T) \geq M$	T ← A − B
T = A * B		M	M	$(\rho T) \geq M$	T ← A × B
T = A + B		M N	M N	$\wedge/(\rho T) \geq M, N$	T ← A + B
T = A − B		M N	M N	$\wedge/(\rho T) \geq M, N$	T ← A − B
T = A * B		M N	M N	$\wedge/(\rho T) \geq M, N$	T ← A × B
T = A + B	M		M	$(\rho T) \geq M$	T ← A + B
T = A − B	M		M	$(\rho T) \geq M$	T ← A − B
T = A * B	M		M	$(\rho T) \geq M$	T ← A × B
T = A + B	M	M	M	$(\rho T) \geq M$	T ← A + B
T = A − B	M	M	M	$(\rho T) \geq M$	T ← A − B
T = A * B	M	M			T ← A + . × B
T = A + B	M	L K		NONE	ERROR
T = A − B	M	L K		NONE	ERROR
T = A * B	M	L K	K	$(M = L) \wedge (\rho T) \geq K$	T ← A + . × B
T = A + B	M N		M N	$\wedge/(\rho T) \geq M, N$	T ← A + B
T = A − B	M N		M N	$\wedge/(\rho T) \geq M, N$	T ← A − B
T = A * B	M N		M N	$\wedge/(\rho T) \geq M, N$	T ← A × B
T = A + B	M N	L		NONE	ERROR
T = A − B	M N	L		NONE	ERROR
T = A * B	M N	L	M	$(N = L) \wedge (\rho T) \geq M$	T ← A + . × B
T = A + B	M N	L K	M N	$(M = L) \wedge (N = K) \wedge \wedge/(\rho T) \geq M, N$	T ← A + B
T = A − B	M N	L K	M N	$(M = L) \wedge (N = K) \wedge \wedge/(\rho T) \geq M, N$	T ← A − B
T = A * B	M N	L K	M K	$(N = L) \wedge (\rho T) \geq M, K$	T ← A + . × B
T = A ** B					T ← A * B
T = A ** B	M		M	$(\rho T) \geq M$	T ← A * B
T = A ** B	M N		M N	$\wedge/(\rho T) \geq M, N$	T ← A * B

(*b*) ⟨erase st⟩ ::= FREE ∪ ⟨id list⟩

Let t : LINEAL → APL, then t(⟨id list⟩) → LIST and the semantics t(⟨erase st⟩) is given by

```
INDEX ← 1
start : → (INDEX > ρ LIST)/end
LIST[INDEX] ← (ρ LIST[INDEX]) ↓ LIST[INDEX]
INDEX ← INDEX + 1
→start
end : RETURN
```

Appendix 3 LINEAL to ALGOL translator

```
              B6700 LINEAL TO ALGOL TRANSLATOR LEVEL II.03.00

   COMMENT EXAMPLE 1
   PROBLEM - SOLVE A SYSTEM OF LINEAR EQUATIONS WHERE COEFFICIENT
   MATRIX IS A 3 BY 3 HILBERT MATRIX;
   DECLARE RHS(3,1),RES(1,3),I,J;
   GENERAL HILB(3,3),INVHILB(3,3);
 >>WARNING 001>>            *IDENTIFIER LONGER THAN 5 CHARACTERS  <<
   COMMENT GENERATE MATRIX;
   LOOP I = 1 TO 3 DO
        HILB(I,I)=/((I+I)-1);                       1
        RHS(I,1)=0;                                 1
        I=I+1;                                      1
        LOOP J=I TO 3 DO                            1
             HILB(I,J)=/((I+J)-1);                  2
             HILB(J,I)=HILB(I,J);                   2
             ENDLOOP J;                             2
        ENDLOOP I;                                  1
   RHS(1,1)=1;
   SOLVE LINEQ HILB/RHS, USE CHOLESKY,SUPPRESS;
>>WARNING 002>>                        *SHOULD ONLY BE USED WITH
                                        SYMMETRIC ARRAYS       <<
   WRITE"SOLUTION OF LINEAR EQUATIONS.";
   WRITE"COEFFICIENT MATRIX IS:";
   WRITE(12,10)HILB;
   WRITE"RIGHT HAND SIDE IS:";
   WRITE(12,10)RHS;
   WRITE"SOLUTION IS:";
   WRITE VECTOR;
   INVHILB=INVERSE HILB;
>>WARNING 003>>  *IDENTIFIER LONGER THAN 5 CHARACTERS          <<
   WRITE"INVERSE MATRIX IS:";
   WRITE INVHILB;
>>WARNING 004>>  *IDENTIFIER LONGER THAN 5 CHARACTERS          <<
   RES=TRANSPOSE(INVHILB*RHS);
>>WARNING 005>>        *IDENTIFIER LONGER THAN 5 CHARACTERS    <<
   WRITE"INVERSE TIMES RIGHT HAND SIDE IS:";
   WRITE RES;
   END.


   NUMBER OF ERRORS DETECTED = 0000.
   NUMBER OF WARNINGS = 0005.
   TRANSLATION TIME = 000.96 SECONDS PROCESSING.
   PROGRAM SIZE = 0034 CARDS.
   TRANSLATOR COMPILED 04/30/73  09:38 PM.
```

```
SOLUTION OF LINEAR EQUATIONS.

COEFFICIENT MATRIX IS:

ROW  1 OF HILB
1.0000000000       0.50000000000          0.33333333333
ROW  2 OF HILB
0.5000000000       0.33333333333          0.25000000000
ROW  3 OF HILB
0.3333333333       0.25000000000          0.20000000000

RIGHT HAND SIDE IS:

ROW  1 OF RHS
1.0000000000
ROW  2 OF RHS
0.0000000000
ROW  3 OF RHS
0.0000000000

SOLUTION IS:

ROW  1 OF VECTOR
 9.0000000000E 00
ROW  2 OF VECTOR
-3.6000000000E 01
ROW  3 OF VECTOR
 3.0000000000E 01

INVERSE MATRIX IS:

ROW  1 OF INVHI
 9.0000000000E 00     -3.6000000000E 01      3.0000000000E 01
ROW  2 OF INVHI
-3.6000000000E 01      1.9200000000E 02     -1.8000000000E 02
ROW  3 OF INVHI
 3.0000000000E 01     -1.8000000000E 02      1.8000000000E 02

INVERSE TIMES RIGHT HAND SIDE IS:

ROW  1 OF RES
 9.0000000000E 00     -3.6000000000E 01      3.0000000000E 01
```

```
          B6700 LINEAL TO ALGOL TRANSLATOR LEVEL II.03.00

COMMENT EXAMPLE 2

THIS EXAMPLE ILLUSTRATES RUN TIME ERROR RECOVERY
AND THE USE OF THE B6700 OPTION;
 $SET B6700
DECLARE VEC(10);
LOOP I=1 TO 100 DO
     VEC(I)=I;                                   1
     ENDLOOP I;                                  1
WRITE VEC;
END.


NUMBER OF ERRORS DETECTED = 0000.
TRANSLATION TIME = 000.37 SECONDS PROCESSING.
PROGRAM SIZE = 0011 CARDS.
TRANSLATOR COMPILED 04/30/73  09:38 PM.

COMPILATION BEGINS.
COMPILED OK.
EXECUTION BEGINS.



PROGRAM TERMINATED ABNORMALLY AT LINE #2 FROM FAULT WITH
INVALID INDEX.
PROBABLE CAUSE: INDEX GREATER THAN MAXIMUM SPECIFIED IN
DECLARE STATEMENT OR LESS THAN OR EQUAL TO ZERO.
USE '$SET SUBCHK' DURING TRANSLATION FOR MORE INFORMATION.
```

```
          B6700 LINEAL TO ALGOL TRANSLATOR LEVEL II.03.00

    1:       COMMENT EXAMPLE 3
    1:
    1:       THIS EXAMPLE ILLUSTRATES THE SUBCHK OPTION AND
    1:       THE CODE OPTION;
    1:        $SET SUBCHK CODE
    1:       DECLARE VEC(10);
ARRAY VEC[1:10];                                          00000320
    1:       LOOP I=1 TO 100 DO
FOR I:=1 STEP 1 UNTIL 100 DO                              00101000
BEGIN                                                     00101005
    2:       VEC(I)=I;                 1
  VEC[I]:=I;                                              00103010
  COMMENT-FROM ASSIGNMENT STMT, LINE 2;                   00103015
    3:       ENDLOOP I;                1
  END;                                                    00103020
    4:       WRITE VEC;
STRING[0]:="VEC   ";                                      00103025
INOUT(2,VECTOR,VEC,10,0,0,STRING,1);                      00103030
COMMENT-FROM WRITE STATEMENT;                             00103035
    5:       END.
EREXIT:END.                                               00103040


NUMBER OF ERRORS DETECTED = 0000.
TRANSLATION TIME = 000.35 SECONDS PROCESSING.
PROGRAM SIZE = 0011 CARDS.
TRANSLATOR COMPILED 04/30/73  09:38 PM.

COMPILATION BEGINS.
COMPILED OK.
EXECUTION BEGINS.

###############################################################
@LINE#  2 SUBSCRIPT # 1 OF VEC  HAS THE VALUE 11 WHICH IS
OUTSIDE THE ALLOWABLE RANGE.
JOB ABORTED.
```

References

AGHER, D., PEARS, S. T., and VARNER, R. N. (1969). *OMNITAB II: User's Reference Manual*, NBS Tech. Note 552. Washington, D.C.: U.S. Government Printing Office.

BRANIN, F. H., *et al.* (1965). An Interpretive Program for Matrix Arithmetic. *IBM System Journal*, Vol. 4, pp. 2-24.

BURLEY, H. T. (1967). A Programming Language for Linear Algebra, *The Computer Journal*, Vol. 10, pp. 69-73.

CARVIN, K. T., and KHALIL, H. M. (1973). A Program Generator for the LINEAL Language. *Comp. Sc. Conf.*, Columbus, Ohio.

COCKE, J., and SCHWARTZ, J. T. (1970). *Programming Languages and Their Compilers*, 2nd Revised Version, New York: Courant Institute of Math. Sciences, N.Y. University.

DEKKER, T. J. (1970). *ALGOL '60 Procedures in Numerical Algebra*, 2nd Edition, Amsterdam: Mathematisch Centrum.

EBELEIN, P. J. (1970). Solution to the Complex Eigen problem by a Norm Reducing Jacobi Type Method, *Numer. Math*, Vol. 14, pp. 232-245.

GOLUB, G. H., and REINSCH, C. (1965). Singular Value Decomposition and Least Squares Solution, *Numer. Math*, Vol. 7, pp. 403-420.

GRIES, D. (1971). *Compiler Construction for Digital Computers*, 1st Edition, New York: John Wiley and Sons, Incorporated.

IVERSON, K. E. (1971). *A Programming Language*, 2nd Edition, New York: John Wiley and Sons, Incorporated.

KALMAN, R. E., and ENGLAR, T. S. (1965). *A User's Manual for the Automatic Synthesis Program*, Springfield, Virginia: Clearinghouse, U.S. Department of Commerce, NASA CR-475.

KAPLOW, R., and BRACKETT, J. (1966). *MAP: A System for On-line Mathematical Analysis*, Springfield, Virginia: Clearinghouse, U.S. Department of Commerce, AD 476443.

KAUFMAN, L. (1972). *A Generalization of the LR Algorithm to Solve $Ax = \lambda Bx$*, Stanford, California: Comp. Sc. Report 72-276.

KHALIL, H. M. (1972). On the Eigenproblem for the Block Tridiagonal Matrices. *CACM*, Vol. 15, pp. 839-840.

KHALIL, H. M., and ULERY, D. L. (1973). Numerical Solution of the Algebraic Eigenproblem, *Comp. Sc. Conf.*, Columbus, Ohio.

LEATHRUM, J. F., and FISHER, D. A. (1970). *A Language Design System*, Paoli, Pennsylvania: Defense, Space, and Special System Group, Burroughs Corporation.

Matrix: A Conversational Matrix Operations Package for the B6700, Detroit: Burroughs Corporation.

MOLER, C. B., and STEWART, G. W. (1971). *An Algorithm for the Generalised Matrix Eigenvalue Problem $Ax = \lambda Bx$*, Stanford, California: Comp. Sc. Report 232.

NEWBOLD, P. M., and AGRAWALA, A. K. (1967). *Two Conversational Languages for Control Theoretical Computations in Time-Sharing Mode*, Springfield, Virginia: Clearinghouse, U.S. Department of Commerce, AD 664221.

RICE, J. R. (1968). *On the Construction of Polyalgorithms for Automatic Numerical Analysis*, *Interactive Systems for Experimental Applied Mathematics*, New York: Academic Press.

SCHLESINGER, S. I., and SASHKIN, L. (1967). A Language for Posing Problems to a Computer, *CACM*, Vol. 10, pp. 279-285.

SMITH, L. B. (1970). A Survey of Interactive Graphical Systems for Mathematics, *Computing Surveys*, Vol. 2, pp. 261-301.

SYMES, L. P., and ROMAN, R. V. (1969). *Syntactic and Semantic Description of the Numerical Analysis Programming Language NAPSS*, Lafayette, Indiana: Purdue University Technical Report, CDS TR11.

SYSTEM/*360 Matrix Language Application Description*, New York: IBM, H20-0479-0.

ULERY, D. L. (1972). *LINEAL—A Language Oriented System for Solving Problems in Linear Algebra*, Newark, Delaware: M.Sc. Thesis, University of Delaware.

ULERY, D. L., and KHALIL, H. M. (1974). Survey of Languages for Numerical Linear Algebra, *The Computer Journal*, Vol. 17, pp. 82-88.

WILKINSON, J. H., and REINSCH, C. (1971). *Handbook for Automatic Computation*, Vol. 2, Linear Algebra, New York: Springer-Verlag.

WILKINSON, J. H. (1965). *The Algebraic Eigenvalue Problem*, Oxford: Clarendon Press.