A survey of language-oriented systems for numerical linear algebra

Dana L. Ulery and H. M. Khalil

Department of Statistics and Computer Science, University of Delaware, 103 Smith Hall, Newark, Delaware 19711, USA

The manipulation of matrices and the solution of linear algebraic equations are the most frequently encountered tasks in scientific computing. Because linear functions are best understood, problems in computer science, statistics, engineering, mathematics, and optimisation tend to be formulated as sequences of linear systems.

For the 'low-level' user who has to solve problems in liner algebra (i.e, one who has no need to learn FORTRAN, PL/I, or Algol), an approach which is most practical is to provide a programming system for linear algebra that will automatically give a reliable solution to the problem at hand. This paper gives a brief description of a number of such systems, including an example program written for each.

(Received October 1972)

The manipulation of matrices and solution of linear algebraic equations are the most frequently encountered tasks in scientific computing. Because linear functions are best understood, problems in computer science, statistics, engineering, mathematics, and optimisation tend to be formulated as sequences of linear systems. A survey of the computational problems of linear algebra and their sources is given in Forsythe [1.2].

The 'state of art' in matrix computation is fairly advanced: there are many well-tested and well-documented algorithms in the literature for solving standard matrix problems [1.0]. However, their use involves learning a higher level language (e.g. FORTRAN, PL/I, Algol) and understanding numerical mathematics, which is by no means an optimal solution for 'low-level' users, whose primary interest and training are in other areas. Such users would appreciate a programming system for linear algebra that would automatically provide a reliable solution to the problem at hand. This alternative can be developed using a language-oriented system or using an operation-oriented system. The latter requires special purpose software and hardware. For the 'low-level' user, whose computing equipment and geographical location varies widely, the former approach seems the most practical and is the one with which this survey will be concerned. For a survey of operation-oriented interactive systems we refer the reader to [1.4].

The capabilities desired in a programming system for linear algebra can appear as a subset of commands in a general problem-oriented language or form the basis of a small, special-purpose language. Systems with languages of the former class are: MAP, NAPSS, POSE, and APL/360. Systems with languages of the latter class are: MARI, ASP, Burley's System, MM, MATLAN, and MATRIX. A brief description of each of these systems, including an example problem written in the respective language, follows.

Characteristics common to each of the systems surveyed are listed here for convenience rather than being repeated in each description:

1. Arrays are treated as units of information and can be manipulated directly.
2. The systems accept real, square arrays.
3. Operators for basic matrix arithmetic are provided, with the exception of APL/360 which does not include a primitive for the inverse.
4. Each operator in the language is realised by a call on a procedure.
5. Arrays are internally checked for dimension compatibility where applicable.

Problem-oriented languages with subsets for linear algebra

1. *MAP* [2.0, 2.1]

MAP (Mathematical Analysis Program) is a conversational system in use since 1964 on the MIT Compatible Time Sharing System CTSS/7094. The MAP language is a combination of short English phrases and arithmetic equations written with infix notation. Functional notation is used to describe multivalued variables, making the language best suited for manipulations of functions of one variable. The design of the language places heavy emphasis on man-computer interaction. The number of data points permitted for specific problems is quite restricted, and the absence of logical operators and looping facilities forces frequent interaction between the user and the system.

In addition to typical operational functions such as square root and exponential, complex procedures can be called upon directly. Among these are procedures for basic matrix arithmetic, including matrix inversion, and linear least square analysis using a maximum of five fitting functions.

Stored programs written in either MAP or MAD can be called as subroutines, and statements in any other programming language which can call on MAD subroutines may be intermixed with MAP programming statements. An example program using MAP to solve $A = (X^T X)^{-1} X^T Y$ would look like:

Step 1

The M terms of the approximating function must be defined. Let $M = 5$:

$$F1(X) = X_1$$
$$\vdots \qquad \vdots$$
$$F5(X) = X_5$$

where $Y_i = A_1 F1(X) + A_2 F2(X) + \ldots + A_5 F5(X)$.

Step 2

Data must be read into arrays XDATA and YDATA, and YDATA (X), F1(X), . . ., F5(X) defined for the values of X found in XDATA.

Step 3

The following interaction then occurs (user type-in is underlined):

LEAST SQUARE

I CAN FIT EQUATIONS OF THE FORM V(Y) = XA * FA(X) + XB * FB(X) + XC * FC(X) + XD * FD(X) + XE * FE(X) WITH A MAXIMUM OF 5 UNKNOWNS, XA, XB, ETC., AND 100

This research was sponsored by a UDRF Grant to develop a programming system for linear algebra.

DATA POINTS. WHAT IS THE NAME OF THE VARIABLE COMPARABLE TO V(Y)?

YDATA(X)

HOW MANY FUNCTIONS, FA(X), FB(X), ETC., WILL BE REQUIRED TO FIT THE DATA? 5

PLEASE PRINT ON THE NEXT LINE THE NAMES OF THE 5 FUNCTIONS REQUIRED.

F1(X) F2(X) F3(X) F4(X) F5(X)

2. *NAPSS* [3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7]
NAPSS (Numerical Analysis Problem Solving System) is currently being developed as an interactive system at Purdue University's Computer Sciences Centre. The system is made up of four parts: a compiler, an interpreter, an editor, and a set of numerical procedures grouped in modules called 'polyalgorithms'. A polyalgorithm is a group of algorithms, each suited to solving a particular type of problem within the general problem class and the logical structure necessary to determine which should be applied to the problem at hand. With the exception of a few self-contained routines, the system is written in FORTRAN.

The NAPSS language is syntactically a procedural language with a block structure similar to PL/I. Arithmetic expressions in NAPSS permit the direct manipulation of arrays, scalars, and functions with the basic operators. Automatic solution of numerical problems is accomplished through use of the SOLVE statement, which calls the proper polyalgorithm. The power of the language comes from the use of the polyalgorithms, the ability to define new algorithms to override those selected by the system, and the control of errors in the solution.

Among the polyalgorithms implemented at various levels, there are two of interest to this survey:

2.1. *Linear equation solver*
For small systems ($<15 \times 15$), the Forsythe–Moler LU decomposition with iterative improvement is selected; for large systems ($\geqslant 80 \times 80$), the successive overrelaxation (SOR) method is tried. If an intermediate system is band or diagonal, SOR is tried; otherwise it is handled as a small system. The user may specify the desired level of accuracy or the number of iterations to be used.

2.2. *Approximation*
The user specifies a desired accuracy level, and each of seven methods of approximation is tried until the accuracy level is obtained. The error curve and some analysis of the approximation obtained is provided.

Plans for the implementation of NAPSS include its use in a time-sharing environment at on-line remote consoles including graphical capabilities as well as its use for batch jobs.

The following program, see Smith [1.6], displays how a user might solve the least squares problem. Assuming the data have already been read into the one-dimensional arrays X and Y, we have:

```
FOR I ← 1, 2, 3, 4 DO
Q[I] ← SUM (Y[K]X[K] ↑ (2(I − 1)), FOR K ← 1 TO 5)
FOR J ← I, I + 1, . . ., 4 DO
R[I, J] ← SUM (X[K] ↑ (2(I + J − 2), FOR K ← 1 TO 5)
IF I ≠ J THEN R[J, I] = R[I, J];;;
SOLVE R * A = Q FOR A;
TABLE (A)
END
```

In this example the solution vector, A, will be found by the built-in procedure for solving simultaneous linear equations.

3. *POSE* [4.0, 4.1]
POSE (Processing, Organising, and Solving Equations), developed in 1967, is currently running in a limited version on an IBM 360 and being implemented for an IBM 1800. The POSE system consists of a language and a preprocessor to translate the POSE program into an executable FORTRAN program which is then compiled and executed.

The language provides the user with procedural and declarative capabilities. FORTRAN conventions are used to form algebraic problem statements, and POSE statements may be freely intermixed with FORTRAN and/or assembly language statements. The declarative capabilities invoke generators for solving mathematical problems including matrix arithmetic, matrix inversion, solution for a system of linear equations, and basic statistical computations. Each generator is implemented as a single general-purpose algorithm for the stated problem. The form of the declaration statement is reminiscent of a FORTRAN subroutine call.

Although the system is not interactive, input may be via typewriter as well as punched card.

A POSE program for computing the eigenvalues of the matrix $A = B^{-1}CB$ would look like:

```
S.0  CALCULATION SEQUENCE
     EXECUTE S.1
     READ DATA
     EXECUTE S.2
     PRINT RPT. 1(E(1), E(2), E(3), E(4), E(5))
     PROBLEM END
S.1  FUNCTION
     RANGE OF X = 1.0(1.0)5.0
     RANGE OF Y = 1.0(1.0)5.0
     C(X, Y) = X * X + Y * Y
     DIMENSION C(5, 5)
S.2  MATRIX
     A = INV(B) * C * B
     E = EIGENVALUES(A)
DATA CASE 1
MATRIX B(5, 5)
     1.59, 0., 2.4, 3.8, 5.9
     0., 1.9, 0., 3.2, 1.0
     1.21, 0., 1.32, 0., 1.4
     0., 2.0, 3.0, 3.1, 3.2
     1.56, 1.2, 1.0, 51.2, 1.5
END
```

4. *APL*/360 [5.0, 5.1, 5.2]
K. E. Iverson's language, APL, has been implemented as an interactive system for IBM System/360 computers since 1968. The system consists of two parts: a supervisor and the APL interpreter. Included in the supervisor are routines for garbage collection and dynamic storage allocation. The object language of the interpreter is 360 machine-code, making the system entirely machine-dependent. Although the APL processor is interpretive, it provides fast response and execution.

The language is characterised by an extensive set of primitives which apply uniformly to all data structures (i.e. scalars, vectors, and rectangular arrays), and which may be combined in typical mathematical infix notation to form expressions. In addition to the usual arithmetic, logical, and control operators, a variety of operators for array manipulation are available, including the particularly powerful reduction-type operators which are extensions of the scalar dyadic primitives. Another convenient notation is the use of composite symbols comprised of two operators to indicate inner and outer products. As a result, array operations can be stated quite concisely. The complex processes of linear algebra, however, must still be coded by the user.

An appropriate APL program for solving a least squares problem is shown below, see Hellerman [5.0]. The Gauss-Seidel method is used for solving the resulting system of linear algebraic equations. Assume the matrix X and the vector Y have been previously input. We have the following:

```
(1) N ← □
    (INPUT NUMBER OF OBSERVATIONS N)
(2) XT ← ⍉X
    (STORE TRANSPOSE OF X IN XT)
(3) Z ← XT +.× X
    (STORE THE PRODUCT OF X AND ITS TRANS-
      POSE IN Z)
(4) V ← XT +.× Y
    (STORE THE PRODUCT OF XT AND Y IN V)
(5) A ← Z GSP V
    (CALL THE PROCEDURE GSP)
(6) □ ← A
    (OUTPUT REGRESSION COEFFICIENTS)
  ∇X ← A GSP B; J; R
    (PROCEDURE DEFINITION)
(1) X ← ⍳0
    (STORE ZEROS IN THE VECTOR X)
(2) J ← 0
    (INITIALIZE ROW INDEX J)
(3) J ← J + 1
    (ADVANCE ROW INDEX)
(4) R[J] ← B[J] − A[J;] +.× X
    (CALCULATE COMPONENT OF RESIDUAL
    VECTOR)
(5) X[J] ← X[J] + R[J] ÷ A[J, J]
    (UPDATE COMPONENTS OF SOLUTION VECTOR)
(6) → (J ≠ N)/3
    (TEST FOR LAST ROW; IF NOT, BRANCH TO LINE
    3)
(7) → ((∧/(E ≥ | R)) ≠ 1)/2
    (TEST SIZE OF RESIDENTIAL, IF LARGE, BRANCH
    TO LINE 2)
```

Special purpose languages for linear algebra

1. *MARI* [6.0]

MARI is a matrix computation program implemented in 1965 for the IBM 7030. It is under control of the 7030 MCP and requires a knowledge of assembly-language programming.

The language consists of sixteen single-address pseudo instructions whose operands are matrices. An implied operand is required for binary operations. These instructions are executed by branching to the appropriate subroutine. The most common matrix operations are allowed: matrix arithmetic, matrix inversion, solution of a system of linear equations using the inverse, solution of the eigen-problem for real symmetric matrices, and I/O. A program written in MARI can have machine instructions intermixed freely with it. All instructions are executed sequentially in an interpretive mode.

MARI features automatic dynamic storage allocation of main and auxiliary memory. In addition, the system economises on storage and computation time by selecting algorithms and mapping matrices according to the following user-specified types: null, scalar, diagonal, symmetric, and general.

To illustrate the use of the MARI language consider the problem of solving a system of simultaneous linear equations $Ax = f$. We assume that the arrays A and f are in the pool and the directory words are reserved.

```
LINK; B; MXOP      (begin interpretive mode)
MXLI, A            (load inverse of matrix A)
MXST, AI           (store result in AI)
MXR*, F            (right multiply by vector F)
MXST, X            (store result as vector X)
MXEND
```

2. *ASP* [7.0]

ASP (Automatic Synthesis Program) was designed as a language to facilitate finding numerical solutions to problems in linear systems theory, particularly in the areas of control, statistical filtering, and optimisation. It has been used mainly as an analytic design tool where extreme precision was not required, and for educational purposes.

The ASP translator, developed at Ames Research Centre in California, in 1965, is written in FAP for IBM 7090–7094 installations. It consists of about 30 independent subroutines (input/output, logical, data handling, and mathematical) and an executive routine used to control their sequencing. The mathematical subroutines include computation of the exponential of a matrix, solution of discrete- and continuous-time riccati equations, and matrix arithmetic. Within the latter group are routines for inverse, pseudo-inverse, trace, maximum matrix norm, and matrix decomposition. The algorithms used to implement most mathematical subroutines are derived from the calculus of variations.

Both I/O and the instruction word are in fixed-field format, and presume previous familiarity with FORTRAN. Instructions consist of a subroutine name followed by a list of parameters, i.e.

⟨subroutine name⟩, ⟨parameter list⟩

The number of matrix identifiers in a single program is limited to 120, and the size of each matrix must not exceed 16 × 16.

An appropriate sequence of statements for solving a simultaneous set of linear equations $Ax = f$ and printing the result follows:

```
BEGIN
LOAD   A, F      (read the arrays A and F)
INVRS  A, AI     (invert A and store inverse in AI)
MULT   AI, F, X  (store the product A⁻¹.F in array X)
RINT   A, AI, X  (print A, its inverse, the solution vector)
END
```

3. *Burley's System* [8.0]

Burley's system, designed mainly for econometricians, provides facilities to carry out processes of linear algebra together with I/O, branching, looping, and procedure definitions. The system was implemented in 1967 in assembly language on a Titan computer at Cambridge and consists of an interpreter and language.

Instructions in the language have the form of a quadruple of integers (x_1, x_2, x_3, x_4), as in a 3-address assembly-language instruction, where x_1 is the operator and x_2, x_3, x_4 are operands. The absence of an operand is indicated by a zero punch. Burley's repetoire of commands contains 50 unique operators, including solutions for eigenvalues and eigenvectors. Each specifies the type of operation to be performed and the type of data to be manipulated. Although the algorithms employed are not discussed, they appear to be simple ones (for instance, the system $Ax = f$ is solved using $A^{-1}f$).

The data structures scalar, vector, and matrix appear in the system. Each has available ten fixed-length locations in store, represented by an integer in the range 0–9. A sequence of operations can be stored as a vector, then executed as a subroutine. Similarly, loops can be stored as vectors, thus providing the user with two very useful facilities.

Burley illustrates his language by the following program to solve the least squares problem:

```
 T   REGRESSION ROUTINE  (TITLE)
 Ø   Ø   Ø   Ø  (INPUT MATR X INTO MO)
 1Ø  Ø   Ø   Ø  (INPUT VECTOR Y INTO VO)
 8   Ø   Ø   1  (TRANSPOSE OF X IS PLACED IN M1)
 3   1   Ø   2  (XᵀX IS PLACED IN M2)
 5   2   Ø   3  ((XᵀX)⁻¹ IS PLACED IN M3)
33   Ø   Ø   1  (XY IS PLACED IN V1)
33   3   1   2  ((XᵀX)⁻¹ XY IS PLACED IN V2)
 T   REGRESSION COEFFICIENTS (HEADING TO BE
                PRINTED)
19   2   Ø   Ø  (OUTPUTS VECTOR 2—B COEFFICIENTS)
```

If we wish to compute expected values and error terms, we follow with

```
33  Ø  2  3  (XB, PREDICTED VALUE IN V3)
12  Ø  3  4  (ACTUAL-PREDICTED VALUE IN V4)
 T  PREDICTED VALUES  (HEADING TO BE PRINTED)
 T  ERROR TERMS (HEADING TO BE PRINTED)
19  3  Ø  Ø  (OUTPUTS V3, CONTAINING PREDICTED
             VALUES)
19  4  Ø  Ø  (OUTPUTS V4, ERROR TERMS)
-1  Ø  Ø  Ø  (TERMINATES RUN)
```

4. *MM* [9.0]

Matrix Manipulator is a conversational language for control-theoretical computations, implemented in 1967 for the SDS 940 Time-Sharing System at Harvard University Computation Laboratory. It is designed as a simple research and educational tool, requiring no previous programming knowledge on the part of the user and expecting no stringent requirements regarding accuracy and efficiency of the numerical methods used to solve the stated problems.

The MM processor is written in FORTRAN II, stored as a library file, and is under direct control of the FORTRAN Operating System.

The allowable data structures are matrices and scalars, defined by two-character names. No more than 60 matrices, each 10 × 10 or less, may occur in a single program. Of these, no more than six may be stored on file for later use.

Twenty commands, classified as defining commands which specify variable name and dimension, and executable commands which control I/O and specify matrix operations, are provided. The command form is straightforward: a variable name (or names) separated by commas, followed by the command designator, i.e.

⟨identifier⟩, ⟨identifier⟩, ⟨op code⟩ | ⟨identifier⟩, ⟨op code⟩

The standard arithmetic operations including INVERSE, DETERMINANT, and DIAGONAL SUM are available, as well as the command EIGENVALUE, which computes all the eigenvalues and eigenvectors for the specified matrix. Each command is implemented by a single algorithm. When solutions cannot be computed because of such problems as singularities or matrix incompatibility, an error message appears and the system awaits further instructions.

An appropriate program for solving $Ax = f$ using the MM Language follows:

```
<MATRICES
A = 5, 5          (array A has the dimension 5 × 5)
F = 5, 1,         (vector F has five rows)
ALL               (end of declarations)
<START
<A, READ          (read elements of matrix A rowwise)
data by row       (data are input at this stage)
<A, INVERT        (invert matrix A)
<F, READ          (read elements of F)
<A, F, MULTIPLY   (postmultiply A inverse by F)
<..., PRINT       (print the product)
<STOP
```

5. *MATLAN* [10.0]

MATLAN, the System/360 Matrix Language, features an extensive set of operators for matrix arithmetic, matrix manipulations, I/O, and debugging. A MATLAN program consists of a series of statements in the form of a reserved-word-operator (⩽8 characters) followed by a list of operands, which may be real or complex, single or double precision.

Attributes may be assigned to matrices to facilitate the efficient use of storage and computing time. Matrices are mapped into storage as either sparse or general arrays. The attributes available to determine selection of the most appropriate algorithm are: symmetric, symmetric and positive definite, positive, integer, ternary, and Boolean.

The MATLAN translator, written in a combination of 360 FORTRAN IV and 360 Assembly Language, translates MATLAN instructions into FORTRAN code, which is then executed. The algorithms used in MATLAN are in FORTRAN and available in the IBM scientific package. Each algorithm has two versions: a core algorithm and a segmenting algorithm.

MATLAN is under control of the OS/360 supervisor and operates in batch mode.

Below is a sample program written in MATLAN which reads a number of symmetric matrices, forms each inverse, and prints the inverse.

```
MAIN
HEADING  'MATRIX INVERSION'
READ     I                  (read number of
                             matrices)
LOOP     LABLE, J, 1, I     (do the following
                             statements I
                             times)
READ     (A, SYM)           (read lower part
                             of the sym-
                             metric matrix A)
ATTRIB   A, SYM = SYM       (assign A the
                             symmetry
                             property)
INV      A, AINVER          (invert A and
                             store in AINVER)
WRITE    AINVER             (write A⁻¹)
LABLE LOOPEND               (end of loop)
END
```

6. *MATRIX* [11.0]

MATRIX is a conversational matrix operations system implemented for the Burroughs B5500 time-sharing system (1968) as a subset of NUMERALS (Numerical Analysis System), a collection of compatible ALGOL procedures in the area of numerical mathematics. Each of these procedures contains a single algorithm, most taken from the current literature and representing the state of the art at the time of implementation. MATRIX is currently being implemented for the B6700 system.

MATRIX, written as an ALGOL 60 program, contains 16 operators, each of which calls on a single NUMERALS procedure in the area of linear algebra. Included are three options for finding the inverse of a matrix: INVERT, SYMINVERT, and IMPROVINVERSE. The first two operators are for inverting general and symmetric matrices, respectively, and the third for computing the inverse with iterative improvement. The SOLVE operator uses Crout reduction with row pivoting to solve systems of linear equations. Either of two operators, JACOBI or GIVENS, can be used to solve the eigenproblem for real, symmetric matrices. The desired accuracy of the eigenvalues computed by the JACOBI operator must be specified. When GIVENS is used, an estimate of the accuracy of each eigenvalue (minus round-off) is provided.

The system accepts only real square arrays of order ⩽1023.

The following example illustrates the structure of the MATRIX package. User type-in is preceded by a question mark.

```
SHORT FORM DESIRED? SHORT FORM ALLOWS ONLY
TERMINAL INPUT-OUTPUT AND SUPPRESSES MOST
EXPLANATIONS
?YES
ENTER PROGRAM NAME
?INVERT
ENTER ORDER OF MATRIX
?3
ORDER = 3
ENTER MATRIX A
ROW 1
?1, .5, .3333,
ROW 2
?.5, .3333, .25,
```

```
ROW 3
?.3333, .25, .2,
CHANGES AND/OR DISPLAY DESIRED?
?YES
ENTER COMMANDS
? D Ø, Ø*
      1.ØØØØØØØØØØ      Ø.5ØØØØØØØØØ      Ø.3333333ØØØ
      Ø.5ØØØØØØØØØ      Ø.3333333ØØØ      Ø.25ØØØØØØØØ
      Ø.3333333ØØØ      Ø.25ØØØØØØØØ      Ø.2ØØØØØØØØØ
?END*
EXPLANATION OF INPUT PARAMETERS DESIRED?
?YES
TOLERANCE IS USED TO TEST FOR SINGULARITY OF
THE MATRIX. IF THE MAXIMUM ELEMENT IN A
COLUMN OF THE REDUCED MATRIX IS LESS THAN
TOLERANCE, THE MATRIX WILL BE DECLARED
SINGULAR. IF A Ø IS ENTERED FOR TOLERANCE, IT
WILL BE SET BY THE PROGRAM
ENTER TOLERANCE.
?Ø
TOLERANCE = 1.5@ − 1Ø
                A INVERSE
ROW 1
    9.ØØØØ611199Ø    −36.ØØØ32Ø3928     3Ø.ØØØ2999924
ROW 2
  −36.ØØØ32Ø3929     192.ØØ166Ø752Ø   −18Ø.ØØ15479ØØ
ROW 3
    3Ø.ØØØ2999925   −18Ø.ØØ15479ØØ     18Ø.ØØ14399491Ø
WANT TO RUN AGAIN?
?NO
```

7. *LINEAL* [12.0, 12.1]

The LINEAL system is currently being developed at the University of Delaware as an interpretive system for the B6700. The system is designed to provide the 'low-level' user with a simple yet powerful tool for automatically solving the common computational problems of linear algebra.

The LINEAL language has been kept small for reasons of simplicity and speed. Problems are stated in mathematical infix notation, frequently using reserved-word operators for readability. Facilities for format-free I/O, matrix arithmetic, element manipulation, and control are provided. Provision for extension is included via the 'define declaration'.

The major strength of the language lies in its automatic problem-solving capability. The assignment of attributes to matrices facilitates efficient selection of algorithms to accomplish this task, as well as efficient storage management. The Solve Statement, an extension of the ideas used in the NAPSS Solve Statement, provides quantitative error information in addition to solutions for linear algebraic systems and for the general eigenproblem. The user has the option to let the system automatically select the numerical algorithm for a particular problem or to designate it himself; similarly, he may define the degree of precision desired or let the system set it by default. The burden of selecting the most appropriate numerical method is assumed by the LEQ-module and the EIGEN-module. The strategy of each is summarised below:

7.1. *LEQ module*

The module strategy is based on selecting the algorithm that seems most appropriate with respect to system size, coefficient matrix type, and desired accuracy. For dense square systems of reasonable size ($\leqslant 60$) a direct method is applied. If the accu-

Table 1 Comparison of features

	PROBLEM-ORIENTED LANGUAGE WITH SUBSETS FOR LINEAR ALGEBRA			SPECIAL-PURPOSE LANGUAGES FOR LINEAR ALGEBRA							
LANGUAGE PROPERTIES	MAP	NAPSS	POSE	APL/360	MARI	ASP	BURLEY'S MM	MATLAN	MATRIX	LINEAL	
Location	MIT	Purdue	Aerosp. Corp.	IBM	IBM	Ames Res. Ctr.	Cambridge	Harvard	IBM	Burroughs	Univ. of Del.
Machine	CTSS/ 7094	CDC6500	IBM360/ IBM1800	IBM360	IBM7030	IBM 7090-7094	Titan	SDS940	IBM360	B5500/ B6700	B6700
Date of Implementation	1964	1966	1967	1968	1965	1965	1967	1967	1968	1968/71	1971
Operating Mode	IP, IA	IP, IA/ C, B	C, B	IP, IA	IP, B	C, B	IP, B	IP, IA	C, B	IP, IA	IP, IA/C, B
Communic. Simplicity		×	×			×		×		×	×
Array = Single Unit of Information	×	×	×	×	×	×	×	×	×	×	×
Format-free I/O	×	×	×	×			×	×	×	×	×
Independent of Other Prog. Language	×	×		×			×	×	×	×	×
Minimal Machine-depen.		×						×		×	×
Optimal Mapping of Arrays by Type		×			×				×		×
Dynamic Allocation		×		×	×				×	×	×
Auto. Garbage Collec.		×		×					×		×
Intermixable with Another Prog. Lang.	MAD		FORTRAN A.L., M.L.		A.L.				FORTRAN		FORTRAN ALGOL
CRITERIA FOR PROCEDURES											
Storage Minimization		×			×				×	×	×
Best Solution		×									×
Ability to Override Int. Proc.		×									×
TYPE OF OPERATIONS											
Matrix Arithmetic	×	×	×	×	×	×	×	×	×	×	×
Element Manipulation		L		×			L	L	×		×
General Eigenproblem		L	L		L		L	L		L	×
Solution to Systems of Lin. Equations		×	×		×					×	×
Control		×	×	×	×	×	×		×		×
Provisions for Exten.	×	×	×	×			×		×		×
Quant. Error Analysis	×	×								×	×
FORMAL DESCRIPTION											
Syntax		×		×							×
Semantics				×							×

IP: = Interpretive C: = Compiled A.L.: = Assembly Language × : = Included in language IA: = Interactive B: = Batch
M.L.: = Machine Language L: = Limited

acy specified is not achieved, the SOR method is used with the solution vector obtained previously as an initial guess. On the other hand, if the square system is sparse or large, the SOR method is initially employed. If it fails, an estimate of the spectral radius is computed. If it is greater than unity, the least squares method is tried. Otherwise, the SOR method is repeated using A^TA as the coefficient matrix. The module tries the least squares approach for nonsquare coefficient matrices.

7.2. *EIGEN-module*

As in the LEQ-module, the strategy is based upon the type and size of the matrices and the degree of accuracy specified. In addition, the method applied depends upon the number of eigenvalues and/or eigenvectors required and the form in which the problem is stated.

The general eigenproblem is stated in the form $ABx = \lambda x$ or $Ax = \lambda Bx$. If B is the identity matrix and only dominant eigenvalues and/or eigenvectors are required, the iterative method is applied. Otherwise (still assuming $B = I$), if A is symmetric either the Jacobi or Householder methods are applied, the former if the matrix is small and the degree of accuracy is not very high; if A is not symmetric, the modified Jacobi for nonsymmetric matrices is applied. When both A and B are general, symmetric matrices, a reduction procedure is employed. If the reduction is successful, the strategy described above for the standard symmetric case is followed and the proper eigenvectors are appropriately recomputed. Otherwise, the nonsymmetric procedure is applied.

Since LINEAL is designed both for educational purposes and applications involving the use of large-order matrices, future plans include the use of the system in both conversational and batch modes, and as a program generator producing either USASI FORTRAN IV or ALGOL 60 programs.

The following program illustrates the use of LINEAL to solve the general eigenproblem of the form $ABx = \lambda x$, where A and B are large, sparse matrices. Execution of the Solve Statement includes immediate labelled output of the results of the specified computation, including information regarding the method of solution and resulting accuracy.

```
declare N, I, J;
read N, I, J;
declare sparse (I) A(N, N): sparse (J) B(N, N);
read A, B;
solve eigenproblem A*B;
end.
```

Conclusion

This survey reveals considerable range in purpose and scope among existing language-oriented systems for solving problems in linear algebra. Common to all these systems are facilities to handle real square arrays as units of information, internal checking for dimension compatability, and operators for basic matrix arithmetic. MAP, ASP, MM, and MATLAN represent one end of the range, restricting their facilities to the most fundamental operations required of such a system. At the other end of the range, the complex problem-solving capabilities of LINEAL and NAPSS coupled with their provisions for quantitative error information offer the user a highly automated, sophisticated tool for the solution of this class of problems. The two languages are distinguished by their design objectives: NAPSS is designed to reduce the amount of analysis required to solve a wide variety of numerical problems and hence limits the analysis techniques used for each problem class; on the other hand, LINEAL is designed to provide in-depth analysis to the single class of numerical problems in linear algebra.

Bibliography

Comm. ACM = Communications of ACM
Proc. AFIPS = AFIPS Conference Proceedings, AFIPS Press, Montvale, N.J.
FJCC = Fall Joint Computer Conference
SJCC = Spring Joint Computer Conference
ISEAM = Interactive Systems for Experimental Applied Mathematics, M. Klever and J. Reinfelds (editors), Academic Press, N.Y., 1968

1. General

1.0 DEKKER, T. J. (1970). *Algol 60 Procedures in Numerical Algebra.* Mathematisch Centrum, (2 parts).
1.1 FORSYTHE, G. E., and MOLER, C. B. (1967). *Computer Solution of Linear Algebraic Systems.* Prentice-Hall, Englewood Cliffs, N.J.
1.2 SAMMET, J. E. (1969). *Programming Languages: History and Fundamentals.* Prentice-Hall, Inc., Englewood Cliffs, N.J.
1.3 SMITH, L. B. (1970). A Survey of Interactive Graphical Systems for Mathematics. *Computing Surveys*, Vol. 2, pp. 261-301.
1.4 WILKINSON, J. H. (1965). *The Algebraic Eigenvalue Problem.* Oxford University Press, Oxford, England.
1.5 ULERY, D. L., and KHALIL, H. M. (1972). Selected Bibliography on Languages for Numerical Linear Algebra. *SIGNUM Newsletter*, Vol. 7, No. 2.

2. MAP

2.0 KAPLOW, ROY, and BRACKETT, J. (1966). *MAP: A System for On-Line Mathematical Analysis.* Clearinghouse, U.S. Department of Commerce, Springfield, Va., AD476443.
2.1 RUYLE, A., BRACKETT, J. W., and KAPLOW, R. (1967). The Status of Systems for On-Line Mathematical Assistance. *Proc. ACM National Meeting*, pp. 151-168.

3. NAPSS

3.0 RICE, J. R. (1968). On the Construction of Polyalgorithms for Automatic Numerical Analysis. *ISEAM*, pp. 301-313.
3.1 RICE, J. R. (1969). A Polyalgorithm for the Automatic Solution of Non-Linear Equations. *Proc. ACM*, 24th National Conference, pp. 179-183.
3.2 RICE, J. R., and ROSEN, S. (1966). NAPSS: A Numerical Analysis Problem Solving System. *Proc. ACM*, 21st National Conference, pp. 51-56.
3.3 SYMES, L. R. (1970). Manipulation of Data Structures in Numerical Analysis Problem Solving System. *Proc. AFIPS SJCC*, pp. 157-164.
3.4 SYMES, L. R. (1969). *A Mathiematical Problem Solving Language and Its Interpreter.* Ph.D. Thesis, Purdue University, Lafayette, Ind
3.5 SYMES, L. R. (1967). *Evaluaton of NAPSS Expressions Involving Polyalgorithms, Functions, Recursion, and Untyped Variables.* Purdue. University Technical Report CDS TR33.
3.6 SYMES, L. R., and ROMAN, R. V. (1968). Structure of a Language for a Numerical Analysis Problem Solving System. *ISEAM*, pp. 67-68.
3.7 SYMES, L. R., and ROMAN, R. V. (1969). *Syntactic and Semantic Description of the Numerical Analysis Programming Language-NAPSS.* Purdue University Technical Report CDS TR11.

4. POSE

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

4.1. SCHLESINGER, S. I., SASHKIN, L., and REED, K. C. (1968). Two Analyst-Oriented Computer Languages: EASL, POSE. *ISEAM*, pp. 91-96.
5. APL
5.0 HELLERMAN, H. (1967). *Digital Computer System Principles*. McGraw-Hill, Inc., New York, N.Y.
5.1 IVERSON, K. E. (1962). *A Programming Language*. John Wiley and Sons, Inc., New York, N.Y.
5.2 PAKIN, S. (1968). *APL/360 Reference Manual*. SRA, Inc., Chicago, Ill.
6. MARI
6.0 BRANIN JR., F. H., HALL, L. V., SUEZ, J., CARLITZ, R. M., and CHEN, T. C. (1965). An Interpretive Program for Matrix Arithmetic. *IBM Systems Journal*, Vol. 4, pp. 2-24.
7. ASP
7.0 KALMAN, R. E., and ENGLAR, T. S. (1965). *A User's Manual for the Automatic Synthesis Program*. Clearinghouse, U.S. Department of Commerce, Springfield, Va., NASA CR-475.
8. BURLEY
8.0 BURLEY, H. T. (1967). A Programming Language for Linear Algebra. *The Computer Journal*, Vol. 10, pp. 69-73.
9. MM
9.0 NEWBOLD, P. M., and AGRAWLA, A. K. (1967). *Two Conversational Languages for Control-Theoretical Computations in the Time-Sharing Mode*. Clearinghouse, U.S. Department of Commerce, Springfield, Va., AD 664221.
10. MATLAN
10.0 *System/360 Matrix Language (MATLAN) Application Description*. H20-0479-0, International Business Machines, DATA Processing Division, White Plains, N.Y.
11. MATRIX
11.0 *MATRIX, A Conversational Matrix Operations Package for the Burroughs B5500 Time-Sharing System*. Burroughs Corporation, Detroit, Michigan.
12 LINEAL
12.0 ULERY, DANA L. (1972). *LINEAL-A Language-Oriented System for Solving Problems in Linear Algebra*. M.S. Thesis, U. of Delaware.
12.1 ULERY, D. L., and KHALIL, H. M. *LINEAL—A Programming System for Linear Algebra*. Unpublished Report.

Book reviews

Software for Control, conference publication 102, 1973; 168 pages. (*Institution of Electrical Engineers*, £7·00)

This is an interesting and also perhaps slightly specialist topic for those interested in computers. However, process control remains one of the pioneering areas of computer application and one of undoubted industrial significance.

There are several excellent papers here emanating from industrially-based groups who have had a long experience of the use of computers for control purposes and have been prepared to contribute the benefit of their experience. These accounts are most valuable and reveal certainly that they have come to feel the way has not been entirely smoothed but that in the end the influence of this effort on their mode of operations has been quite considerable. They would probably now agree that taken overall the investigation was justified; presumably those who do not agree are keeping mum!

Unlike the more settled realms of scientific and commercial applications where languages of staid maturity such as FORTRAN and COBOL have existed for many years, this area of on-line control (or perhaps one should say instantaneous on-line control for sake of distinction) has never reached a steady state of affairs. Thus it is that this conference's publication is particularly timely and reveals that the time cannot be long removed when the current ferment of discussion and development in the field will need to give rise to some practical recommendations for a degree of standardisation. There is a clear indication among the papers of a school of thought which feels that CORAL 66 is the language to standardise upon; at least five papers make mention of it. A school of thought with more advanced ideas is discernible among advocates of RTL/2. Two of the papers are specifically concerned with this; one outlining some aspects of specification and the other dealing with application of the language.

Today with the plummetting cost of basic electronics the economic horizon for control applications of computing devices is descending rapidly and a much broader range of applications are now acceptable economically speaking. This must ensure an increasing audience for this interesting volume which while being quite valuable to those in the trade, must surely be rated as compulsory reading for those contemplating taking the plunge. For these people the message comes through clearly: that the way is long, but the rewards are worthy of consideration.

J. H. WESTCOTT (London)

A Guide to Teaching about Computers in Secondary Schools, by D. D. Spencer, 1973; 152 pages, (*Abacus Computer Corporation*, Ormond Beach, Florida, $12.95)

My initial reaction to this book was that it would have to be extraordinary good to justify a price of $12.95 for 152 pages. I also feared that it would be loose and woolly in the style of so many 'Ways of teaching about . . .' books. As a result of having read it from cover to cover I am more enthusiastic about it. The author has clearly spent many years in the classroom. His feet are firmly on the ground and whilst his ideas and philosophies derive from the American educational system, many of them are equally applicable over here. Indeed much of the content of the book would be useful to first and second year teachers whether or not they are teaching computing science. Of course, most of the resource material and journals mentioned by the author are American, but I know from personal experience that quite a lot of this is worth ordering from the States. It can be applied very usefully in this country.

The book is divided into three parts. Part 1 considers computer science in the secondary school curriculum. Alas, the author only allocates 36 pages to this section—many of his ideas justify an expansion in more detail. Part 2: Methods of teaching computer science contains much advice which is relevant to all teachers in its first four chapters. Alas, Part 3, on the School administrative uses of the computer, whilst listing some useful applications, is far too brief to be of practical value to the teacher who is involved in designing administrative systems using a computer.

To summarise then, this book really is expensive, but it does contain a lot of valuable information and advice for new teachers, it should certainly find a place in college and university libraries and I hope that many Heads of Departments in schools would consider that they were investing their money wisely to have one to thrust into the hands of the slightly nervous probationer teacher.

W. R. BRODERICK (Romford)