The structure and use of an interpretive program for matrix operations is treated.

The discussion emphasizes the nature of the programming language and the method of storage allocation. The system provides automatic storage allocation for external disk storage as well as for core memory.

An interpretive program for matrix arithmetic

by F. H. Branin, Jr., L. V. Hall, J. Suez, R. M. Carlitz, and T. C. Chen

This paper describes an interpretive system for matrix operations. Featuring automatic storage allocation for all matrices, the system provides virtually the same freedom for the coding of matrix operations as that common in arithmetic operations on individual numbers. The allocation scheme, embracing disk storage as well as core memory, is applicable to a wide range of matrix problems and, in principle, to a broad class of machine configurations.

For the sake of illustration, the discussion treats an experimental version, called MARI, which has been programmed for the IBM 7030. A combination of features makes MARI easier to use and more economical of storage, we believe, than previously-developed matrix interpreters.[1-5]

The MARI program and its associated language were developed along the following guide lines.

- The only details of storage assignment requiring the user's attention should be the designation of a single directory word for each named matrix and the reservation of space for a matrix pool.
- Specification of operations should be as straightforward as in standard symbolic programming.
- Matrices of several different types and of arbitrary dimensions should be allowed. To conserve memory, each type should have its own storage format.
- Selection of subroutines to handle operations on the various matrix types should be completely automatic.

In accordance with these guide lines, the following main features were incorporated in MARI. The matrix types permitted are: null, scalar, diagonal, symmetric, and rectangular. Each matrix is prefixed by two header words that specify its directory word address, type, and numbers of rows and columns.

principal features

The following matrix operations are allowed: addition, subtraction, right- and left-multiplication, transposition, inversion, solution of linear equations, eigenvalue/eigenvector computations for symmetric matrices, and input/output. With few exceptions, these operations are specified by single-address pseudo instructions that involve entire matrices as the unit of information.

Dynamic memory allocation of all matrices, whether in core memory or on disk, is combined with indirect addressing of each matrix by means of a directory word. Thus, matrices can be indexed as if each were a single word in memory, since each matrix is fully symbolized by its individual directory word in all pseudo instructions.

Pseudo instructions, as well as any interspersed machine instructions, are executed interpretively.

Subroutines are selected automatically to execute each operation; where necessary, proper account is taken of matrix types, and checks are made for dimensional compatibility.

Input/output facilities provide for entering matrices into the system, extracting them for special processing, and printing them.

These features require a considerable amount of bookkeeping in allocating memory, in interpreting each matrix operation, and, in selecting the appropriate subroutine to carry out each matrix operation. Even so, this work, occupying only a small percentage of the total running time, is more than justified by the resulting convenience to the user.

control

MARI operates under the control of the 7030 Master Control Program (MCP)[6] and occupies about 3600 full words of core storage. (A key to the symbols used occurs at the end of the paper.) The rest of core storage is available for the programmer's work area (programs, data, etc.) and for a POOL, in which matrices are stored. Except for the area occupied by MCP, disk storage is available for a POOL extension, called DPOOL, and for a special workspace needed during rearrangements of DPOOL. However, the user does not need detailed knowledge of POOL and DPOOL contents to run his program correctly and, in a majority of cases, efficiently.

matrix addressing

Each matrix is referred to indirectly by means of its *directory word* (DWD), which is outside POOL and points to the actual location of the matrix in POOL (or DPOOL). Each matrix in POOL is prefixed by two *header words* (HD1 and HD2), the first of which points back to DWD, whereas the second gives the matrix type and dimensions. A special directory word, called PSAC, is reserved for use as a pseudo accumulator which acts for matrix operations as the counterpart of an ordinary accumulator. For smooth sequencing of MARI, a subsidiary accumulator (AC2) is provided.

All matrix operations expressible in single-address format are

Table 1

Instruction	*Comment*
LINK;B,MXOP	'SUBROUTINE LINKAGE
MXL,A	'LOAD MATRIX A
MXR*,B	'RIGHT MULTIPLY BY MATRIX B
MXST,C	'STORE RESULT AS MATRIX C
MXEND	

written symbolically as floating-point STRAP instructions with the prefix MX. They are assembled by the 7030 STRAP-II assembly program as binary instructions belonging to the set of 16 unused (normally invalid) floating-point codes on the IBM 7030. These special instructions are then executed interpretively under control of the subroutine MXOP.

example

For example, the matrix operation C = A*B would be coded as shown in Table 1. The interpreter, which is within the subroutine MXOP, retains control until MXEND is encountered. The symbolic addresses A, B, and C actually refer to the directory words of the corresponding matrices. During execution, MXL,A simply loads the DWD at A into PSAC. MXR*,B refers to PSAC for the DWD of its implied operand, and to B for the DWD of its addressed operand. Space in POOL is automatically assigned for the result of this operation, and the DWD of the resulting matrix is placed in PSAC. Finally, MXST,C copies the DWD in PSAC into the location C. MXEND terminates the interpretive mode of execution.

Thus, MARI enables the user to perform matrix operations conveniently, treating each matrix as a single word.

Memory organization and matrix types

The core memory of the 7030 is divided into four main parts when MARI is used:

- Programmer's area (for programs, data, workspace, etc.).
- Matrix POOL (administered by the memory allocation program in MARI).
- MARI program and all its subroutines.
- MCP (7030 master control program).

The disk storage is divided into three parts:

- DPOOL, an extension of the matrix POOL.
- RESERV, an area used as a workspace.
- MCP work area.

The method of specifying the boundaries of POOL and DPOOL is described later.

matrix types

MARI handles six matrix types: null, scalar, diagonal, symmetric, columnwise rectangular and rowwise rectangular. The type codes of these matrices can be represented symbolically by the STRAP code of Table 2.

MARI limits rectangular matrices to columnwise representation in the POOL. However, the user can enter a columnwise as well as a rowwise matrix from his data area into the POOL; in either case, the matrix is stored columnwise and assigned the type code 10.0. The method for entering matrices into the POOL is described in more detail later.

Table 2

Name	*Instruction*	*Comment*
NULL	SYN,0.0	'NULL MATRIX
SCAL	SYN,1.0	'SCALAR MATRIX
DIAG	SYN,2.0	'DIAGONAL MATRIX
SYM	SYN,5.0	'SYMMETRICAL MATRIX
COL	SYN,10.0	'COLUMNWISE MATRIX
ROW	SYN,11.0	'ROWWISE MATRIX

Whenever a matrix is entered into the POOL, its directory word (DWD) is stored at the *directory word address*, DWA, in 7030 index word format.[7,8,9] The format of these index words is given in Figure 1. For matrices in the POOL, the *index flag* (XF) is set to zero, and the format is as shown in Figure 2, where HD1A is the header word 1 address of the matrix in the POOL, and SIZE is the number of words occupied by the matrix and its two header words.

For matrices on the disk, XF is set to 1. Thus, representing the 3-bit flag field in octal notation, the format is as shown in Figure 3. Here HD1A is the arc address of HD1 on the disk, each matrix starting at the beginning of an arc. NEXT DWA is the DWA of the next-listed matrix on the disk; if this matrix is the last-listed matrix on the disk, NEXT DWA is zero.

The two header words for each matrix have the format given in Figure 4, where DWA is the directory word address, TYPE is the matrix type code, ROWS is the number of rows, and COLS is the number of columns.

The index flag, XF, is not used in either header word; but in HD1, bit 26 = 1 indicates that this matrix must be left in core (i.e., cannot be moved to the disk) since it is an operand or result in the matrix operation currently being executed. Figure 5 depicts the memory layout for several matrices in the POOL.

Following HD2, the matrix elements are stored according to the conventions presented in Table 3. These conventions are followed in the print format for each matrix.

The different matrix types are allowed in order to economize on storage space and computation time whenever permitted by the structure of a particular matrix. The price paid for this feature is a number of subroutines for handling the various cases that arise. However, the gain in capacity and performance has

Figure 1 Index word format

VF The *value field* is used to modify the address field of instructions to produce an effective address. Numbers are in true form and the twenty-fifth bit is a sign bit. In addressing, the first 18 bits apply to a word address and the remaining 6 bits to a bit address.

FF The *flag field* can indicate programmer-defined properties of the index word. The first bit in the field is called the *index flag* bit.

CF The *count field*, used primarily for loop control, is typically preset to some number, *n*. A CB (count and branch) instruction placed at the bottom of the loop will decrease the count by one, and branch if and only if the new count is nonzero. Concurrent adjustment of the value field can be specified. When a count becomes zero, a refill occurs if called for by a CBR (count, branch, and refill) instruction.

RF The *refill field* usually contains the address of an index word which is placed in the register when refill is requested.

Figure 2 Directory word format for core storage

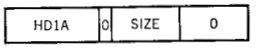


Figure 3 Directory word format for disk storage

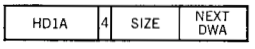
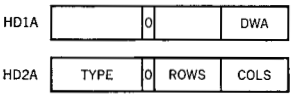


Figure 4 Header word formats



been found well worth the effort. Indeed, several more matrix types (such as tridiagonal, complex, and compound matrices) could easily be added without unduly increasing the number of subroutines.

Figure 5 Matrix storage organization

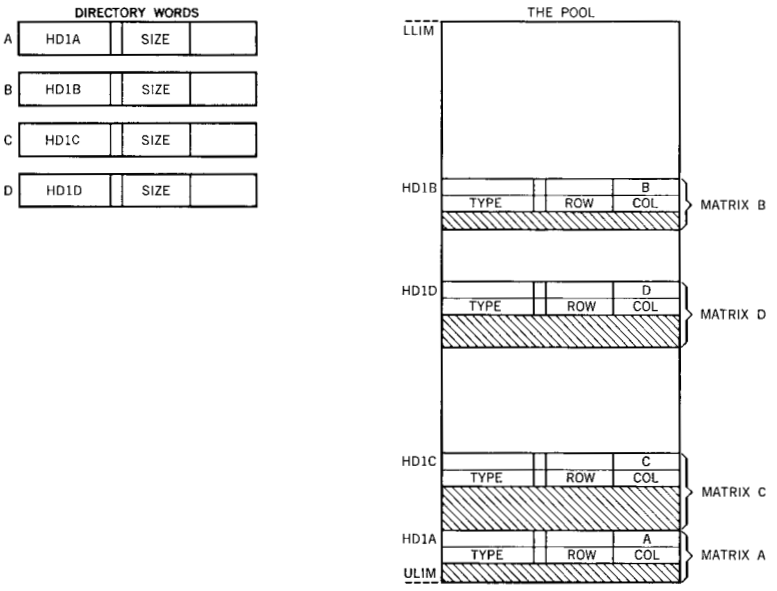


Table 3 Matrix storage formats

Type	*Format*
NULL	no data needed
SCAL	a single floating-point number
DIAG	diagonal elements only, as floating-point numbers in successive storage locations
SYM	diagonal and subdiagonal elements only, as floating-point numbers in successive storage locations ordered columnwise
COL	all elements as floating-point numbers in successive storage locations ordered columnwise

The MARI language and its usage

As mentioned, all single-address symbolic instructions for matrix operations carry the prefix MX, and each instruction is converted by the STRAP-II assembler into one of the 16 invalid floating-point machine codes. These codes, along with any legitimate machine codes that may be interspersed, are identified by the interpretive section of the subroutine MXOP. Bona fide STRAP instructions are executed by means of the EXIC instruction (execute indirect and count), whereas MX pseudo instructions are executed by means of appropriate subroutines.

Table 4 Matrix pseudo instructions

Instruction	*Function*
MXL,A	load matrix A
MXLN,B	load negative of matrix B
MXLT,C	load transpose of matrix C
MXLI,D	load inverse of matrix D
MX+,E	add matrix E
MX−,F	subtract matrix F
MXN+,G	negate matrix in PSAC and add matrix G
MXR*,H	right multiply by matrix H
MXL*,I	left multiply by matrix I
MXL*I,J	left multiply by inverse of matrix J
MXST,K	store matrix K
MXREL,L	release matrix L (from storage)
MXEND	end of matrix interpretation

pseudo instructions

A list of the 13 implemented matrix pseudo instructions is given in Table 4. Although not depicted, the addressed operands in each case can be indexed, if desired, just as in any legitimate floating-point instruction. Sign modifiers, available on standard 7030 floating-point instructions, are not used in matrix pseudo instructions. The operand addresses are treated by the STRAP-II assembler as if they referred to normalized floating-point numbers.

In writing a program of matrix operations, each matrix may be regarded as a single entity that can be loaded into or stored from the pseudo accumulator, PSAC. All binary operations expect to find the implied operand in this pseudo accumulator. But an intermediate result that is not stored from the pseudo accumulator is automatically destroyed by any subsequent operation that places a new result in the pseudo accumulator. Furthermore, storing the contents of the pseudo accumulator into a DWD that already represents a matrix destroys that matrix. Thus the resemblance to arithmetic operations involving single numbers is complete.

illustrative problem

To illustrate the use of the MARI language consider the problem of solving the matrix equation

$$AX = B, \tag{1}$$

where A is nonsingular and the equation is partitioned as follows:

$$\begin{bmatrix} P & Q \\ Q^t & R \end{bmatrix} \begin{bmatrix} Y \\ Z \end{bmatrix} = \begin{bmatrix} S \\ T \end{bmatrix}. \tag{2}$$

Let us assume that the dimensions of these submatrices are

P : 100×100

Q : 100×150

R : 150×150

$Y, S : 100 \times 1$

$Z, T : 150 \times 1$

and that P, Q and R are of arbitrary type, with P nonsingular. We also assume that the matrices P, Q, R, S, and T are already in the POOL, and that a full word is reserved at Y, Z, PI, QTPI, and W for directory words of the final and intermediate results.

Expanding Equation 2, we obtain

$$PY + QZ = S \tag{3}$$

and

$$Q^tY + RZ = T. \tag{4}$$

The solutions to these equations are:

$$Y = P^{-1}(S - QZ) \tag{5}$$

and

$$Z = (R - Q^tP^{-1}Q)^{-1}(T - Q^tP^{-1}S). \tag{6}$$

An appropriate program for solving this problem is shown in Table 5. For the dimensions cited, this sequence of operations, which is depicted in Figure 6, would cause the execution of well over ten million machine instructions. The greatest demand for space in the POOL occurs during the execution of the instruction

Figure 6 Solution of matrix equations by partitioning

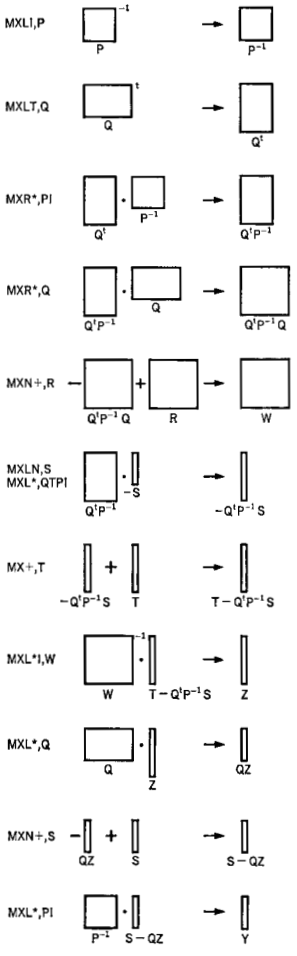


Table 5

Instruction	*Comment*
LINK;B;MXOP	'BEGIN INTERPRETIVE MODE
MXLI,P	'LOAD INVERSE OF MATRIX P
MXST,PI	'STORE RESULT IN PI
MXLT,Q	'LOAD TRANSPOSE OF MATRIX Q
MXR*,PI	'RIGHT MULTIPLY BY PI
MXST,QTPI	
MXR*,Q	
MXN+,R	'NEGATE PREVIOUS RESULT AND
	'ADD MATRIX R
MXST,W	
MXLN,S	'LOAD NEGATIVE OF MATRIX S
MXL*,QTPI	'LEFT MULTIPLY BY QTPI
MX+,T	'ADD MATRIX T
MXL*I,W	'LEFT MULTIPLY BY INVERSE OF
	'MATRIX W
MXST,Z	'STORE RESULT AS MATRIX Z
MXREL,W	'RELEASE MATRICES W AND
MXREL,QTPI	'QTPI SINCE NO LONGER NEEDED
MXL*,Q	
MXN+,S	
MXL*,PI	
MXST,Y	'STORE RESULT AS MATRIX Y
MXEND	'END INTERPRETIVE MODE

MXN+,R. At most, 22,502 words are needed for each of the two operand matrices and the result matrix, since each of these matrices has the dimensions 150 × 150. An additional 102 words are needed for a special matrix, called DSKLST, to be described later. Hence, the POOL must contain at least 67,608 words.

intermediate operand addressing

The instructions MXL, MXLN, MXLT, MXLI, and MXREL specify unary operations and thus require only an addressed operand. All the other instructions (except MXEND, which uses no operands) require both an addressed operand and an implied operand, the latter being obtained from PSAC. The addressed and implied operands are directory words, representing entire matrices.

The instruction MXL simply places a copy of the addressed directory word into PSAC, whereas MXST places a copy of PSAC into the addressed directory word. The instructions MXLN, MXLT, and MXLI, however, require that a new matrix be generated in a region of core memory POOL other than that occupied by the original matrix. A directory word for this new matrix is then placed in PSAC—and nowhere else unless and until an MXST instruction is executed. This process also occurs whenever a new matrix is generated by executing instructions such as MX+, MXR*, etc. Thus, an intermediate result (such as the product Q^tP^{-1} in Equation 6) remains *unnamed* unless it is specifically assigned a name as the result of an MXST instruction.

operand release

Unnamed intermediate results are automatically destroyed either after they have been used as the implied operand of any instruction other than MXST, or after they have been "overwritten" by a load-type instruction. The POOL area occupied by such a matrix is *released*, i.e., made available for reassignment by the memory allocation program, MAP. Explicit release of a matrix may be called for by the MXREL instruction, as illustrated in the previous example.

indexing example

To show how matrices can be indexed and how STRAP instructions can be interspersed with MX pseudo instructions, let us consider the problem of solving, say, 100 different sets of matrix equations of the form

$$A_iX_i = B_i; \qquad i = 1, 2, \cdots 100. \tag{7}$$

We will assume that 100 successive directory words, starting at location A, refer to 100 different square matrices of arbitrary type and dimensions in the POOL and, similarly, that 100 directory words at B refer to corresponding B matrices (or vectors). Space for 100 directory words at X is reserved for results.

Rather than inverting the matrices, the contrived pseudo instruction MXL*I (left multiply by the inverse) is used to call the subroutine for solving the matrix equations in the code shown in Table 6. Note that the interspersion of standard 7030 instructions creates no particular difficulties. Except for the COUNT AND BRANCH instruction (CB+), these instructions are executed under the EXIC instruction. All branch instructions require special handling as described later.

Table 6

Instruction	*Comment*
LINK;B,MXOP	'CALL MXOP
LVI,$5,0.0	'INITIALIZE INDEX REGISTER 5
LCI,$5,100	
JOE MXL,B($5)	'LOAD MATRIX B(I)
MXL*I,A($5)	'LEFT MULTIPLY BY INVERSE OF 'MATRIX A(I)
MXST,X($5)	'STORE RESULT AS MATRIX X(I)
CB+,$5,JOE	'REPEAT 100 TIMES
MXEND	

result dimensionality

In all these computations, the type and dimensions of each result matrix are implied by the types and dimensions of the operand matrices. The programmer need not be concerned with result dimensionality; this question is handled automatically by the MARI program.

Three special subroutines are used for entering matrices into or retrieving them from the POOL, for printing matrices, and for finding eigenvalues and eigenvectors. These subroutines can be called either independently of the MXOP subroutine or from within MXOP.

input

To enter a matrix into the POOL, the subroutine MXIO is used. This subroutine assumes that the matrix elements are stored in the appropriate sequence in some area, called DATA, outside the POOL. Using the matrix name, type, and dimensions as specified in the calling sequence, MXIO requests space in the POOL from the memory allocation program, which creates the directory word and first header word. Then, MXIO transmits to the POOL all the matrix elements from DATA and, as HD2, the last full word in the calling sequence.

The calling sequence for MXIO is given in Table 7. If the matrix type has not yet been defined in the MARI program, if the dimensions are incorrectly specified, or if the matrix to be entered is too large for the POOL, an error message is printed and the program stops.

output

MXIO is also used to retrieve a matrix from the POOL and then place its elements in an external DATA area. In this operation, however, the matrix dimensions are omitted from the calling sequence, and the size of the DATA area is given as shown in Table 8.

Since the actual size and dimensions of the matrix may not be known explicitly, the size of the DATA area may be insufficient. In such a case, an error message is given, and the programmer can find the actual size of the matrix from its directory word. If the size of the DATA area is sufficient, the matrix elements are transmitted from the POOL to DATA without the header words, and a copy of HD2 is placed in the index word (XW) of the calling sequence.

printing

Matrices may be printed, along with an identifying heading, by calling the subroutine MXPRNT as shown in Table 9. The print format employs a heading of identifying information, followed by the type, number of rows, and number of columns. The matrix elements are printed as 14-digit normalized floating-point numbers. Each page can have up to 50 rows and four columns, with the current row and column number appearing in the appropriate positions. The print format for each matrix type is designed to simulate its storage format in the POOL.

eigenproblem

For finding eigenvalues and eigenvectors of a symmetric matrix, a calling sequence corresponding to the matrix equation $AX = X\Lambda$ is used, as shown in Table 10.

Table 7

Instruction	*Comment*
CNOP	
LINK;B,MXIO	'CALL MXIO
,DATA	'AREA CONTAINING MATRIX ELEMENTS
,NAME	'MATRIX NAME (ADDRESS OF DWD)
XW,TYPE,ROW,COL	'MATRIX TYPE CODE AND DIMENSIONS

Table 8

Instruction	*Comment*
CNOP	
LINK;B,MXIO	'CALL MXIO
,DATA	'ADDRESS OF AREA RESERVED 'FOR MATRIX ELEMENTS
,NAME	'MATRIX NAME
XW,SIZE	'SIZE OF DATA AREA

Table 9

Instruction	*Comment*
LINK;B,MXPRNT	'CALL MXPRNT
,NAME	'MATRIX NAME
,IDENT	'ADDRESS OF IDENTIFYING INFOR- 'MATION (ALPHANUMERIC)

Table 10

Instruction	*Comment*
LINK;B,MXEIG	'CALL MXEIG
,A	'NAME OF MATRIX
,LAMBDA	'DIAGONAL MATRIX OF EIGENVALUES
,X	'COLUMNAR MATRIX OF EIGENVECTORS

Table 11

Instruction
CNOP
LINK;B,MXIO
,DATA1
,A
XW,SYM,10,10
LINK;B,MXIO
,DATA2
,B
XW,COL,4,10
LINK;B,MXOP
MXLI,A
MXR*,B
MXST,X
LINK;BE,MXPRNT
,X
,IDX
MXL*,A
MX−,B
.
.
.
.
MXEND

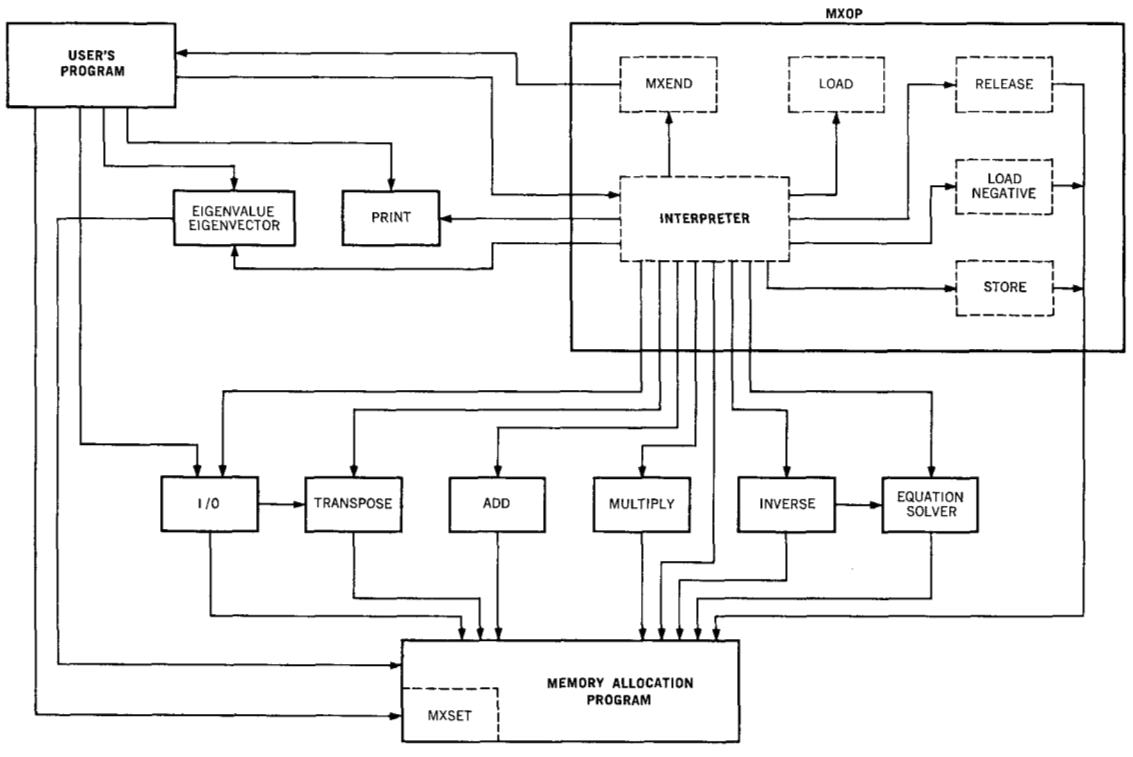
At present, MXEIG does not handle the eigenproblem for asymmetric matrices since these matrices may have complex eigenvalues and eigenvectors. The algorithm used for symmetric matrices is based on Jacobi plane rotations. Extra working space, amounting to $N + (N/2)(N + 1)$ full words, is used by MXEIG and automatically released when the subroutine is finished.

Whenever these three subroutines are called from within the MXOP subroutine, the BE (branch enabled) instruction is used to indicate that the subroutines are to be executed directly instead of in the interpretive mode. This direct mode of execution is allowed because these subroutines return to a special location within MXOP, permitting resumption of the interpretive mode of operation. Thus, it is possible to write the code of Table 11 for printing an intermediate result without interrupting the interpretive mode of execution between the call for MXOP and the execution of MXEND.

Figure 7 shows the hierarchy of subroutines that implement the MARI program. The user's program may call directly any one of the five subroutines MXOP, MXPRNT, MXEIG, MXIO, or MXSET. When MXOP has been called and still has control, the last four can also be called indirectly through MXOP.

Six subroutines are included in MXOP, namely, the interpreter, load, load negative, matrix release, MXEND, and store subroutines. The interpreter acts as the main control center of the MARI program. It updates the MARI instruction counter, selects and

Figure 7 Possible subroutine calls



calls the appropriate subroutines, and updates the pseudo accumulators, PSAC and AC2.

Dynamic memory allocation

The memory allocation program, MAP, administers POOL and DPOOL, keeping a list of all holes (available blocks of storage) in both POOL and DPOOL. All unavailable blocks, except the one reserved for a disk list (DSKLST) explained below, are occupied by matrices. Under the control of MAP, any matrix in POOL may be moved to make room for a new matrix. Although most of the matrices in POOL can also be moved to DPOOL, a specified few (usually the operands in the operation being performed) must remain in core. Thus, MAP can be called upon to perform five different functions:

Table 12

Instruction
LINK;B,MXSET ,LLIM ,ULIM ,DISKMIN

- Set the boundaries of POOL and, initially, set up DPOOL. (This is the problem program's only possible direct request of MAP.)
- Provide a block of specified size to accommodate a matrix.
- Change an unavailable block into an available block when a matrix is released.
- Alter the availability of a matrix for storage on disk.
- Move a matrix from core to disk or vice versa.

POOL boundaries

Before any matrix operation can be executed, the POOL must be specified. The POOL area must provide both for the matrices to be handled and for 102 words for the DSKLST described below. The size of POOL is specified by the calling sequence of Table 12, where LLIM is the address of the first word of POOL (lowest memory address), and ULIM is the address of the last word of POOL. DISKMIN is the minimum matrix size that may be transferred to DPOOL if more room is needed in the POOL. If DISKMIN is zero, MAP assumes a value of 257 words, so that only those matrices larger than half an arc may be sent to DPOOL.

If the boundaries of an existing POOL are moved by a subsequent call of MXSET, some matrices may have to be sent to DPOOL to fit the old POOL into the new POOL area. If the old POOL and new POOL do not overlap, the contents of the region between them may be destroyed.

DPOOL boundaries

During the first execution of MXSET, the space on disk not preempted by MCP is divided into two regions: the DPOOL and the RESERV area. RESERV is used as work space to store the POOL during a disk squeeze, freeing core memory for use in rearranging matrices in DPOOL. RESERV occupies, at most, half of the available disk space, up to a maximum of 64 arcs.

A request for space in the POOL is made to MAP in the following way: the value field of index register 14 specifies the DWA of the matrix that is to occupy the requested space; the count field specifies the size of the matrix.

Whenever a request for memory is made, space is assigned by MAP within the smallest adequate hole and at that end of the hole which is nearest ULIM. The holes are chained together, both

forward and backward, according to both location in the POOL and hole size. Since the first two words in each hole are used for this purpose, a hole of a single word is not permitted. The format of these two words is given in Figure 8.

double-threaded hole lists

In WD1, XF = 1 signifies that this is the first word of a hole (and not HD1 of a matrix); the field NEXT WD1A is the *next word* 1 *address* in order of increasing memory location. PRIOR WD1A is the address of WD1 of the preceding hole in memory. SIZE specifies the number of words in the hole—including both WD1 and WD2.

In WD2, the field NEXT WD2A signifies the *next word* 2 *address* in order of increasing size, and PRIOR WD2A the address of WD2 of the next smaller hole.

The termini of these two chains are at fixed locations outside the POOL, having the format of Figure 9. Here, LLIM WD1A is the WD1 address nearest LLIM, and ULIM WD1A is that nearest to ULIM. FIRST WD2A is the WD2 address of the smallest hole, and LAST WD2A is that of the largest.

Obviously, the refill field at LLIM WD1A must point to HOLL1, whereas the value field at ULIM WD1A must point to HOLLOC in order to complete the forward and backward chains. Similarly, at FIRST WD2A, the refill field must point to HOLSIZ, whereas at LAST WD2A, the value field must point to HOLS1. A diagram depicting several holes and their linkages is shown in Figure 10.

disk hole list

Since holes also appear between occupied areas on the disk, a list of disk holes is maintained in a special reserved area of POOL adjacent to ULIM. This list is set up as if it were a matrix with its two header words as shown in Figure 11, i.e., as a columnwise matrix of 100 rows and 1 column, with DSKLST as its directory word. This matrix is made ineligible for transmittal to the disk by setting bit 26 to 1 in HD1.

Figure 8 Formats for first two words of each hole

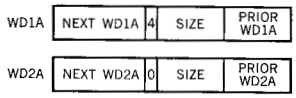


Figure 9 Formats of chain-terminating words

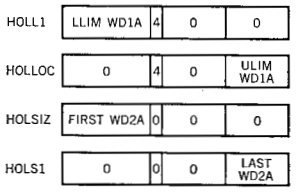
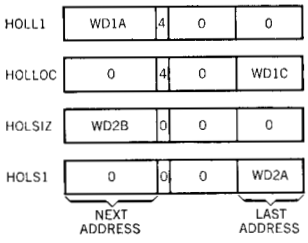


Figure 10 Holes and their linkages



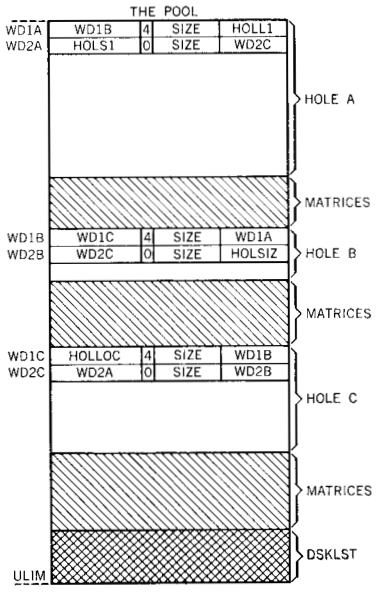
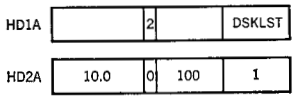


Figure 11 Formats for header words of disk hole list



Although more details of the disk-hole list are given later, it should be noted here that on the very first execution of MXSET, when the POOL is initially established, MAP is requested to assign 102 words to DSKLST. By making this request first, the allocation of DSKLST to the area just adjacent to ULIM is assured.

A word outside the POOL, called HTOTAL, specifies the cumulative total of space in all the holes in the POOL. When a request is made to MAP for space in the POOL, the size of this request is first compared against HTOTAL and then against the size of the largest hole, pointed to by HOLS1. Thus, three situations can arise. First, if HTOTAL is not large enough to accommodate the request, a disk operation must be initiated to make space in the POOL. Second, if HTOTAL is adequate, but the largest hole is not, a "squeeze" operation within the POOL is initiated. Third, if the largest hole is adequate, a search is made for the smallest adequate hole by following the WD2-chain forward from HOLSIZ. These three situations are discussed in reverse order.

hole search

Since holes of just one word cannot be allowed, the term "smallest adequate hole" means any hole whose size exactly equals or exceeds by two or more words the size of the requested space. When the smallest adequate hole has been identified, MAP assigns space at the ULIM-end of this hole. Before returning the results of this assignment to the requesting subroutine, however, MAP updates WD1 and WD2 of this hole to account for its reduced size. If the hole size becomes zero as the result of an exact fit, the WD1 and WD2 chains of the remaining holes are linked around the obliterated hole by updating the words at NEXT WD1A, PRIOR WD1A, NEXT WD2A, and PRIOR WD2A. In addition, MAP updates the DWD of the matrix, for which space has been requested, by setting its value field equal to the HD1 address and its count field equal to the SIZE of this matrix. Finally, the HD1 is stored with its refill field pointing to the DWD and with bit 26 = 1 to indicate that this matrix must not be sent to DPOOL.

POOL squeeze

When HTOTAL is adequate to accommodate a storage request, but the largest hole is too small, a POOL squeeze is required. During this squeeze, the appropriate matrices are moved towards LLIM to enlarge the hole nearest ULIM until this hole exactly satisfies the request. A squeeze is effected in three steps.

First, the WD1 chain of holes is followed backward from HOLLOC until the cumulative sum of hole sizes equals, or exceeds by two or more words, the requested space. If equality is obtained, the WD1 address of the hole nearest LLIM becomes the relocation address, RELOC. Otherwise, RELOC is set equal to this WD1 address *plus* the number of excess words. All matrices between RELOC and WD1 of the hole nearest ULIM must then be squeezed into a contiguous block, starting at RELOC.

Second, using the information in WD1 of the hole containing RELOC (or of the preceding hole, if RELOC = WD1A), HD1 of the adjacent matrix is picked up. The size of this matrix is determined from the corresponding DWD. The matrix is then transmitted

so on, until HTOTAL is large enough for a hole search or POOL squeeze. The disk operation and pool squeeze are shown in Figure 13.

DPOOL squeeze

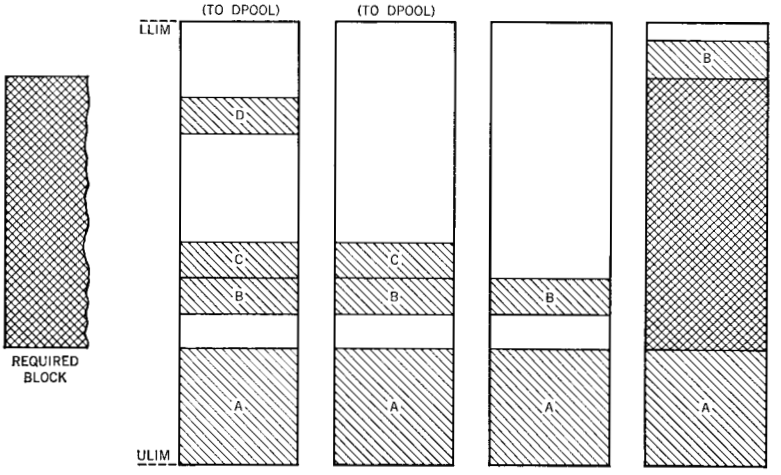
To provide information for a squeeze of DPOOL, the matrices in DPOOL are chained together by the refill field of each DWD. The first link on this chain is the word DWDLNK whose refill field is the DWA of one of the matrices in DPOOL. (The refill field is set equal to zero if DPOOL is empty.) The refill field of the DWD of this matrix then points to the DWD of the "next" matrix in DPOOL—and so on, the last DWD having a zero refill field. The order of matrices in this chain is inconsequential, being determined by the order in which they are put into DPOOL, rearranged, etc.

As described above, DSKLST is the DWD of a special matrix at the ULIM end of POOL containing a list of disk holes. The associated word LSTHO has its value field pointing to the first such disk-hole word, and its count field equal to the number of disk holes. The elements of this disk-hole matrix are in index word format, with the value field containing the arc address and the count field containing the number of arcs of the hole in question. These disk-hole words are ordered according to increasing arc addresses.

A squeeze of DPOOL is initiated by MAP whenever an adequate disk hole cannot be found, but DSKTOT indicates that a combination of disk holes will satisfy the request for space in DPOOL. If DSKTOT is too small to accommodate a given matrix in DPOOL, the next available matrix in POOL is tried. If none can be found, the program stops after printing an error message. The DPOOL squeeze is effected by placing all or part of the POOL onto the RESERV area of the disk and then using the vacated region of POOL as a temporary workspace.

If the size of RESERV is smaller than that of POOL, only part of the POOL is sent out to RESERV. The list of disk holes is then transmitted to the vacated area of POOL. The remaining part

Figure 13 Disk operation and pool squeeze



of this area—which must be large enough to accept at least one full arc from the disk—is used in squeezing the DPOOL. All the matrices in DPOOL are then moved to a contiguous block, starting at the lowest available arc address. The list of disk holes and the chain of matrices in DPOOL are searched and updated appropriately during this squeeze operation. Since the number of disk holes can conceivably exceed 100, additional increments of 50 words at a time are automatically made for the disklist matrix, always leaving this matrix at the ULIM end of the POOL.

It should be noted that whenever a matrix in DPOOL is fetched into POOL to participate in an operation, this matrix is automatically released from DPOOL and all essential bookkeeping fields are updated.

The interpreter

The subroutine MXOP contains the interpreter as well as the code for executing the matrix instructions MXL, MXLN, MXREL, MXST, and MXEND. All instructions included between the code LINK;B,MXOP and MXEND are within the domain of MXOP and can be considered as part of the calling sequence to MXOP, with MXEND as terminus.

The instructions within the domain of MXOP are examined, interpreted, and executed sequentially under full control of the interpreter. Each such instruction is tested for the bit patterns that correspond to the 16 invalid floating-point codes reserved for MX instructions.

indirect execution

Ordinarily, bona fide machine instructions, after being identified as such, are executed indirectly by means of the 7030 instruction EXIC. This process enables the interpreter to keep its place. However, a test is first made for the sequence LINK;BE,$\cdots$, since this code is used to signal the direct rather than interpretive execution of the subroutines MXSET, MXIO, MXPRNT, and MXEIG. These subroutines are peculiar in that they return to the fixed-location EXIT inside MXOP. In this way, the interpreter is enabled to relinquish its control to any of these special subroutines within the domain of MXOP and then to regain control smoothly at EXIT without losing its place.

Since these subroutines can also be called from outside the domain of MXOP, a test is made at EXIT to determine whether MXOP was in control before the call was made. Control is then either relinquished or retained, as the case requires.

No other subroutines within the domain of MXOP may be called with LINK;BE,$\cdots$, since MXOP would be forced to relinquish control. Any subroutine, except MXSET, MXIO, MXPRNT, or MXEIG, may be called from within the domain of MXOP by the sequence LINK;B,$\cdots$, but this subroutine will then be executed interpretively. Since this mode of execution is very inefficient, any long string of machine code should be executed outside the domain of MXOP.

branching

Whenever an attempt is made to execute a successful branch instruction (such as B, BB, BIND, or CB) by means of EXIC, an EXE interrupt occurs, because the location counter has been altered out of sequence. MXOP negotiates with MCP to take this interrupt and handle it as follows: the subject branch instruction is copied from its original location to a place inside MXOP, where its address is replaced by a fixed address in MXOP. This branch instruction is then executed. At the fixed address to which it branches, a transfer to its original effective address is made under control of MXOP.

In this way, the interpreter can maintain control and follow the proper sequence of instructions within its domain. However, a branch to MCP cannot be handled properly by MXOP, since the instruction B,$MCP would be removed from its context before being executed; this vitiates the calling sequence for MCP.

indexed operands

When an MX instruction has been identified by MXOP, the effective address of the DWD of its addressed operand (if any) is computed. (Indexing of MX operands is freely permitted, using index registers 1 through 13.) This DWD is then examined to see if the subject matrix is already in POOL. If not, it is automatically fetched from DPOOL. If the matrix does not exist, an error message is printed.

HD1 of the matrix is then flagged (bit 26 = 1) to indicate that this matrix is essential to the operation about to be executed and, therefore, not available for transmittal to DPOOL.

MXOP then relinquishes control to the appropriate arithmetic subroutine, passing along to it the DWA, SIZE, and HD2 of the implied operand (if any) and of the addressed operand.

After the arithmetic subroutine has done its task, it branches to a fixed location in MXOP (called RETURN), leaving the DWD of the result matrix in the fixed location AC2, also inside MXOP. AC2 acts as a temporary second accumulator.

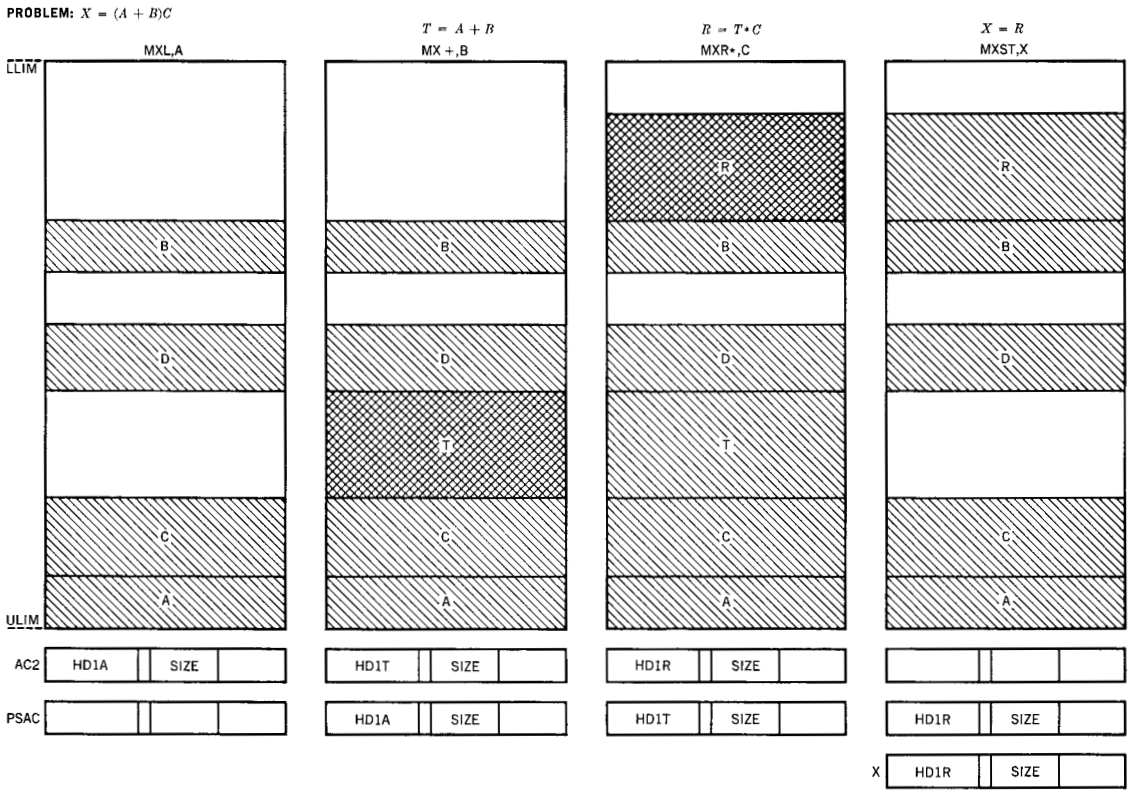
At RETURN, a decision is made whether or not to release the matrix whose DWD appears in PSAC. If this matrix is an unnamed intermediate result (i.e., has no DWD other than PSAC), it is released; otherwise, it is not released. The operands of the MX instruction just executed are then unflagged by setting bit 26 to 0 in HD1. Next, the DWD in AC2 is copied into PSAC, and the corresponding HD1 is updated to point to PSAC if and only if it had been pointing to AC2. (In the case of an MXL instruction, a copy of the DWD is put into AC2, but HD1 still points to the original DWA. At RETURN, AC2 is copied into PSAC, but HD1 is left pointing to DWA.) It should be noted that, in this case, HD1 will have already been flagged as a consequence of a request to MAP for assignment of space to AC2. Finally, the pseudo instruction counter within MXOP is updated, and the next instruction is decoded.

Table 13

Instruction
MXL,A
MX+,B
MXR*,C
MXST,X

Figure 14 illustrates the status of the POOL, AC2, and PSAC just prior to the transfer of AC2 to PSAC, following execution of the instructions given in Table 13.

Figure 14 Accumulator usage and storage allocation



Arithmetic subroutines

As previously mentioned, the subroutines MXL, MXLN, MXREL, MXST, and MXEND are contained within MXOP. These operations, which are simple and require no selection mechanism, are described first.

unary operations

MXL loads AC2 with a copy of the DWD of its addressed operand; it then goes to RETURN, which leaves the correct DWD in PSAC, and HD1 pointing to the original DWA.

MXLN requests MAP to assign space in POOL to AC2 for a copy of the addressed matrix; transmits HD2 unchanged to the new location followed by all the matrix elements with changed sign; and goes to RETURN, which leaves the correct DWD in PSAC, and HD1 pointing to PSAC.

MXREL zeros out DWD and releases the matrix area in POOL or DPOOL. In POOL, either HD1 and HD2 are replaced by WD1 and WD2 of a hole, or the released area is merged with adjacent hole(s). In DPOOL, the hole is added to the disk-hole list (in DSKLST) or merged with adjacent hole(s) therein. Instead of going to RETURN, MXREL updates the MXOP instruction counter.

MXST effects the release of the matrix (if any) pointed to by the addressed DWD. If PSAC refers to a named matrix (i.e., HD1 does not point back to PSAC), MAP is requested to assign a new

space for a duplicate of this matrix, which is then moved to the new space. Otherwise, PSAC is duplicated in the DWD, and HD1 is set to point back to this DWD. Finally, the MXST updates the MXOP instruction counter.

MXEND terminates the interpretive regime and branches to the next instruction.

The selection mechanism for the unary operations MXLT and MXLI is simply a branch vector to which an indexed branch is taken. The index value, equal to the TYPE code in HD2, selects the appropriate branch instruction in the branch vector, and this instruction branches directly to the desired subroutine. The functions of MXLT and MXLI are as follows:

MXLT requests MAP to assign space in POOL to AC2 for a copy of the addressed matrix. If TYPE $\neq$ COL, HD2 and all matrix elements are transmitted to the new location; if TYPE = COL, rows and columns are interchanged during transmission; and if TYPE = NULL or COL, the row and column numbers in HD2 are interchanged. Finally, MXLT goes to RETURN, which leaves the correct DWD in PSAC, and HD1 pointing to PSAC.

MXLI requests MAP to assign space in POOL to AC2 for the inverse matrix; requests space for $2N\ (N + 1)$ words of workspace if TYPE = SYM or COL; checks HD2 for TYPE $\neq$ NULL and ROW = COL; inverts SCAL and DIAG matrices by reciprocation; converts SYM matrix to COL, and then inverts by Gaussian elimination; releases workspace if used; and goes to RETURN, which leaves the correct DWD in PSAC, and HD1 pointing to PSAC.

binary operations

The selection mechanism for the binary operations MX+, MX−, MXN+, MXR*, MXL*, and MXL*I requires a branch table, since two TYPE codes are involved. This table consists of several columns, each of which is a branch vector headed by an indexed branch instruction. Using the *smaller* of the two operand TYPE codes, an indexed branch is executed to the head of the appropriate column in the branch table. The indexed branch instruction found there uses the TYPE code of the other operand to select the proper row in that column. In this location, a direct branch to the desired subroutine is executed.

In all six of these matrix instructions, a check of the dimensions of the two operands is made before executing the instruction; if the dimensions are not compatible, an error message is printed and the program stops.

The three instructions MX+, MX−, and MXN+ are handled by the same subroutine after the selection mechanism for addition has been used. This is accomplished as follows: after the TYPE code, SIZE, and dimensions of the result matrix have been computed, and a memory assignment has been obtained from MAP, the operand matrix having the *higher* TYPE code is moved into the new area, with or without a change of sign, as required by the operation. The other operand matrix is then processed by using add-to-memory (M+) or subtract-from-memory (M−) instructions. HD2 of the result, which corresponds to that of the

operand of higher TYPE code, is automatically set up just after the memory allocation has been made.

The instructions MXR* and MXL* are also handled by the same subroutine after the selection mechanism for multiply has been used. Again, the TYPE code, SIZE, and dimensions of the result matrix are automatically computed, and HD2 is properly set up by the chosen subroutine.

As previously mentioned, MXL*I is a contrived mnemonic code that permits use of a single address instruction to call a subroutine for solution of equations rather than for matrix inversion. Actually, the same subroutine is used for both MXL*I and MXLI, but in different ways. When MXL*I calls the subroutine, the number of columns M in the implied operand is used in requesting a temporary workspace of $(M + N)\ (N + 1)$ full words. This workspace is released after the computation has been completed.

In both MXL*I and MXLI, a singular matrix causes the printing of an error message, and then stops the program.

Remarks

The MARI program could be implemented without unusual difficulty on other computers that have sufficient random-access disk storage or bulk core memory. However, as is usual in system experiments, the special characteristics of the vehicle computer were exploited. Thus, several features of the 7030 are used to advantage by the program. For example, the multi-field structure of the 7030 index word permits the use of a single word (the directory word) to represent each matrix. This allows conventional indexing of matrices. Moreover, the 7030 machine instructions for refilling the index registers prove useful in double-threading the hole lists. Finally, the 16 unused floating-point operation codes provide a convenient vehicle for the various matrix pseudo operations.

The 7030 Execute Indirect and Count (EXIC) instruction offers an additional advantage. The MARI interpreter must not relinquish control within its domain, even in the case of interspersed machine instructions (i.e., non-matrix instructions). These instructions are handled under the aegis of the EXIC instruction, which lends control to the object instruction, and updates a pseudo instruction counter in the process. A successful branch instruction behaves like an escape attempt; such an attempt is defeated by a compulsory 7030 interrupt ("execute-exception") that restores control to the interpreter.

Summary

The experimental matrix computation program described in this paper embodies dynamic memory allocation of all matrices, and enables the user to specify most matrix operations by using single-address pseudo instructions. Several different matrix types are allowed, and the most common matrix operations (addition,

subtraction, right- and left-hand multiplication, transposition, inversion, solution of linear equations and eigenvalue/eigenvector computations) have been implemented. If desired, additional matrix types and/or matrix operations may be incorporated.

A matrix computation language, similar in structure to the 7030 symbolic language, has been developed. Matrix pseudo instructions can be interspersed with machine instructions; under control of the interpreter, both types of instructions are executed interpretively.

The memory allocation solution applies to disk as well as core memory. The algorithm used for memory allocation is not claimed to be optimal; as is well known, the efficiency of any such algorithm is highly problem dependent. However, the algorithm chosen is a reasonable approach to a variety of representative problems.

CITED REFERENCES AND FOOTNOTE

1. J. Cline, G. Slike, and K. Lantz, "General matrix abstraction," SHARE Library Number F1MBMTX1, Abstract Number 138, The Glenn L. Martin Company, Baltimore, Md., 1956.
2. W. Outten, "General matrix abstraction for tapes," SHARE Library Number F1MBMTX2, Abstract Number 367, The Glenn L. Martin Company, Baltimore, Md., 1957.
3. G. W. Armerding, "Matrix manipulating interpretive program for the 709," SHARE Library Number F1•LLFMMIP, Abstract Number 936, MIT Lincoln Laboratory, Bedford, Mass., 1959.
4. H. M. Gladney (Princeton University) and Mrs. C. M. Kimme, "Matrix package for use with IBM FORTRAN monitor," SHARE Library Number FOXBE MAT, Abstract Number 1497, Bell Telephone Laboratories, Murray Hill, N. J., 1963.
5. R. A. Hoodes, R. E. Bargmann, T. A. Huck, L. W. Paul, "Statistically Oriented Matrix Programming System (STORM)," International Business Machines Corporation, Bethesda, Md.
6. IBM Reference Manual, 7030 Data Processing System, Master Control Program, C22-6678-1, International Business Machines Corporation.
7. IBM Reference Manual, STRAP-II 7030 Assembly Program, C28-6129, International Business Machines Corporation.
8. IBM Reference Manual, 7030 Data Processing System, A22-6530-2, International Business Machines Corporation.
9. W. Buchholz, editor, *Planning A Computer System*, McGraw-Hill Book Company, New York, 1962.
10. J. K. Iliffe and J. G. Jodeit, "A dynamic memory allocation scheme," *The Computer Journal* 5, 200–209 (1962). The scheme described in this article is similar in many respects to the one developed for MARI; it includes both indirect addressing, via a "codeword," and the ability to specify matrix operations as single-addressed instructions.

Symbol key

AC2	auxiliary pseudo accumulator
CF	count field
COLS	number of columns of matrix
DATA	data area outside the pool
DISKMIN	minimum size of matrix to be transmitted to DPOOL

Symbol key (Cont'd.)

DPOOL	extension of POOL in disk memory
DSKTOT	total space available in DPOOL
DWA	directory word address
DWD	directory word
DWDLNK	contains address of first listed matrix in DPOOL
FF	flag field
HD	header word
HOLL1	LLIM terminus of hole location chain
HOLLOC	ULIM terminus of hole location chain
HOLS1	largest hole terminus of hole size chain
HOLSIZ	smallest hole terminus of hole size chain
HTOTAL	total space available in POOL
LLIM	lower memory address of POOL
LSTHO	contains (1) address of first entry in disk list and (2) number of disk holes
MAP	memory allocation program
MARI	name of the matrix arithmetic interpreter described
MCP	IBM 7030 master control program
MXOP	subroutine for matrix operations
POOL	a region of core memory for matrix storage
PSAC	a DWD used as a pseudo accumulator
RESERV	disk storage area used as workspace
RF	refill field
ROWS	number of rows of matrix
SIZE	size of matrix or hole
TYPE	matrix type code
ULIM	upper memory address of POOL
VF	value field
WD	word
XF	index flag
XW	index word