MATMATH — A Matrix Handling Package for Forth

Denise S. (Derby) Stilling and L. Glen Watson

Department of Mechanical Engineering
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 0W0

Abstract

This paper discusses a matrix mathematics package which illustrates the utility of Forth for numerical analysis. Since matrices are the primary building block for much of the numerical analysis used in present day science and engineering, the package was developed to be suitable for a variety of applications. The code presented assumes the presence of the proposed FVG Standard floating point extensions **[DUNC84]** and follows the Forth-83 Standard **[FORT83]**. The package provides for the creation, initialization and fundamental manipulations of matrices. To demonstrate the utility of the package, two applications have been presented.

Introduction

In many branches of engineering, numerical analysis is carried out almost exclusively in FORTRAN. There have been suggestions that this should not be the case and Forth has been recommended as a viable replacement. For example, Noble **[NOBL88]**, **[NOBL89]** states that Forth is fast, compact and easy to program, debug and maintain. In addition, Forth efficiently exploits available resources as demonstrated by MacIntyre **[MACI84]** whose example uses Forth on a PC to solve a problem which others have studied using FORTRAN on a Cray. Watson and Stilling **[WATS90]** support Noble's arguments and stress the reusability of Forth code. In this paper, a matrix mathematics package is presented in hopes of enhancing the popularity of Forth for applied scientific computations. The authors were motivated to develop such a package when programming a finite element simulation and a neural network controller **[STIL90]**. Special effort was taken to design a serviceable utility package with extensible, re-usable routines.

The package includes: words to create matrices, words to interrogate matrices as to their dimensions or element size, words to designate specific matrices as the "active" arguments in various matrix operations, words to facilitate input and output of matrix elements, words for calculating various norms of matrices, words to perform matrix addition, subtraction, multiplication, inversion and scaling, and words to produce a duplicate or transposed copy of a matrix.

As a comprehensive utility package, the selected matrix operators encompass those encountered in an introductory linear algebra course and those required in engineering design and

analysis. Although the set of operators is not exhaustive, supplemental functions can be derived by combining the defined operations or by modifying the source code in Appendix A **[WATS91a]**.

The operators have been defined using the Forth-83 Standard **[FORT83]** word-set; the authors have attempted to denote or define any anomalous words. This creates transportable and comprehensible code in contrast to machine-dependent versions such as that of Ruzinsky **[RUZI83a&b]**. Since in most engineering applications, computations are performed on the set of real numbers, this package was limited to floating point arithmetic. Presumably, the user's Forth is equipped with floating point words which follow the proposed FVG standard floating point extension **[DUNC84]**. Floating point arithmetic tends to be more functional than integer arithmetic since scaling procedures become impractical when the magnitudes of the calculated numbers are unknown.

The programming approach possesses a unique degree of flexibility. In performing any matrix manipulation, the operation internally adjusts to the size of its operand(s). That is, the operands themselves impose the necessary size constraints for the routines. Thus, reusable, generic routines can be defined where fewer parameters are required as arguments to the operators.

Matrix Mathematical Package

The matrix mathematical package contains a variety of operators and utilities. The cardinal function, **MATRIX**, creates the data structure. Supplemental operators are provided to access the parameters characterizing the structure. Another set of routines perform input and output of the elements of the matrix; including routines to define a matrix as a null or an identity matrix, to initialize all elements to a fixed value or to fill the matrix from values on the stack. Matrix transpose and copy facilities that include matrix partitioning and assembling routines are available. Algebraic operators, such as: scalar multiplication, matrix multiplication, matrix inverse, addition and subtraction are provided. Also included are matrix-specific operators, for calculating: determinants and norms and for performing LU decomposition. Operators for creating temporary or scratch space have also been provided.

Defining the MatrixData Structure

In Forth, data structures, such as matrices, can be defined using the **CREATE ... DOES>** construct **[BASI83]**. With this construct, information characterizing the structure along with code which operates on this information can be encapsulated. There is no consensus as to the amount of extra information to be included in the data structure. MacIntyre, for one, questions the degree of "intelligence" that data structures should possess. He suggests that parameters such as the number of dimensions, data type, address type and storage byte-width should be included in each array header **[MACI86]**. This issue is avoided when application-specific code is developed; usually, matrices of fixed dimensions are employed **[STOD85]**, **[LAQU88]**.

Since most problems encountered in engineering can be formulated using vectors or two dimensional arrays, this package has been so restricted. Each matrix is an entity that provides self-indexing for addressing or accessing its contents. During the **CREATE** phase, a header containing the matrix name, its dimensions and element storage size plus an appropriate amount of data storage space is placed in the dictionary. At execution time, the **DOES>** phase returns the address of the specified element location. This defining word is **MATRIX** whose usage follows:

number_of_rows number_of_columns **MATRIX** <matrix.name>

Row and column matrices, i.e. vectors, are defined with **MATRIX** by setting either the corresponding row or column value to one. This implementation of **MATRIX** assumes a zero based index system; i.e. the first element is stored at (0,0).

The element size, which is stored in a self-fetching variable, **#BYTES**, is coded to default to **FPSIZE**, the byte-width of a floating point value. The space allotted to <matrix.name> is based on the element size and the number of elements as specified by the number of rows and columns. Using a self-fetching variable accommodates for various stack widths. Non-standard element sizes can be declared prior to matrix creation with the command:

```
size_in_bytes EQU #BYTES
```

Note that most 32-bit Forth versions use 4 bytes for integer values and 4 bytes for double precision values; whereas, 16-bit Forth versions use 2 bytes for integer values and 4 bytes for double precision values. As denoted in the source code, the location or address of an element can be obtained by specifying:

```
row_number column_number <matrix.name>
```

Brodie suggested this syntax **[BROD87]** which has been followed in various application-specific codes **[RUZI83a], [BARN84], [CARP88], [NEUB90].**

Operators for accessing the parameters that characterize a matrix have been provided. The number of rows of a matrix can be obtained by using the following command:

```
?#ROWS  <matrix.name>.
```

Similarly, the number of columns of a matrix can be accessed by:

```
?#COLUMNS  <matrix.name>
```

and the storage width of each element can be discerned using the command:

```
?#BYTES  <matrix.name>.
```

When defining block operators for matrix manipulation, it is essential to realize that each matrix stores its contents contiguously, row by row. Data storage by rows is well-suited for many matrix operators; especially, those whose algorithms employ row exchanges. When coding algorithms, Forth words for memory transfer can be used advantageously.

Creating Operators

To create generic, reusable operators, the arguments of the operators were vectored. This vector execution philosophy enables operators to be coded for a matrix structure; that is, each operator can be associated with any matrix without requiring any modification to the operation. In this package, matrix operations which require one, two or three operands have been defined.

Assigning operands to a routine is accomplished using the state-dependent words, **UNI_MAT, BI_MAT** and **TRI_MAT**. In assigning a matrix to be the "active" argument for an operation, the code field address (cfa) of the designated matrix is stored in a variable. In order to perform matrix data manipulation, the code pointed to by the variable must be executed.

Assignments of matrices are made to the variables **LMATRIX** (left-hand matrix), **RMATRIX** (right-hand matrix) and **SMATRIX** (solution matrix). For example, operators involving three matrices are prefaced with the command **TRI_MAT**. The command **TRI_MAT A B C**, stores the cfa of matrix A in the variable, **LMATRIX**, the cfa of matrix B in the variable, **RMATRIX** and the cfa of matrix C in the variable, **SMATRIX**. Similarly, two argument matrix operations employ the statement, **BI_MAT A B**, which places the cfa of matrix A in **LMATRIX** and places the cfa

of matrix B in **RMATRIX**. Lastly, **UNI_MAT** is used to preface single argument operations and stores the cfa of the matrix in **LMATRIX**.

Accessory words which return the names of the currently active matrices and their related characteristics have been provided. **LMAT.NAME**, **RMAT.NAME** and **SMAT.NAME**; **L#ROWS**, **R#ROWS** and **S#ROWS**; and **L#COLS**, **R#COLS** and **S#COLS** return the corresponding name, number of rows or number of columns of the assigned left-hand, right-hand or solution matrix, respectively. The operators **LM@**, **RM@** and **SM@** perform data retrieval and **LM!**, **RM!** and **SM!** provide data storage based on the above vectoring. These operators are the core of the matrix operator words.

Matrix Operators

A brief description of the operators that are coded in high level Forth follows. An effort has been made to adhere to Forth 83 standard word-set **[FORT83]**. The corresponding stack effects and appropriate syntax for each operator are also given. Note that "f" refers to a floating point value and "m" and "n" are integer values. Correct matrix dimensioning has also been assumed.

Matrix Input/Output

ELEMENT DATA STORAGE

A floating point value on the stack is stored in the nth row and mth column of matrix A:

```
f n m A F!
```

ELEMENT DATA RETRIEVAL

A floating point value taken from the nth row and mth column of matrix A is placed on the stack:

```
n m A F@
```

MATRIX DISPLAY

.MAT (--)

Prints the contents of the matrix in formatted form.

Usage: **UNI_MAT A .MAT**

Interactive Matrix Interrogation Words

?DIMENSIONS (--)

Displays the characteristic parameters of the active matrix.

Usage: **UNI_MAT A ?DIMENSIONS**

?#COLUMNS (-- n)

Returns the number of columns in the matrix which follows the operator.

Usage: **?#COLUMNS A**

?#ROWS (-- n)

Returns the number of rows in the matrix which follows the operator.

Usage: **?#ROWS A**

?#BYTES (-- n)
Returns the number of bytes per element in the matrix which follows the operator.
Usage: **?#BYTES A**

Matrix Initialization

STACK->MAT (f_0 . . .f_{nm} --)
A matrix is initialized using the values on the stack, hence the number of elements on the stack corresponds to the number of elements in the matrix. Values entered on the stack are stored by rows, beginning with the (0,0) element. Warning: the stack depth determines the permissible number of elements that can be entered.
Usage: **UNI_MAT A STACK->MAT**

DIAGONAL (--)
Zeros all non-diagonal elements of a matrix. In this context, the non-diagonal elements are assumed to be any cell where the row and column indices are not equivalent.
Usage: **UNI_MAT A DIAGONAL**

MZERO (--)
Creates a null matrix by initializing all elements to zero.
Usage: **UNI_MAT A MZERO**

MFILL (f --)
Sets each element in the matrix to the floating point value on the stack .
Usage: **UNI_MAT A MFILL**

IDENTITY (--)
If the active argument is a square matrix an identity matrix is created; otherwise a Kronecker Delta function is performed using the matrix indices.
Usage: **UNI_MAT A IDENTITY**

Algebraic Matrix Operations

MPLUS (--)
Performs the matrix addition: A + B = C.
Usage: **TRI_MAT A B C MPLUS**

MMINUS (--)
Performs the matrix subtraction: A - B = C.
Usage: **TRI_MAT A B C MMINUS**

MMULT (--)
Performs the matrix multiplication: A * B = C.
Usage: **TRI_MAT A B C MMULT**

SMULT (--)

Performs a scalar multiplication on the matrix; the scalar multiplier is stored in a floating point variable, **SVAR**.

Usage: **TRI_MAT SVAR A C SMULT**

INVERSE (--)

Inverts a square matrix in place.

Usage: **UNI_MAT A INVERSE**

SOLVE (--)

Used for solving a set of linear equations. In the form, A X = B, where A is the coefficient matrix (square matrix) and B is the constant matrix (given vector of a single set of forcing functions). The solution matrix, X is placed in matrix, B.

Usage: **BI_MAT A B SOLVE**

Other Operators

DETERMINANT (-- f)

Calculates the determinant of A.

Usage: **UNI_MAT A DETERMINANT**

LUDECOMP (--)

Performs a lower-upper triangulation using Crout's method; the result is a LU decomposed rowwise permutation of the original matrix.

Usage: **UNI_MAT A LUDECOMP**

Matrix Norms

ROW-NORM (-- f)

Determines the row norm which is defined as the largest sum of the absolute values of the elements in a row.

Usage: **UNI_MAT A ROW-NORM**

COL-NORM (-- f)

Determines the column norm which is defined as the largest sum of the absolute values of the elements in a column.

Usage: **UNI_MAT A COL-NORM**

E-NORM (-- f)

Determines the Euclidian norm of a matrix.

Usage: **UNI_MAT A E-NORM**

Copying and Partitioning Matrices

MCOPY (--)
Copies the contents of one matrix to another.
Usage: **BI_MAT A B MCOPY**

TRANS_COPY (--)
Copies the transpose of one matrix to another; the first matrix is left intact.
Usage: **BI_MAT A B TRANS_COPY**

MINSERT (n m --)
Replaces the elements of the second matrix with those of the first matrix. The upper left hand element of the first matrix is matched to the *nth, mth* element of the second matrix. Replacement is done by rows.
Usage: **BI_MAT A B MINSERT**

MEXTRACT (n m --)
The second matrix is a partition of the first matrix. The (0, 0) element of the second matrix is matched to the *nth, mth* element of the first matrix.
Usage: **BI_MAT A B MEXTRACT**

Dynamic Memory Management Operators

TEMP_MATRIX (--)
Creates the framework of a temporary data structure for the storage of characterizing parameters which will include the number of columns, the number of rows, the byte-width and the start of the base address of the data storage space. Since these structures are similar to those created using MATRIX, the previously described matrix operators can be used.
Usage: **TEMP_MATRIX A**

TEMP_ALLOT (n m --)
Sets the characterizing parameters of the temporary data structure (*n* number of rows, *m* number or columns, current value of **#BYTES** for byte-width and sets the base address to the current end of the dictionary). Then the required data space is allocated past the end of the dictionary.
Usage: **UNI_MAT A TEMP_ALLOT**

TEMP_DEALLOT (--)
Resets the characterizing parameters of the temporary data structure and frees the data storage memory.
Usage: **UNI_MAT A TEMP_DEALLOT**

These operators are integral to algorithms that require work space (i.e. **DETERMINANT, SOLVE, LUDECOMP** and **INVERSE**). After creating (**TEMP_MATRIX**) and allocating space (**TEMP_ALLOT**), the data structure can be manipulated with the matrix operators. Retrieving the

data storage space (**TEMP_DEALLOT**) must be completed prior to subsequent creation of matrices or definitions.

A full description of each algorithm can be found in various sources on numerical analysis. Most algorithms used in this package have been adapted from those presented in *Numerical Recipes* **[PRES87]**, *Handbook of Algorithms and Data Structures* **[GONN84]** and *HP-15C Advanced Functions Handbook* **[HPC82]**.

Anomalies to Forth-83 Standard

The use of nonstandard words has been minimized. Those that have been incorporated into the code are explained to improve the code's comprehensibility and portability. To assist with vector execution, the nonstandard word, **PERFORM** has been used **[HAYD90]**. This word executes the code stored at the address on the stack. The high level Forth word **EQU** **[LMI86]** which is a "changeable constant" has been used in defining the matrix element cell-size. A similar implementation of this structure, called **VALUE**, appears in Lewis's work **[LEWI85]**. The routines to identify and describe active arguments use **.NAME**. The function of this LMI extension **[LMI87]** is to print the name corresponding to the given name field address.

As previously mentioned, data manipulation uses floating point operators as proposed in the FVG Standard. The unique implementation of **MZERO** takes advantage of the way floating point numbers are stored in LMI-FORTH+. The two cell definition of a floating point number can be set to zero by setting each cell to zero. Thus an efficient method to create a null matrix is to fill the data space with zeros. An alternative definition has been provided to ensure compatibility.

One of the employed nonstandard words **S>F** converts a single precision integer to a floating point number. An equivalent high level definition would combine **S>D** **[TRAC87b]** and the FVG Standard word, **FLOAT**. The floating point logical operator **F0<>** is merely a combination of the logic operators **F0=** and **NOT**.

Forth Coding Techniques

In developing software, two widely accepted, programming objectives are readability and efficiency. Special efforts have been made to enhance the readability of the source code. This is particularly important because as noted by Stoddard "well understood operations, such as matrix multiplication are quite tricky to express in Forth, and when coded bear little resemblance to a text book description of the underlying algorithm" **[STOD85]**. As well, the algorithms have been reformulated to enhance efficiency.

One weakness of Forth lies in its lack of support for loops nested to an arbitrary level. According to the Forth-83 standard **[FORT83]**, loop indices are recoverable to only two levels. As well, the loop accessing utilities are relative to loop nesting rather than absolute referencing. The nesting of more than two layers requires either clever stack manipulation or a kludge such as using variables with mnemonic names like **EYE**, **JAY** or **KAY**, to store the indices. This latter approach has been adopted where such nesting is required, as in **MMULT** **[WATS90]**.

Forth is adept at memory handling. Transferring blocks of memory using **CMOVE** has been used extensively in algorithms requiring row interchanges. Alternately, element-by-element transfers may be implemented; however run-time efficiency tends to be decreased. Similarly, initialization routines employ **FILL** to operate more efficiently.

The vector execution technique utilized in this package provides greater flexibility than was previously indicated. The operator, **SMULT**, references a variable rather than another matrix as discussed in the definition of **TRI_MAT**.

Some applications benefit greatly from dynamic memory management. Many Forths provide this by accessing the host operating system; unfortunately, this is not standard, resulting in nontransportable code. Still other vendors provide no such support. For this reason, a high level, dynamic management scheme has been employed. Rather than creating work spaces which restrict the size of the arguments being passed to an operation, temporary data structures are created. This is accomplished by creating temporary, matrices. In complex operators, at run-time characterizing parameters are stored and the required data space for intermediate storage and computations is allocated. Subsequently, the operations are terminated by recovering the allotted data space. Such dynamic allocation is an integral part of the **DETERMINANT**, **LUDECOMP**, **SOLVE** and **INVERSE** operators. The syntax for creating and data manipulation of these temporary matrices is consistent with the rest of the **MATMATH** package. The other matrix operations can be performed on the temporary matrices. A drawback of this technique is that the framework of the temporary data structure remains in the dictionary after liberating the allocated data storage space. But this skeleton can be revitalized; that is, "new" information to create scratch space with different characteristics can be allocated. Unfortunately, this dynamic memory management demands allocation of space prior to usage and deallocation of this space prior to defining further application code or additional data structures. (Refer to Appendix A, screens 36 and 37 for a sample implementation).

Future Developments

The authors found the package easy to use and it contained all of the matrix operations needed in their present work **[STIL90]**. Undoubtedly, some other users may feel the scope of this package needs to be extended. Possible areas for improvement include error handling, mixed data structure handling, or additional matrix operators and speed or size optimization of the code **[WATS91a]**.

Error handling routines tend to assist in creating a user friendly package, often at the expense of run-time efficiency. In this package profuse error handling and bounds checking routines are limited to detecting singularities. The detection of a singularity leads to aborting LU decomposition operations which are integral to inversion, finding determinants and linear equation solving. Further routines could be added for checking compatibility in element storage size or matrix bounds checking as required by the operators **[WATS91a]**.

Some applications may require mixed data structures. Hence operations should be defined so as to perform the appropriate type of manipulation; this is most likely to present problems if there is no separate floating point stack. Perhaps the routines could be re-coded for the various data types and special conversion operators defined **[WATS91b]**. Also, the technique of operator overloading **[STRO87]** may be applied. By defining primitives for integer, double precision, floating point and complex arithmetic operations, the matrix structure and operators can be reformatted to correctly access the appropriate algebraic operator **[WATS91b]**.

Without any doubt, there will be user applications which require operations which cannot be cobbled from the existing operators. In some cases, a more efficient set of operators based on algorithms specific to the user's problem may prove more efficacious than the general purpose routines that were used. The Forth-83 standard word-set **[FORT83]** has been followed closely to enhance portability and understandability. Unfortunately, this fails to tap Forth's forte, accessing machine dependent routines, which often reduces execution time. For applications where speed is a priority, the usual techniques for optimizing code such as reformulating algorithms, redefining inner loops in machine code and in-line programming can be followed.

Another concern is space optimization. This can be achieved by better factoring of code or by loading only the operations required. The source code (Appendix A) has been documented to facilitate such optimization.

Applications

Linear algebra spans many domains of scientific application. For instance, the solving of simultaneous, independent equations is part of circuit analysis, structural design and control applications. As illustrated in Figure 1, the current flow through each loop can be calculated by applying Kirchoff's Laws **[BOBR85]**.

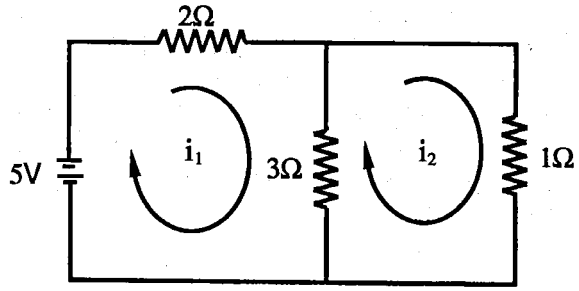

Figure 1.
Simple electrical circuit

The equations for each loop are

$5 = 2i_1 + 3(i_1 - i_2)$, (1)

and

$0 = -3i_1 + 4i_2$, (2)

After solving the above equations, the curents are found to be

$i_1 = 1\ ^4/_{11}$A

and

$i_2 = 1\ ^9/_{11}$ A.

The solution of this set of linear equations can be computed from the following code:

```
\ Creating appropriate matrices
2 2 MATRIX COEF  2 1 MATRIX VECTOR

\ Initializing of the matrices
5.0E0 -3.0E0 -3.0E0 4.0E0   UNI_MAT   COEF     STACK->MAT
5.0E0 1.0E0                 UNI_MAT   VECTOR   STACK->MAT
```

```
\ Solving the equation
BI_MAT          COEF        VECTOR        SOLVE

\ Output of the solution matrix
.( The loop currents are ) CR   UNI_MAT  VECTOR     .MAT
```

Another exposition of the versatility of this package follows in the use of linear algebra to solve an algebraic equation. The algebraic equation (3) performs a unique scaling of the elements in matrix, C. The following code incorporates the special operators of the matrix package to solve the equation.

$$d_{ij} = (1 - \eta\, a_{ii}\, b_{1i})\, c_{ij} \qquad (3)$$

```
\ Declaration of matrices
5 5 MATRIX A     5 1 MATRIX B        1 5 MATRIX C  5 5 MATRIX D
5 1 MATRIX ONE   5 1 MATRIX TEMP1    5 1 MATRIX TEMP2

\ Defining the scalar
FVARIABLE     ETA     -1.0E0     ETA     F!

\ Initialization of matrices
1.0E0  UNI_MAT  ONE  MFILL

\ Implementation of equations assume other matrices have been initialized
UNI_MAT   A                          DIAGONAL
TRI_MAT   A       B       TEMP1      MMULT
TRI_MAT   TEMP1   ETA     TEMP2      SMULT
TRI_MAT   ONE     TEMP2   TEMP1      MMINUS
TRI_MAT   TEMP1   C       D          MMULT

\ Output Solution
.( The solution is ) CR  UNI_MAT  D  .MAT
```

Conclusions

This mathematical package for linear algebra illustrates the dexterity of Forth for numerical computations. As presented, this utility package has attempted to attain a high degree of portability. Hopefully, this will ease transferring of the program to a variety of machines and encourage the use of Forth for numerical analysis.

The authors hope that this package may serve as a first draft of a standard Forth linear algebra utility. Similar resource packages **[STAR87], [MOSA90]** are available for matrix handling; unfortunately, most of these packages are protected by proprietary rights and implementation details are usually withheld, resulting in numerous re-creations of the same resource. By making

the code accessible development time for similar problems should be reduced. The authors encourage others to use this code for personal, non-commercial purposes and invite feedback on the package.

The original and innovative nature associated with creating generic operators lies with the vectoring technique. This technique requires the use of a quasi-prefix syntax in the commands of **UNI_MAT**, **BI_MAT** and **TRI_MAT**. Some of the merits of this technique include readable application source code, the ability to liberate state dependency in assigning operands, ease in extending the matrix operator library and streamlining of application code where repetitive use of the same matrix operator occurs. Alternately, assignment of the matrix arguments using the Forth commands **['] ** or **'** would be viable. Drawbacks of this technique for the user include remembering the required state syntax and the need to declare arguments for each use although the matrix operand has not changed. When attempting to implement the matrix package so that the matrix operators use postfix syntax, stack manipulation within the defined operators tended to be more involved and the apparent need to parse the input stream in both states added to the programming complexity. The authors did not investigate all possibilities for implementing generic matrix operators. However, from our extensive use, **MATMATH** appears to be easy to use and practical. The nature of the employed, quasi-prefix syntax is offered as an issue for debate in the Forth community.

Using Forth's extensibility, operators that cannot be synthesized from the existing code can be created by modifying the code.

We have found that if the syntax of **MATMATH** serves as a standard, then modular programming with mathematical routines written by more than one individual can be easily interfaced and improved program development results.

Some of the unique programming techniques used in **MATMATH** are relevant to other applications. The vectoring of more than one argument enables operands to be exchanged among routines and improves the readability and maintainability of routines. The dynamic memory management scheme was created to avoid imposing size constraints on arguments when space for intermediate steps was necessary. Although this high level dynamic allocation scheme requires the immediate freeing of memory, routines that require a large amount of space for a brief interval, may use this technique. Alternately, a rudimentary, dynamic memory management scheme could have employed **EVAL** [TRAC87a&b] and **FORGET**. **EVAL**, a nonstandard command, enables strings to be interpreted. Scratch space can be created at run-time if the strings contain the commands for creating matrices. The dictionary space is not wasted, since **FORGET** cleanly frees the dictionary. This technique is extensible to other temporary operations. However, implementation requires the nonstandard commands of **EVAL** [TRAC87a&b] and **"** [HAYD90]. As experienced and as noted by a reviewer, this technique is problematic when creating an embedded system where the text interpreter is shed.

MATMATH has been created as a powerful and portable utility package for linear algebra. Through the use of generic, reusable operators and consistent syntax, compatibility and standardization is achieved. The authors encourage others to improve the utility of this package to assist in realizing the potential Forth has for applied scientific computation.

Acknowledgement

The comments of the reviewers have enhanced the presentation and content of this paper. A special tribute to the reviewer who suggested the basis of the discussed dynamic memory management technique. Partial financial assistance was provided by the Natural Sciences and Engineering Research Council of Canada (0GP0005575). Also gratefully appreciated was the

support provided from Zonta International Foundation in the form of the Amelia Earhart Fellowship to the first author.

References

[BARN84] Joe Barnhart, "Forth and the Fast Fourier Transform", *Dr. Dobb's Journal*, 19(9):34ff, 1984.

[BASI83] James Basile, "Multi-Dimension Arrays", *JFAR*, 1(2):79-80, 1983.

[BOBR85] Leonard S. Bobrow, *Fundamentals of Electrical Engineering*, Holt, Rinehart and Winston, Inc.: New York, 1985.

[BRAD86] Mitch Bradley, *Forthmacs User Guide*, Bradley Fothware: Mountain View, California, 1986.

[BROD87] Leo Brodie and Forth Inc. *Starting Forth*, pp. 267-8, Prentice-Hall, Inc.: Englewood Cliff, New Jersey, 1987.

[CARP88] John D. Carpenter, "Trainable Neural Nets in Forth", *1987 FORML Conference Proceedings*, pp. 415-434, 1988.

[DUNC84] Ray Duncan and Martin Tracy, "The FVG Standard Floating Point Extension", *Dr. Dobb's Journal*, 9(9):110-115, 1984.

[FORT83] Forth Standards Team, *Forth-83 Standard*, Mountain View Press: Mountain View, California, 1983.

[GONN84] G. H. Gonnet. *Handbook of Algorithms and Data Structures.* Addison-Wesley Publishing Co.: Reading, Massachusetts, 1984.

[HAYD90] Glen Haydon, *All About Forth — An Annotated Glossary*, 3rd Edition, MVP-Forth Series Moutain View Press: Mountain View California, 1990.

[HPC82] Hewlett-Packard Company, *HP-15C Advanced Functions Handbook,* (00015-90011), Corvallis, OR, pp. 96ff, 1982.

[LAQU88] Robert E. La Quey, "Networks of Neurons", *1987 FORML Conference Proceedings*, pp. 191-200, 1988.

[LEWI85] Steven M. Lewis, "Should VARIABLE be an Immediate State-sensitive Word?", *JFAR*, 3(1):53-60, 1985.

[LMI86] *LMI FORTH USER NEWSLETTER*, pp.5-6, Laboratory Microsystems, Inc. California, Aug. 1986.

[LMI87] LMI, *PC/FORTH+ User's Manual*, Laboratory Microsystems, Inc.: California, 1987.

[MACI86] Ferren MacIntyre, "Forth Advanced Scientific Tools: How Intelligent Should Arrays Be?", *Proceedings of 1986 Rochester Forth Conference*, published in *JFAR*, 4(2):335-338.

[MACI84] Ferren MacIntyre, "Number Crunching with 8087 FQUANs: The Mie Equations", *JFAR*, 2(3):51-62, 1984.

[MOSA90] Mosaic Industries, Inc. *The RealFORTH Card* (brochure). 1990

[NEUB90] Karl-Deitriech Neubert, "Little Universe: A Self-referencing State Table", *JFAR*, 6(2):117-130, 1990.

[NOBL88] J. V. Noble, "Fortran is Dead: Long Live Forth!", *JFAR*, 5(2):261-270, 1988.

[NOBL89] J.V. Noble, "Scientific Computations in Forth", *Computers in Physics*, 3(5):31-38, 1989.

[PRES87] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling, *Numerical Recipes The Art of Scientific Computing*, Cambridge University Press: New York, 1987.

[RUZI83a] Steven A. Ruzinsky, "Fast Matrix Operations in Forth, Part I", *Dr. Dobb's Journal*, 8(6):56ff, 1983.

[RUZI83b] Steven A. Ruzinsky, "Fast Matrix Operations in Forth, Part II", *Dr. Dobb's Journal*, 8(7):70ff, 1983.

[STAR87] Mike K. Starling, "MATH.DOC", *4XFORTH™* (Disk update), Version 3.03, 1987.

[STIL90] Denise S. (Derby) Stilling and L. Glen Watson, "Neural Net Control of Intelligent Structures", *Proceedings of the 1990 Rochester Forth Conference* , pp. 139-141, June, 1990.

[STOD85] Bill Stoddart, "Readable and Efficient Parameter Access Via Argument Records", *JFAR*, 3(1):61ff., 1985.

[STRO87] Bjarne Stroustrup, *The C++ Programming Language*, Addison-Wesley Publishing Co., Reading, Massachussetts, 1987.

[TRAC87a] Martin Tracy, "A Forth Standard Prelude", *Dr. Dobb's Journal*, 12(10):40-45, 1987.

[TRAC87b] Martin Tracy, "The Forth Column", *Dr Dobb's Journal, 12(12):144ff., 1987.*

[WATS90] L. Glen Watson and Denise S. (Derby) Stilling, "Number Crunching With Forth", *Proceedings of the 1990 Rochester Forth Conference*, pp. 152-153, June, 1990.

[WATS91a] L. Glen Watson and Denise S. (Derby) Stilling, "Experience with Linear Algebra and the Forth Utility Package, MATMATH", *Proceedings of the 1991 Rochester Forth Conference*, pp. 129-131, June,1991

[WATS91b] L. Glen Watson and Denise S. (Derby) Stilling, *MATMATH: A Linear Algebra Package User's Guide*, 1991.

Appendix A: SOURCE CODE

The program was developed using LMI FORTH+ with special effort to adhere to the Forth-83 standard **[FORT83]** word-set. The authors believe that all non-standard words have been either noted or redefined. To test the transportability of the code, the source code was downloaded to another Forth vendor **[BRAD86]** with the only changes required being the implementation of **VALUE [LEWI85]** for **EQU** and adhering to the vendor's floating point syntax.

The source code does not depend on the text interpreter being either case sensitive or case insensitive. However, when compiled using a case sensitive Forth, the appropriate case of the Forth primitives must be used. Note that the following source code assumes Forth primitives are in upper case. For simplicity, the matrix operators are also defined using upper case.

Comments have been interspersed to indicate usage and to depict stack effects. Also the required screens for each function have been listed to assist with customizing the code for desired operators when space optimization is crucial. The space occupied by the entire code is about 5 Kbytes.

```
\ SCREEN 0
\ MATH UTILITIES                                        D&G  09/14/90

*****************************************************************
*****************************************************************
**                                                             **
**            MATMATH - A LINEAR ALGEBRA PACKAGE               **
**                                                             **
**     Copyright 1990 by L. Glen Watson & D.S.D. Stilling      **
**                University of Saskatchewan                   **
**                                                             **
**            All commercial rights are reserved.              **
**                                                             **
**                                                             **
*****************************************************************
*****************************************************************


\ SCREEN 1
\ MATMATH -  Anomolies to FORTH-83                  D&G  09/17/90
\ Screens 1 and 2 contain required non-standard extensions
\\ This causes the rest of the screen to be treated as comments.

\ Assumes existence of floating point package that follows
\ FVG Standard [DUNC74].

: F0<>  ( f -- flag )
        \ true flag if floating point number not equal to 0
   F0= NOT  ;

: S>F  ( n -- f)
       \ converts single precision number to floating point #
   S>D FLOAT ;

\ References: [LMI86] [TRAC87b]


\ SCREEN 2
\ MATMATH -  Anomolies to FORTH-83 (contd.)          D&G  09/17/90
\\

\ .NAME ( Prints the header or name for a given nfa ) [LMI86]

\ EQU ( a changeable constant ) [LMI86]
\    implementation similar to default action of VALUE [LEWI85]

\ PERFORM  ( executes a definition; cfa is stored in a variable)
: PERFORM   ( addr -- )                \ [HAYD90]
   @ EXECUTE ;


\ SCREEN 3
\ MATMATH - Matrix Parameter Access Words            D&G  09/17/90

 I8087      \ Loads INTEL 8087 floating point routines
            \ for LMI FORTH+ [LMI86]

4 CONSTANT WSIZE
            \ size of single precision value in LMI FORTH+

8 CONSTANT FPSIZE
            \ size of a floating point value in LMI FORTH+

FPSIZE EQU #BYTES
            \ initialize #BYTES to fpsize

\ The augmented base dictionary is assumed to be comprised
\ of the words on these screens (1 to 3).

\ SCREEN 4
\ MATMATH - Matrix Declaration                       D&G  09/17/90

: MATRIX    \ Usage:  #rows #columns  MATRIX TEST
            \ Matrix structure defining word
    CREATE ( #rows, #columns ---  )
        2DUP , , #BYTES DUP , * * ALLOT
    DOES> ( row , column --- addr )
        >R R@ @ ROT * + R@ WSIZE 2* + @ * R> WSIZE 3 * + +
;
```

```
\ SCREEN 5
\ MATMATH- Interrogation of Matrix Parameter      D&G  09/21/90

: ?#COLUMNS ( --- n )          \ Usage: ?#COLUMNS A
    ' >BODY @ ;                \  returns # of columns in A

: ?#ROWS    ( --- n )          \ Usage: ?#ROWS A
    ' >BODY WSIZE  + @  ;      \  returns # rows in A

: ?#BYTES   ( --- n )          \ Usage: ?#BYTES A
    ' >BODY WSIZE 2* + @  ; \  returns # of bytes allotted for
                               \  each element entry in Matrix A

\ Useful for iteractively determining matrix parameters
\  necessary for bounds checking and other error handling
\  routines.

\ Requires  Screens 4 & 5.

\ SCREEN 6
\  MATMATH - Vectoring variables                  D&G  09/20/90

\ The following variabled are used for vector execution;
\  the variables will contain the cfa's of the 'active'
\  arguments.

VARIABLE LMATRIX    ( Left-hand Matrix )

VARIABLE RMATRIX    ( Right-hand Matrix )

VARIABLE SMATRIX    ( Solution Matrix )

: [LIT] ( --- )
     [COMPILE] LITERAL
;


\ SCREEN 7
\ MATMATH - Single Argument specifier            D&G  09/20/90

: UNI_MAT   ( --- )       \ Usage: UNI_MAT <matrix-name>
   STATE @                \ Vectors one matrix
     IF    ' [LIT] LMATRIX [LIT] COMPILE !
     ELSE  ' LMATRIX !
     THEN ;
 IMMEDIATE


\ Requires Screens 4, 6 & 7.

\ SCREEN 8
\ MATMATH - Two argument specifier               D&G  09/20/90

: BI_MAT  ( --- ) \ Usage: BI_MAT <matrix-name> <matrix-name>
   STATE @        \ Vectors two matrices
     IF    ' [LIT] LMATRIX [LIT] COMPILE !
           ' [LIT] RMATRIX [LIT] COMPILE !
     ELSE  ' LMATRIX ! ' RMATRIX !
     THEN ;
IMMEDIATE


\ Requires Screens 4, 6 & 8.

\ SCREEN 9
\ MATMATH - Three argument specifier             D&G  09/20/90

: TRI_MAT  ( --- )
     \ Usage:  TRI_MAT <matrix-name> <matrix-name> <matrix-name>
     STATE @  \ Vectors three matrices
      IF   ' [LIT]  LMATRIX [LIT] COMPILE !
           ' [LIT]  RMATRIX [LIT] COMPILE !
           ' [LIT]  SMATRIX [LIT] COMPILE !
      ELSE ' LMATRIX ! ' RMATRIX ! ' SMATRIX !
      THEN ;
 IMMEDIATE


\ Requires Screens 4, 6 & 9.
```

```
\ SCREEN 10
\ MATMATH - Current Argument Assignment              D&G  09/20/90

: LMAT.NAME ( -- )   \ Usage: LMAT.NAME
   LMATRIX @         \ returns name of the assigned Left-hand
   >NAME .NAME ;     \ matrix.

: RMAT.NAME ( -- )   \ Usage RMAT.NAME
   RMATRIX @         \ returns name of the assigned Right-hand
   >NAME .NAME ;     \ matrix.

: SMAT.NAME ( -- )   \ Usage SMAT.NAME
    SMATRIX @        \ returns name of the Solution Matrix.
    >NAME .NAME ;


\ Requires Screens 4, 6 to 10.

\ SCREEN 11
\ MATMATH - Parameters of Active Arguments           D&G  09/20/90

: L#ROWS ( --- i ) LMATRIX @ >BODY WSIZE + @ ;  \ get #rows
: L#COLS ( --- i ) LMATRIX @ >BODY @ ;          \ get #columns
: ?L#BYTES ( --- n ) LMATRIX @ >BODY WSIZE 2* + @ ;
                              \ fetches size of element storage

: R#ROWS ( --- i ) RMATRIX @ >BODY WSIZE + @ ;  \ get #rows
: R#COLS ( --- i ) RMATRIX @ >BODY @ ;          \ get #columns

: S#ROWS ( --- i ) SMATRIX @ >BODY WSIZE + @ ;  \ get #rows
: S#COLS ( --- i ) SMATRIX @ >BODY @ ;          \ get #columns


\ Requires Screens 4, 6 to 9 & 11.

\ SCREEN 12
\ MATMATH - Element Accessing Commands               D&G  09/20/90

: LM@  ( i j --- f ) LMATRIX   PERFORM F@ ;     \ fetch a fp #
: LM!  ( f i j --- ) LMATRIX   PERFORM F! ;     \ store a fp #

: RM@  ( i j --- f ) RMATRIX   PERFORM F@ ;     \ fetch a fp #
: RM!  ( f i j --- ) RMATRIX   PERFORM F! ;     \ store a fp #

: SM@  ( i j --- f ) SMATRIX   PERFORM F@ ;     \ fetch a fp #
: SM!  ( f i j --- ) SMATRIX   PERFORM F! ;     \ store a fp #


\ Requires Screens 4, 6 to 9  & 12.

\ SCREEN 13
\ MATMATH - Parameter Access Utility (?DIMENSION) D&G  09/20/90

: ?DIMENSION ( -- )  \ Usage: UNI_MAT A ?DIMENSION
    CR               \ Provides characteristic data of a matrix
    LMATRIX @ >NAME .NAME  ." has  "
    L#ROWS . ." rows and "  L#COLS . ." columns."
    CR
    ?L#BYTES  ." Element length is " . ." bytes."
;


\ Requires Screens 4, 6, 7, 11 & 13.

\ SCREEN 14
\ MATMATH - Matrix Manipulation Variables            D&G  09/20/90

\ Index variables for loops ( Kludge for enhanced readability )
      VARIABLE EYE
      VARIABLE JAY
      VARIABLE KAY
      VARIABLE ELEL
      VARIABLE EYEEYE
```

```
\ SCREEN 15
\ MATMATH - Initializing a Matrix (STACK->MAT)     D&G  05/28/91

: STACK->MAT ( fo ... fnm -- ) \ Usage: UNI_MAT A STACK->MAT
   0 L#ROWS 1-                 \ Initializes the matrix with
    DO 0 L#COLS 1-             \ values from the stack.
       DO J I LM! -1
       +LOOP -1
    +LOOP
;


\ Requires Screens 4, 6, 7, 11, 12 & 15.

\ SCREEN 16
\ MATMATH - Matrix printing/output (.MAT)          D&G  05/28/91
\ (Requires user input for advancing the display after each scr)
 4 PLACES  \ Sets # of decimal places to be printed to left of
           \ the decimal.
 VARIABLE LINES      25 CONSTANT (#LINES
: .MAT ( --- )    \ Usage: UNI_MAT A .MAT
   CR 0 LINES !   \ Formatted printing of a matrix
   L#ROWS 0
     DO L#COLS 0
        DO J I LM@ 4 10 F.R
           I 0> IF I 7 MOD 0= IF CR 1 LINES +! THEN THEN
        LOOP CR 1 LINES +!  (#LINES LINES @ - 3 - 0<
        IF ." Press any key to continue" KEY DROP 0 LINES ! CR
        THEN
     LOOP
;


\ Requires Screens 4, 6, 7, 11, 12 & 16.

\ SCREEN 17
\ MATMATH - Determining Norms (ROW-NORM)           D&G  09/20/90
\ Beginning of various norms calculation for a matrix.

: ROW-NORM ( -- f )    \ Usage: UNI_MAT A ROW-NORM
    0.0E0 L#ROWS 0     \ Determines the row norm of matrix, A
     DO  0.0E0 L#COLS 0
         DO J I LM@ FABS F+
         LOOP FMAX
     LOOP
;


\ Requires Screens 4, 6, 7, 11, 12 & 17.
\ SCREEN 18
\ MATMATH - Determining Norms (COL-NORM)           D&G  09/20/90

: COL-NORM ( -- f )   \ Usage: UNI_MAT A COL-NORM
    0.0E0 L#COLS 0    \ Determines the column norm of a matrix
     DO 0.0E0 L#ROWS 0
        DO I J LM@ FABS F+
        LOOP FMAX
     LOOP
;

\ Requires Screens 4, 6, 7, 11, 12 & 18.

\ SCREEN 19
\ MATMATH - Determining Euclidian norm (E-NORM)    D&G  09/20/90

: E-NORM ( -- f )   \ Usage: UNI_MAT A E-NORM
    0.0E0 L#ROWS 0 \ Determines the Euclidean norm of a matrix
     DO L#COLS 0
         DO J I LM@ FDUP F* F+
         LOOP
     LOOP
    FSQRT
;
```

```
\ SCREEN 20
\ MATMATH - Creating a Null Matrix (MZERO)          D&G  09/21/90

: MZERO  ( -- )              \ Usage: UNI_MAT A MZERO
   0 0 LMATRIX PERFORM       \ Creates a Null Matrix
   L#ROWS L#COLS ?L#BYTES * *
   0 FILL
;
\\  \ Alternate element-by-element definition of MZERO
: MZERO  ( -- )              \ Usage: UNI_MAT A MZERO
   0.0E0 L#COLS 0            \ Creates a Null Matrix
    DO L#ROWS 0
       DO FDUP I J LM!
       LOOP
    LOOP FDROP
;

\ Requires Screens 4, 6, 7, 11, 12 & 20.

\ SCREEN 21
\ MATMATH - Initializing a Matrix (MFILL)           D&G  09/21/90

: MFILL  ( f -- )  \ Usage: UNI_MAT A MFILL
   L#COLS 0        \ Fills a matrix with the number on the stack
    DO L#ROWS 0
       DO FDUP I J LM!
       LOOP
    LOOP
   FDROP
;

\ Requires Screens 4, 6, 7, 11, 12 & 21.

\ SCREEN 22
\ MATMATH - Creates an identity matrix (IDENTITY) D&G  09/20/90

: IDENTITY  ( -- )     \ Usage: UNI_MAT A IDENTITY
   L#COLS 0            \ Creates an identity matirx
    DO L#ROWS 0
       DO I J = IF   1.0E0
                 ELSE 0.0E0
                 THEN
          I J LM!
       LOOP
    LOOP
;

\ Requires Screens 4, 6, 7, 11, 12 & 22.


\ SCREEN 23
\ MATMATH - Matrix addition (MPLUS)                 D&G  09/20/90

: MPLUS  ( -- )  \ Usage: TRI_MAT A B C MPLUS
    S#ROWS 0     \ Places the matrix addition of A + B in C
     DO S#COLS 0
        DO  J I LM@
            J I RM@    F+
            J I SM!
        LOOP
     LOOP
;


\ Requires Screens 4, 6, 9, 11, 12 & 23.

\ SCREEN 24
\ MATMATH - Matrix subtraction (MMINUS)             D&G  09/20/90

: MMINUS ( -- )  \ Usage: TRI_MAT A B C MMINUS
    S#ROWS 0     \ Places the matrix difference of A - B in C
     DO S#COLS 0
        DO J I LM@
           J I RM@  F-
           J I SM!
        LOOP
     LOOP
;

\ Requires Screens 4, 6, 9, 11, 12 & 24.
```

```
\ SCREEN 25
\ MATMATH - Matrix multiplication (MMULT)          D&G  09/20/90

: MMULT ( -- )   \ Usage: TRI_MAT A B C MMULT
    S#ROWS 0     \ Places the product of A and B in C
     DO S#COLS 0 I EYE !
        DO 0.0E0 L#COLS 0
           DO EYE @ I LM@
                 I J RM@  F* F+
           LOOP
           J I SM!
        LOOP
     LOOP
;


\ Requires Screens 4, 6, 9, 11, 12, 14 & 25.

\ SCREEN 26
\ MATMATH -  Scalar Multiplication (SMULT)         D&G  09/20/90

: SMULT ( -- )    \ Usage: TRI_MAT SVAR A C SMULT
   R#COLS 0       \ Scales the contents of A by SVAR and stores
    DO R#ROWS 0   \ the result in C
       DO I J RM@
          LMATRIX @ >BODY F@ F*
          I J SM!
       LOOP
    LOOP
;


\ Requires Screens 4, 6, 9, 11, 12 & 26.

\ SCREEN 27
\ MATMATH - Matrix Partitioning (MINSERT)          D&G  09/20/90

: MINSERT ( n m -- ) \ Usage: BI_MAT A B MINSERT
   L#ROWS 0          \ Inserts matrix  A into B with the 0 0
    DO 2DUP          \ element of A matching element n m of B
      I       0   LMATRIX  PERFORM
      ROT I + ROT RMATRIX  PERFORM
      L#COLS ?L#BYTES *  CMOVE
    LOOP
   2DROP
;

\ Alternate element-by-element partitioning algorithm follows


\ Requires Screens 4, 6, 8, 11, 12 & 27.

\ SCREEN 27a
\ MATMATH - Matrix Partitioning (MINSERT)        11:25 05/06/92
\ Alternate element-by-element partitioning algorithm

\\
VARIABLE NN  VARIABLE MM
: MINSERT  ( n m -- ) \ Usage: BI_MAT A B MINSERT
  MM ! NN !
  L#ROWS 0       \ Inserts matrix  A into B with the 0 0
    DO L#COLS 0  \ element of A matching the mth nth element
       DO        \ of B
          J I   LM@
          J NN @ + I MM @ + RM!
       LOOP
    LOOP
;


\ Requires Screens 4, 6, 8, 11, 12 & 27a.
```

```
\ SCREEN 28
\ MATMATH - Matrix Partitioning (MEXTRACT)          D&G  09/20/90

: MEXTRACT ( n m -- ) \ Usage: BI_MAT A B MEXTRACT
   R#ROWS 0           \ Extracts matrix B from matrix A with n m
    DO 2DUP           \ element of A matching element 0 0 of B
      SWAP I +  SWAP  LMATRIX  PERFORM
      I         0     RMATRIX  PERFORM
      R#COLS  ?L#BYTES  *   CMOVE
    LOOP
   2DROP
;

\ Alternate element-by-element partitioning algorithm follows

\ Requires Screens 4, 6, 8, 11, 12 & 28.

\ SCREEN 28a
\ MATMATH - Matrix Partitioning (MEXTRACT)          11:26 05/06/92
\ Alternate element-by-element partitioning algorithm

\\
\ VARIABLE NN  VARIABLE MM

: MEXTRACT ( n m -- ) \ Usage: BI_MAT A B MEXTRACT
  MM ! NN !           \ Extracts matrix B from matrix A with the
    R#COLS 0          \ n m element of A matching element 0 0
       DO R#ROWS 0    \ of B
         DO I MM @ +  J NN @ + LM@ I J RM!
         LOOP
      LOOP
;

\ Requires Screens 4, 6, 8, 11, 12 & 28a.

\ SCREEN 29
\ MATMATH - Matrix Copying Facilities (MCOPY)       D&G  09/20/90
: MCOPY ( -- )              \ Usage BI_MAT A B MCOPY
   0 0 LMATRIX PERFORM      \ Copies A to B; both matrices assumed
   0 0 RMATRIX PERFORM      \ to be the same size
   L#ROWS L#COLS ?L#BYTES * *
  CMOVE
;
\\ Alternate element-by-element definition of MCOPY
: MCOPY ( -- ) \ Usage BI_MAT A B MCOPY
   L#COLS 0    \ Copies contents of A into B; element by element
    DO L#ROWS 0
       DO I J LM@ I J RM!
       LOOP
    LOOP
 ;

\ Requires Screens 4, 6, 8, 11, 12 & 29.

\ SCREEN 30
\ MATMATH - Copying Facilities (TRANS_COPY)         D&G  09/20/90

: TRANS_COPY ( -- ) \ Usage: BI_MAT A B TRANS_COPY
   L#COLS 0         \ The transpose of A is placed in B.
    DO L#ROWS 0
       DO  I J LM@  J I RM!
       LOOP
    LOOP
;

\ Requires Screens 4, 6, 8, 11, 12 & 30.

\ SCREEN 31
\ MATMATH - Special Utility (DIAGONAL)              D&G  09/20/90

: DIAGONAL ( -- ) \ Usage: UNI_MAT A DIAGONAL
   L#COLS 0       \ Retains the diagonal elements and all other
    DO L#ROWS 0   \ elements are set to zero.
       DO I J = IF
                ELSE 0.0E0 I J LM!
                THEN
       LOOP
    LOOP
;
\ Requires Screens 4, 6, 7, 11, 12 & 31.
```

```
\ SCREEN 32
\ MATMATH - Matrix Utilities (Variables)          D&G  05/04/92

\  The algorithms for the following matrix utilities, LUDECOMP,
\   DETERMINANT, INVERSE and SOLVE have been adapted from
\   Numerical Recipes pp. 35ff [PRES87]

\ Temporary storage variables ( mainly for LUDECOMP)
 VARIABLE IMAX
 VARIABLE O/E          1 O/E !
 FVARIABLE AMAX
 FVARIABLE TEMP
 FVARIABLE FOM
 FVARIABLE TINY        1.0E-20 TINY F!

\ SCREEN 33
\ MATMATH -Dynamic Memory Management (TEMP_MATRIX)D&G  05/04/92

\ Defining command that stores required parameters for
\  the temporary or space/matrix that will operate similar
\  to the data structures defined using MATRIX

: TEMP_MATRIX
   CREATE
     0 , 0 , 0 , 0 ,  ( #Columns, #Rows, #bytes, base address)
   DOES>
     >R R@ @ ROT * + R@ WSIZE 2* + @ * R> WSIZE 3 * + @ +
;


\ Requires screens 1, 2, 3 & 33.

\ SCREEN 34
\ MATMATH -Dynamic Memory Management (TEMP_ALLOT) D&G  05/04/92

\ TEMP_ALLOT -- allots required scratch space at the end of the
\  dictionary and stores necessary characterizing parameters.

: TEMP_ALLOT  \ Usage:  n m UNI_MAT A TEMP_ALLOT
   2DUP
   LMATRIX @ >BODY !       ( Storing #columns in matrix header )
   LMATRIX @ >BODY WSIZE + !       ( Store #rows in header      )
    #BYTES DUP                     ( Byte width of an element   )
   LMATRIX @ >BODY WSIZE 2* + !    ( Store element byte width   )
    HERE                           ( Beginning address for data)
   LMATRIX @ >BODY WSIZE 3 * + !   ( Store address in header    )
   * *  ALLOT                     ( Allot required data space  )
;


\ Requires screens 1, 2, 3, 33 & 34.

\ SCREEN 35
\ MATMATH -Dynamic Memory Management(TEMP_DEALLOT) D&G 05/04/92

\ TEMP_DEALLOT -- frees scratch space at the end of the
\  dictionary and clears the characterizing parameters.

: TEMP_DEALLOT  \ Usage:   UNI_MAT A TEMP_DEALLOT
   LMATRIX @ >BODY @              ( # Columns in tempary matrix)
   LMATRIX @ >BODY WSIZE +    @   ( # Row in tempary matrix )
   LMATRIX @ >BODY WSIZE 2* + @   ( Byte width in temp. matrix )
    * * NEGATE ALLOT              ( Deallocate memory space )
   0 LMATRIX @ >BODY !             ( Zero # Columns )
   0 LMATRIX @ >BODY WSIZE +   !   ( Zero # Row     )
   0 LMATRIX @ >BODY WSIZE 2* + !  ( Zero Byte-width)
   0 LMATRIX @ >BODY WSIZE 3 * + !  ( Zero Base Address )
;


\ Requires screens 1, 2, 3, 33 & 35.
```

```
\ SCREEN 36
\ MATMATH -Application of Dynamic Memmory Mgmt.   D&G  05/04/92
\ (for LUDECOMP)

\ Scratch space/arrays are created when the routine is executed;
\  at the end of the operation this space

\ Interm data storage & initialization for dyn. memory alloc.
 VARIABLE ADDR
: LMAT->ADDR LMATRIX  @  ADDR ! ;
: ADDR->LMAT ADDR @ LMATRIX !   ;

\ Necessary Temporary Matrix Structures
 TEMP_MATRIX SCALER
 TEMP_MATRIX RECORD

\ SCREEN 37
\ MATMATH -Application of Dynamic Memmory Mgmt.   D&G  05/04/92
\ (for LUDECOMP)
: SET-UPL ( -- )    \ Allocates scratch space
    L#COLS 1 LMAT->ADDR UNI_MAT SCALER TEMP_ALLOT ADDR->LMAT
     ( Creates floating point array based on size of LMATRIX  )
     WSIZE EQU #BYTES              ( Set bit-width -> integer  )
    L#COLS 1 LMAT->ADDR UNI_MAT RECORD TEMP_ALLOT ADDR->LMAT
     ( Creates integer array based on size of LMATRIX         )
      FPSIZE EQU #BYTES            ( Set bit-width -> fl. pt   )
;

: M_RESTORE ( -- )    \ Freeing of Scratch space
  LMAT->ADDR UNI_MAT RECORD TEMP_DEALLOT ADDR->LMAT
  LMAT->ADDR UNI_MAT SCALER TEMP_DEALLOT ADDR->LMAT
;

\ Requires screens 33, 34, 35, 36 and 37.

\ SCREEN 38
\ MATMATH -  Matrix Utility (LUDECOMP)            D&G  05/04/92
\ LUDECOMP: replaces a square matrix with a rowwise permutation
\ of its LU decomposition; the algorithm uses Crout's Method.

\ Inner Loops of forthcoming code
: UPDATE_TEMP ( -- ) \ TEMP = TEMP - LMAT[i,k]LMAT[k,j]
      TEMP F@
      EYE @ KAY @ LM@
      KAY @ JAY @ LM@   F*  F-
      TEMP  F!
;
: CALC_FOM ( -- ) \ Calculate Figure of Merit for Pivot
   EYE @ 0  SCALER
   F@  TEMP F@  FABS     F*
   FOM F!
;


\ SCREEN 39
\ MATMATH -  LUDECOMP (continued)                 D&G  05/04/92
\ Finding the largest element in each row for implict scaling
\  information ( Do 12 Loop of algorithm )
: SCALOR ( -- )
   L#ROWS 0
    DO 0.0E0 AMAX F!   L#COLS 0
        DO J I LM@ FABS FDUP AMAX F@ F>
           IF AMAX  F! ELSE FDROP THEN
        LOOP
        AMAX F@ F0=
        IF M_RESTORE LMATRIX @ >NAME .NAME ." is singular."
            TRUE ABORT
        THEN
        1.0E0 AMAX F@ F/   I 0 SCALER F!
    LOOP
;
```

```
\ SCREEN 40
\ MATMATH -  LUDECOMP (continued)                    D&G  09/17/90

\ Solves Upper Triangular Coef. [Eqn 2.3.12] Do 14 & 13 Loops
: CALC_BETA ( -- )
    0 EYE !
    BEGIN EYE @ JAY @ LM@  TEMP F!  EYE @ 0>
          IF 0 KAY !
             BEGIN
               UPDATE_TEMP
               1 KAY +!   KAY @  EYE @  >=
             UNTIL
             TEMP F@   EYE @ JAY @ LM!
          THEN
          1 EYE +! EYE @ JAY @  >=
    UNTIL
;

\ SCREEN 41
\ MATMATH -  LUDECOMP (continued)                    D&G  09/17/90
\  Solves Lower Triangular Coef. [Eqn 2.3.13] Do 16 & 15 Loops
: CALC_ALPHA  ( -- )
   JAY @ EYE !
    BEGIN
      EYE @ JAY @ LM@ TEMP F! JAY @ 0>
      IF 0 KAY !
        BEGIN
          UPDATE_TEMP   1 KAY +! KAY @ JAY @ >=
        UNTIL
          TEMP F@ EYE @ JAY @ LM!
      THEN
      CALC_FOM   FOM F@   AMAX F@  F>=
      IF EYE @ IMAX !  FOM F@ AMAX F! THEN
      1 EYE +! EYE @ L#ROWS >=
    UNTIL  ;


\ SCREEN 42
\ MATMATH -  LUDECOMP (continued)                    D&G  05/04/92

\  Row Interchange (Do 17 Loop) & Largest Pivot Element (Do 16)
: ROW_INTERCHANGE ( -- )
   JAY @ IMAX @ <>    \ Check if necessary to interchange rows
   IF L#ROWS 0        \ Interchange the rows element by element
       DO IMAX @ I LM@   FOM F!
          JAY  @ I LM@   IMAX @ I LM!
                 FOM F@   JAY  @ I LM!
       LOOP
      O/E @ -1 * O/E !       \ Adjust parity of # of interchanges
      JAY  @ 0 SCALER F@     \ Change scaling factor
      IMAX @ 0 SCALER F!
   THEN
;


\ SCREEN 43
\ MATMATH -  LUDECOMP (continued)                    D&G  05/04/92

\  Dividing by Pivot Element (Do 18 Loop)
: DIVXPIVOT ( -- )
   IMAX @ JAY @ 0 RECORD !
   JAY @ L#ROWS 1- <>
   IF JAY @ DUP LM@ F0=
      IF TINY F@ JAY @ DUP LM! THEN
      1.0E0 JAY @ DUP LM@ F/ FOM F!
      JAY @ 1+ EYE !
        BEGIN
          EYE @ JAY @ LM@ FOM F@ F* EYE @ JAY @ LM!
          1 EYE +! EYE @ L#ROWS  >=
        UNTIL
   THEN
;
```

```
\ SCREEN 44
\ MATMATH -  LUDECOMP (continued)                    D&G  09/17/90

\  Crout's Method summation over columns (Do 19 Loop)
: FACTORIZATION ( -- )
    L#ROWS 0
      DO I JAY !
        I 0>
        IF CALC_BETA THEN
        0.0E0 AMAX  F!
        CALC_ALPHA
        ROW_INTERCHANGE
        DIVXPIVOT
      LOOP
;

\ SCREEN 45
\ MATMATH -  LUDECOMP (continued)                    D&G  05/04/92

: LUDECOMP  \ Usage: UNI_MAT A LUDECOMP
   SET-UPL  \ LU Decomposition using Crout's Method.
   1 0/E !  \ This version frees work/scratch vectors.
   SCALOR
   FACTORIZATION
   L#ROWS DUP LM@ F0=
   IF TINY F@ L#ROWS DUP LM! THEN
   M_RESTORE
;

\ Requires Screens 4, 6, 7, 11, 12, 14 & 32-45.

\ SCREEN 46
\ MATMATH -  LUDECOMP_                               D&G  05/04/92
\  (without memory restoration)

: LUDECOMP_ ( -- ) \ Performs LU Decomposition on LMAT
   SET-UPL         \  Scratch space retained; integral part
   1 0/E !         \  of DETERMINANT, SOLVE and INVERSE.
   SCALOR
   FACTORIZATION
   L#ROWS DUP LM@ F0=
   IF TINY F@ L#ROWS DUP LM! THEN
;

\ Requires Screens 4, 6, 7, 11, 12, 14, 32-44 & 46.

\ SCREEN 47
\ MATMATH - Dynamic Memmory Mgmt.                    D&G  05/04/92
\ ( for DETERMINANT and SOLVE )

TEMP_MATRIX ORIGINAL  \ Temporary or scratch Matrix Structure

: SET-UP2  ( -- )  \ Allocates scratch space
   L#ROWS L#COLS
   LMAT->ADDR UNI_MAT ORIGINAL TEMP_ALLOT ADDR->LMAT
     ( Creates floating point array based on size of LMATRIX  )
;

: M_RESTORE2 ( -- )   \ Freeing of Scratch space
  LMAT->ADDR UNI_MAT ORIGINAL TEMP_DEALLOT ADDR->LMAT
;

\ SCREEN 48
\ MATMATH - Dynamic Memmory Mgmt.                    D&G  05/04/92
\ ( for DETERMINANT and SOLVE )
: LMAT->ORIG  ( -- )       \ Used to preserve "original" argument
   SET-UP2                 \ Creation of the work space, ORIGINAL
    0 0 LMATRIX PERFORM     \ Copies "original" matrix (active
   0 0 ORIGINAL            \ argument) to temporary matrix.
   L#ROWS L#COLS ?L#BYTES * *     \ Similar to MCOPY.
   CMOVE
;
: ORIG->LMAT ( -- )
   0 0 ORIGINAL                \ Transfers "original" matrix back
   0 0 LMATRIX PERFORM         \ to matrix vectored to LMATRIX,
   L#ROWS L#COLS ?L#BYTES * * \ then, free the scratch space.
   CMOVE                       \ Similar to MCOPY.
   M_RESTORE2                  \ Restore memory space.
;
```

```
\ SCREEN 49
\ MATMATH -  Matrix Utility (DETERMINANT)          D&G  05/04/92
\  [PREIS87] p. 39

: DETERMINANT ( --  ) \ Usage: UNI_MAT TEST DETERMINANT
   LMAT->ORIG          \ The determinant is calculated from a
   LUDECOMP_           \ LU decomposition of the matrix.
     O/E @ S>F
     L#ROWS 0
       DO  I DUP LM@ F*
       LOOP
   M_RESTORE          \ Free scratch space of LUDECOMP_
   ORIG->LMAT         \ Transfer "original" and free space
;


\ Requires Screens 4, 6, 7, 11, 12, 14, 32-44 & 47-49.

\ SCREEN 50
\ MATMATH -  Matrix Utility (SOLVE)                D&G  09/20/90

\ Forward and Backward Substitution used in SOLVE follow same
\  algorithm as those in LUDECOMP

\ Inner Loops of forthcoming code
: UPDATE_TEMPi ( -- )  \ TEMP = TEMP - LMAT[i,j]RMAT[j,0]
      TEMP F@
      EYE @ JAY @ LM@
      JAY @ 0 RM@   F*  F-
      TEMP  F!
;


\ SCREEN 51
\ MATMATH -  SOLVE (continued)                     D&G  05/04/92
\ Forward Substition
: FWD ( -- )  \ Equation 2.3.6 (Do loops 12 &11)[PRES87]pp.36-8
   L#ROWS 0
     DO I 0 RECORD @ ELEL !
        ELEL @ 0 RM@ TEMP F! I 0 RM@ ELEL @ 0 RM! EYEEYE @ -1 <>
        IF EYEEYE @ JAY !
          BEGIN I EYE ! UPDATE_TEMPi
                1 JAY +! JAY @ I >=
          UNTIL
        ELSE TEMP F@ F0<>
          IF I EYEEYE ! THEN
        THEN
        TEMP F@  I 0 RM!
      LOOP
;


\ SCREEN 52
\ MATMATH -  SOLVE (continued)                     D&G  09/17/90

\ Backward Substition
: BWD ( -- )  \ Equation 2.3.7 (Do loops 14 &13)[PRES87]pp.36-8.
    0 L#ROWS 1-
     DO I 0 RM@ TEMP F! I L#ROWS 1- <
        IF I 1+ JAY !
           BEGIN I EYE ! UPDATE_TEMPi
                 1 JAY +! JAY @ L#ROWS >=
           UNTIL
        THEN
       TEMP F@  I I LM@ F/ I 0 RM! -1
     +LOOP
;
```

```
\ SCREEN 53
\ MATMATH -  SOLVE (continued)                          D&G  05/04/92

: SOLVE  ( -- )  \ Usage: BI_MAT A B  SOLVE
    LMAT->ORIG   \ Solves a set of linear equations Ax = B
    LUDECOMP_    \ The solution vector is returned in B
    -1 EYEEYE !
    FWD
    BWD
    M_RESTORE   \ Free memory space from LUDECOMP_ temp. arrays
    ORIG->LMAT  \ Transfer "original" matrix; free assoc. space
;


\ Requires Screens 4, 6, 7, 11, 12, 14, 32-44, 46-48 & 50-53.

\ SCREEN 54
\ MATMATH -  SOLVE_                                     D&G  09/20/90

: SOLVE_  ( -- ) \ SOLVE; assumes LUDECOMP performed;
   -1 EYEEYE !   \ scratch space not recovered; integral part
   FWD BWD       \ of INVERSE
;


\ Requires Screens 4, 6, 7, 11, 12, 14, 32-44, 46-48, 50-52 & 54.

\ SCREEN 55
\ MATMATH - Dyn. Memory Mgmt (for INVERSE)              D&G  05/04/92

\ Scratch space/arrays are created when the routine is
\  executed; at the end of the operation this space

\ Interm data storage & initialization for dynamic memory alloc.
  VARIABLE NN
  TEMP_MATRIX INVERTED  \ Scratch space
  TEMP_MATRIX COL_INV


\ SCREEN 56
\ MATMATH - Dyn. Memory Mgmt (for INVERSE)              D&G  05/04/92

: SET-UPI ( -- )   \ Allocates scratch space
    L#ROWS DUP LMAT->ADDR UNI_MAT INVERTED TEMP_ALLOT ADDR->LMAT
     ( Creates floating point matrix based on size of LMATRIX )
    L#COLS 1   LMAT->ADDR UNI_MAT COL_INV  TEMP_ALLOT ADDR->LMAT
     ( Creates floating point column vector based on L#COLS   )
;

: M_RESTOREI ( -- )   \ Freeing of Scratch space
   LMAT->ADDR UNI_MAT COL_INV  TEMP_DEALLOT ADDR->LMAT
   LMAT->ADDR UNI_MAT INVERTED TEMP_DEALLOT ADDR->LMAT
;


\ SCREEN 57
\ MATMATH -  Matrix Utility (INVERSE)                   D&G  05/04/92
\ (for INVERSE) [PRES87] p.38.

: EXTRACT_COL  ( n -- ) \ n is selected column #
    NN ! L#ROWS 0
      DO I NN @ INVERTED  F@
         I 0    COL_INV   F!
      LOOP
;
: REPLACE_COL ( n -- ) \ n is selected column #
   NN ! L#ROWS 0
     DO I 0    COL_INV  F@
        I NN @ INVERTED F!
     LOOP
;
\ These routines could be cobbled from MINSERT & MEXTRACT.
```

```
\ SCREEN 58
\ MATMATH -  INVERSE (continued)                    D&G  05/04/92

: INVERSE ( -- )  \ Usage: UNI_MAT A INVERSE
   LUDECOMP_
   SET-UPI
   LMAT->ADDR UNI_MAT INVERTED IDENTITY ADDR->LMAT
   L#COLS 0
     DO I EXTRACT_COL ['] COL_INV RMATRIX ! SOLVE_
        I REPLACE_COL
     LOOP
   0 0 INVERTED       0 0 LMATRIX PERFORM
   L#ROWS L#COLS ?L#BYTES * * CMOVE ADDR->LMAT
   M_RESTOREI  M_RESTORE
;

\ Requires Scns 4, 6-8, 11, 12, 14, 22, 32-44, 46-8, 50-2 & 54-8.
```