MATLAB Users' Guide

Cleve Moler*
Department of Computer Science
University of New Mexico

November, 1980

Abstract

MATLAB is an interactive computer program that serves as a convenient "laboratory" for computations involving matrices. It provides easy access to matrix software developed by the LINPACK and EISPACK projects. The program is written in Fortran and is designed to be readily installed under any operating system which permits interactive execution of Fortran programs.

*http://www.mathworks.com/company/cleve_bio.shtml

Contents

1 Introduction

MATLAB is an interactive computer program that serves as a convenient "laboratory" for computations involving matrices. It provides easy access to matrix software developed by the LINPACK and EISPACK projects [1-3]. The capabilities range from standard tasks such as solving simultaneous linear equations and inverting matrices, through symmetric and nonsymmetric eigenvalue problems, to fairly sophisticated matrix tools such as the singular value decomposition.

It is expected that one of MATLAB's primary uses will be in the classroom. It should be useful in introductory courses in applied linear algebra, as well as more advanced courses in numerical analysis, matrix theory, statistics and applications of matrices to other disciplines. In nonacademic settings, MATLAB can serve as a "desk calculator" for the quick solution of small problems involving matrices.

The program is written in Fortran and is designed to be readily installed under any operating system which permits interactive execution of Fortran programs. The resources required are fairly modest. There are less than 7000 lines of Fortran source code, including the LINPACK and EISPACK subroutines used. With proper use of overlays, it is possible run the system on a minicomputer with only 32K bytes of memory.

The size of the matrices that can be handled in MATLAB depends upon the amount of storage that is set aside when the system is compiled on a particular machine. We have found that an allocation of 5000 words for matrix elements is usually quite satisfactory. This provides room for several 20 by 20 matrices, for example. One implementation on a virtual memory system provides 100,000 elements. Since most of the algorithms used access memory in a sequential fashion, the large amount of allocated storage causes no difficulties.

In some ways, MATLAB resembles SPEAKEASY [4] and, to a lesser extent, APL. All are interactive terminal languages that ordinarily accept single-line commands or statements, process them immediately, and print the results. All have arrays or matrices as principal data types. But for MATLAB, the matrix is the only data type (although scalars, vectors and text are special cases), the underlying system is portable and requires fewer resources, and the supporting subroutines are more powerful and, in some cases, have better numerical properties.

Together, LINPACK and EISPACK represent the state of the art in software for matrix computation. EISPACK is a package of over 70 Fortran subroutines for various matrix eigenvalue computations that are based for the most part on Algol procedures published by Wilkinson, Reinsch and their colleagues [5]. LINPACK is a package of 40 Fortran subroutines (in each of four data types) for solving and analyzing simultaneous linear equations and related matrix problems. Since MATLAB is not primarily concerned with either execution time efficiency or storage savings, it ignores most of the special matrix properties that LINPACK and EISPACK subroutines use to advantage. Consequently, only 8 subroutines from LINPACK and 5 from EISPACK are actually involved.

In more advanced applications, MATLAB can be used in conjunction with other programs in several ways. It is possible to define new MATLAB functions and add them to the system. With most operating systems, it is possible to use the local file system to pass matrices between MATLAB and other programs. MATLAB command and statement input can be obtained from a local file instead of from the terminal. The most power and flexibility is obtained by using MATLAB as a subroutine which is called by other programs.

This document first gives an overview of MATLAB from the user's point of view. Several extended examples involving data fitting, partial differential equations, eigenvalue sensitivity and other topics are included. A formal definition of the MATLAB language and an brief description of the parser and interpreter are given. The system was designed and programmed using techniques described by Wirth [6], implemented in nonrecursive, portable Fortran. There is a brief discussion of some of the matrix algorithms and of their numerical properties. The final section describes how MATLAB can be used with other programs. The appendix includes the HELP documentation available on-line.

2 Elementary operations

MATLAB works with essentially only one kind of object, a rectangular matrix with complex elements. If the imaginary parts of the elements are all zero, they are not printed, but they still occupy storage. In some situations, special meaning is attached to 1 by 1 matrices, that is scalars, and to 1 by n and m by 1 matrices, that is row and column vectors.

Matrices can be introduced into MATLAB in four different ways:

- Explicit list of elements,
- Use of FOR and WHILE statements,
- Read from an external file,
- Execute an external Fortran program.

The explicit list is surrounded by angle brackets, ’<’ and ’>’, and uses the semicolon ’;’ to indicate the ends of the rows. For example, the input line

```
A = <1 2 3; 4 5 6; 7 8 9>
```

will result in the output

```
A     =

    1.    2.   3.
    4.    5.   6.
    7.    8.   9.
```

The matrix A will be saved for later use. The individual elements are separated by commas or blanks and can be any MATLAB expressions, for example

```
x = < -1.3, 4/5, 4*atan(1) >
```

results in

```
X     =

  -1.3000   0.8000   3.1416
```

The elementary functions available include `sqrt, log, exp, sin, cos, atan, abs, round, real, imag,` and `conjg`.

Large matrices can be spread across several input lines, with the carriage returns replacing the semicolons. The above matrix could also have been produced by

```
A = < 1 2 3
      4 5 6
      7 8 9 >
```

Matrices can be input from the local file system. Say a file named 'xyz' contains five lines of text,

```
A = <
1 2 3
4 5 6
7 8 9
>;
```

then the MATLAB statement `EXEC('xyz')` reads the matrix and assigns it to A .

The FOR statement allows the generation of matrices whose elements are given by simple formulas. Our example matrix A could also have been produced by

```
for i = 1:3, for j = 1:3, a(i,j) = 3*(i-1)+j;
```

The semicolon at the end of the line suppresses the printing, which in this case would have been nine versions of A with changing elements.

Several statements may be given on a line, separated by semicolons or commas.

Two consecutive periods anywhere on a line indicate continuation. The periods and any following characters are deleted, then another line is input and concatenated onto the previous line.

Two consecutive slashes anywhere on a line cause the remainder of the line to be ignored. This is useful for inserting comments.

Names of variables are formed by a letter, followed by any number of letters and digits, but only the first 4 characters are remembered.

The special character prime (') is used to denote the transpose of a matrix, so

```
x = x'
```

changes the row vector above into the column vector

```
X     =

  -1.3000
   0.8000
   3.1416
```

Individual matrix elements may be referenced by enclosing their subscripts in parentheses. When any element is changed, the entire matrix is reprinted. For example, using the above matrix,

```
a(3,3) = a(1,3) + a(3,1)
```

results in

```
A     =

    1.    2.    3.
    4.    5.    6.
    7.    8.   10.
```

Addition, subtraction and multiplication of matrices are denoted by +, -, and * . The operations are performed whenever the matrices have the proper dimensions. For example, with the above A and x, the expressions A + x and x*A are incorrect because A is 3 by 3 and x is now 3 by 1. However,

```
b = A*x
```

is correct and results in the output

```
B      =


   9.7248
  17.6496
  28.7159
```

Note that both upper and lower case letters are allowed for input (on those systems which have both), but that lower case is converted to upper case.

There are two ”matrix division” symbols in MATLAB,
and / . (If your terminal does not have a backslash, use $ instead, or see CHAR.) If A and B are matrices, then `A\B` and `B/A` correspond formally to left and right multiplication of B by the inverse of A , that is inv(A)*B and B*inv(A), but the result is obtained directly without the computation of the inverse. In the scalar case, 3
1 and 1/3 have the same value, namely one-third. In general, `A\B` denotes the solution X to the equation `A*X = B` and `B/A` denotes the solution to `X*A = B`.

Left division, `A\B`, is defined whenever B has as many rows as A . If A is square, it is factored using Gaussian elimination. The factors are used to solve the equations `A*X(:,j) = B(:,j)` where `B(:,j)` denotes the j-th column of B. The result is a matrix X with the same dimensions as B. If A is nearly singular (according to the LINPACK condition estimator, RCOND), a warning message is printed. If A is not square, it is factored using Householder orthogonalization with column pivoting. The factors are used to solve the under- or overdetermined equations in a least squares sense. The result is an m by n matrix X where m is the number of columns of A and n is the number of columns of B . Each column of X has at most k

nonzero components, where k is the effective rank of A .

Right division, `B/A`, can be defined in terms of left division by `B/A = (A'\B')'`.

For example, since our vector b was computed as A*x, the statement

```
y = A\b
```

results in

```
Y      =


  -1.3000
   0.8000
   3.1416
```

Of course, y is not exactly equal to x because of the roundoff errors involved in both A*x and A$\underline{,}$ but we are not printing enough digits to see the difference. The result of the statement

```
   e = x - y
```

depends upon the particular computer being used. In one case it produces

```
   E     =

      1.0e-15 *

        .3053
       -.2498
        .0000
```

The quantity 1.0e-15 is a scale factor which multiplies all the components which follow. Thus our vectors x and y actually agree to about 15 decimal places on this computer.

It is also possible to obtain element-by-element multiplicative operations. If A and B have the same dimensions, then A .* B denotes the matrix whose elements are simply the products of the individual elements of A and B . The expressions A ./ B and A .
B give the quotients of the individual elements.

There are several possible output formats. The statement

```
   long, x
```

results in

```
   X     =

      -1.300000000000000
        .800000000000000
       3.141592653589793
```

The statement

```
   short
```

restores the original format.

The expression A**p means A to the p-th power. It is defined if A is a square matrix and p is a scalar. If p is an integer greater than one, the power is computed by repeated multiplication. For other values of p the calculation involves the eigenvalues and eigenvectors of A.

Previously defined matrices and matrix expressions can be used inside brackets to generate larger matrices, for example

```
   C = <A, b; <4 2 0>*x, x'>
```

results in

```
   C     =

      1.0000   2.0000   3.0000   9.7248
      4.0000   5.0000   6.0000  17.6496
      7.0000   8.0000  10.0000  28.7159
     -3.6000  -1.3000   0.8000   3.1416
```

There are four predefined variables, EPS, FLOP, RAND and EYE. The variable EPS is used as a tolerance is determining such things as near singularity and rank. Its initial value is the distance from 1.0 to the next largest floating point number on the particular computer being used. The user may reset this to any other value, including zero. EPS is changed by CHOP, which is described in §11.

The value of RAND is a random variable, with a choice of a uniform or a normal distribution.

The name EYE is used in place of I to denote identity matrices because I is often used as a subscript or as sqrt(-1). The dimensions of EYE are determined by context. For example,

```
B = A + 3*EYE
```

adds 3 to the diagonal elements of A and

```
X = EYE/A
```

is one of several ways in MATLAB to invert a matrix.

FLOP provides a count of the number of floating point operations, or ”flops”, required for each calculation.

A statement may consist of an expression alone, in which case a variable named ANS is created and the result stored in ANS for possible future use. Thus

```
A\A - EYE
```

is the same as

```
ANS = A\A - EYE
```

(Roundoff error usually causes this result to be a matrix of ”small” numbers, rather than all zeros.)

All computations are done using either single or double precision real arithmetic, whichever is appropriate for the particular computer. There is no mixed-precision arithmetic. The Fortran COMPLEX data type is not used because many systems create unnecessary underflows and overflows with complex operations and because some systems do not allow double precision complex arithmetic.

2. MATLAB functions

Much of MATLAB's computational power comes from the various matrix functions available. The current list includes:

```
INV(A)          - Inverse.
DET(A)          - Determinant.
COND(A)         - Condition number.
RCOND(A)        - A measure of nearness to singularity.
EIG(A)          - Eigenvalues and eigenvectors.
SCHUR(A)        - Schur triangular form.
HESS(A)         - Hessenberg or tridiagonal form.
POLY(A)         - Characteristic polynomial.
SVD(A)          - Singular value decomposition.
PINV(A,eps)     - Pseudoinverse with optional tolerance.
RANK(A,eps)     - Matrix rank with optional tolerance.
LU(A)           - Factors from Gaussian elimination.
```

CHOL(A)	- Factor from Cholesky factorization.
QR(A)	- Factors from Householder orthogonalization.
RREF(A)	- Reduced row echelon form.
ORTH(A)	- Orthogonal vectors spanning range of A.
EXP(A)	- e to the A.
LOG(A)	- Natural logarithm.
SQRT(A)	- Square root.
SIN(A)	- Trigonometric sine.
COS(A)	- Cosine.
ATAN(A)	- Arctangent.
ROUND(A)	- Round the elements to nearest integers.
ABS(A)	- Absolute value of the elements.
REAL(A)	- Real parts of the elements.
IMAG(A)	- Imaginary parts of the elements.
CONJG(A)	- Complex conjugate.
SUM(A)	- Sum of the elements.
PROD(A)	- Product of the elements.
DIAG(A)	- Extract or create diagonal matrices.
TRIL(A)	- Lower triangular part of A.
TRIU(A)	- Upper triangular part of A.
NORM(A,p)	- Norm with p = 1, 2 or 'Infinity'.
EYE(m,n)	- Portion of identity matrix.
RAND(m,n)	- Matrix with random elements.
ONES(m,n)	- Matrix of all ones.
MAGIC(n)	- Interesting test matrices.
HILBERT(n)	- Inverse Hilbert matrices.
ROOTS(C)	- Roots of polynomial with coefficients C.
DISPLAY(A,p)	- Print base p representation of A.
KRON(A,B)	- Kronecker tensor product of A and B.
PLOT(X,Y)	- Plot Y as a function of X .
RAT(A)	- Find "simple" rational approximation to A.
USER(A)	- Function defined by external program.

Some of these functions have different interpretations when the argument is a matrix or a vector and some of them have additional optional arguments. Details are given in the HELP document in the appendix.

Several of these functions can be used in a generalized assignment statement with two or three variables on the left hand side. For example

```
<X,D> = EIG(A)
```

stores the eigenvectors of A in the matrix X and a diagonal matrix containing the eigenvalues in the matrix D. The statement

```
EIG(A)
```

simply computes the eigenvalues and stores them in ANS.

Future versions of MATLAB will probably include additional functions, since they can easily be added to the system.

3 Rows, columns and submatrices

Individual elements of a matrix can be accessed by giving their subscripts in parentheses, eg. A(1,2), x(i), TAB(ind(k)+1). An expression used as a subscript is rounded to the nearest integer.

Individual rows and columns can be accessed using a colon ':' (or a '—') for the free subscript. For example, A(1,:) is the first row of A and A(:,j) is the j-th column. Thus

```
A(i,:) = A(i,:) + c*A(k,:)
```

adds c times the k-th row of A to the i-th row.

The colon is used in several other ways in MATLAB, but all of the uses are based on the following definition.

```
j:k    is the same as  <j, j+1, ..., k>
j:k    is empty if  j > k .
j:i:k  is the same as  <j, j+i, j+2i, ..., k>
j:i:k  is empty if  i > 0 and j > k or if i < 0 and j < k .
```

The colon is usually used with integers, but it is possible to use arbitrary real scalars as well. Thus

```
1:4  is the same as  <1, 2, 3, 4>
0: 0.1: 0.5 is the same as <0.0, 0.1, 0.2, 0.3, 0.4, 0.5>
```

In general, a subscript can be a vector. If X and V are vectors, then X(V) is ¡X(V(1)), X(V(2)), ..., X(V(n))¿ . This can also be used with matrices. If V has m components and W has n components, then A(V,W) is the m by n matrix formed from the elements of A whose subscripts are the elements of V and W. Combinations of the colon notation and the indirect subscripting allow manipulation of various submatrices. For example,

```
A(<1,5>,:) = A(<5,1>,:)  interchanges rows 1 and 5 of A.
A(2:k,1:n)  is the submatrix formed from rows 2 through k
   and columns 1 through n of A .
A(:,<3 1 2>)  is a permutation of the first three columns.
```

The notation A(:) has a special meaning. On the right hand side of an assignment statement, it denotes all the elements of A, regarded as a single column. When an expression is assigned to A(:), the current dimensions of A, rather than of the expression, are used.

4 FOR, WHILE and IF

The FOR clause allows statements to be repeated a specific number of times. The general form is

```
FOR variable = expr,  statement, ..., statement, END
```

The END and the comma before it may be omitted. In general, the expression may be a matrix, in which case the columns are stored one at a time in the variable and the following statements, up to the END or the end of the line, are executed. The expression is often of the form j:k, and its "columns" are simply the scalars from j to k. Some examples (assume n has already been assigned a value):

```
for i = 1:n, for j = 1:n, A(i,j) = 1/(i+j-1);
```

generates the Hilbert matrix.

```
for j = 2:n-1, for i = j:n-1, ...
   A(i,j) = 0; end; A(j,j) = j; end; A
```

changes all but the "outer edge" of the lower triangle and then prints the final matrix.

```
for h = 1.0: -0.1: -1.0, (<h, cos(pi*h)>)
```

prints a table of cosines.

```
<X,D> = EIG(A); for v = X, v, A*v
```

displays eigenvectors, one at a time.

The WHILE clause allows statements to be repeated an indefinite number of times. The general form is

```
WHILE expr relop expr,   statement,..., statement, END
```

where relop is =, ¡, ¿, ¡=, ¿=, or ¡¿ (not equal) . The statements are repeatedly executed as long as the indicated comparison between the real parts of the first components of the two expressions is true. Here are two examples. (Exercise for the reader: What do these segments do?)

```
eps = 1;
while 1 + eps > 1, eps = eps/2;
eps = 2*eps

E = 0*A;  F = E + EYE; n = 1;
while NORM(E+F-E,1) > 0, E = E + F; F = A*F/n; n = n + 1;
E
```

The IF clause allows conditional execution of statements. The general form is

```
IF expr relop expr,   statement, ..., statement,
   ELSE statement, ..., statement
```

The first group of statements are executed if the relation is true and the second group are executed if the relation is false. The ELSE and the statements following it may be omitted. For example,

```
if abs(i-j) = 2, A(i,j) = 0;
```

5 Commands, text, files and macros.

MATLAB has several commands which control the output format and the overall execution of the system.

The HELP command allows on-line access to short portions of text describing various operations, functions and special characters. The entire HELP document is reproduced in an appendix.

Results are usually printed in a scaled fixed point format that shows 4 or 5 significant figures. The commands SHORT, LONG, SHORT E, LONG E and LONG Z alter the output format, but do not alter the precision of the computations or the internal storage.

The WHO, WHAT and WHY commands provide information about the functions and variables that are currently defined.

The CLEAR command erases all variables, except EPS, FLOP, RAND and EYE. The statement A = ¡¿ indicates that a "0 by 0" matrix is to be stored in A. This causes A to be erased so that its storage can be used for other variables.

The RETURN and EXIT commands cause return to the underlying operating system through the Fortran RETURN statement.

MATLAB has a limited facility for handling text. Any string of characters delineated by quotes (with two quotes used to allow one quote within the string) is saved as a vector of integer values with '1' = 1, 'A' = 10, ' ' = 36, etc. (The complete list is in the appendix under CHAR.) For example

```
'2*A + 3'  is the same as  <2 43 10 36 41 36 3>
```

It is possible, though seldom very meaningful, to use such strings in matrix operations. More frequently, the text is used as a special argument to various functions.

```
NORM(A,'inf')    computes the infinity norm of A .
DISPLAY(T)       prints the text stored in T .
EXEC('file')     obtains MATLAB input from an external file.
SAVE('file')     stores all the current variables in a file.
LOAD('file')     retrieves all the variables from a file.
PRINT('file',X)  prints X on a file.
DIARY('file')    makes a copy of the complete MATLAB session.
```

The text can also be used in a limited string substitution macro facility. If a variable, say T, contains the source text for a MATLAB statement or expression, then the construction

```
> T <
```

causes T to be executed or evaluated. For example

```
T = '2*A + 3';
S = 'B = >T< + 5'
A = 4;
> S <
```

produces

```
B     =

   16.
```

Some other examples are given under MACRO in the appendix. This facility is useful for fairly short statements and expressions. More complicated MATLAB "programs" should use the EXEC facility.

The operations which access external files cannot be handled in a completely machine-independent manner by portable Fortran code. It is necessary for each particular installation to provide a subroutine which associates external text files with Fortran logical unit numbers.

6. Census example

Our first extended example involves predicting the population of the United States in 1980 using extrapolation of various fits to the census data from 1900 through 1970. There are eight observations, so we begin with the MATLAB statement

```
   n = 8
```

The values of the dependent variable, the population in millions, can be entered with

```
   y = < 75.995   91.972  105.711  123.203   ...
        131.669  150.697  179.323  203.212>'
```

In order to produce a reasonably scaled matrix, the independent variable, time, is transformed from the interval [1900,1970] to [-1.00,0.75]. This can be accomplished directly with

```
   t = -1.0:0.25:0.75
```

or in a fancier, but perhaps clearer, way with

```
   t = 1900:10:1970;   t = (t - 1940*ones(t))/40
```

Either of these is equivalent to

```
   t = <-1 -.75 -.50 -.25 0 .25 .50 .75>
```

The interpolating polynomial of degree n-1 involves an Vandermonde matrix of order n with elements that might be generated by

```
   for i = 1:n, for j = 1:n, a(i,j) = t(i)**(j-1);
```

However, this results in an error caused by 0**0 when i = 5 and j = 1 . The preferable approach is

```
   A = ones(n,n);
   for i = 1:n, for j = 2:n, a(i,j) = t(i)*a(i,j-1);
```

Now the statement

```
   cond(A)
```

produces the output

```
   ANS   =

      1.1819E+03
```

which indicates that transformation of the time variable has resulted in a reasonably well conditioned matrix.

The statement

```
   c = A\y
```

results in

```
C     =


  131.6690
   41.0406
  103.5396
  262.4535
 -326.0658
 -662.0814
  341.9022
  533.6373
```

These are the coefficients in the interpolating polynomial

$$c_1 + c_2 t + \dots + c_n t^{n-1}$$

Our transformation of the time variable has resulted in t = 1 corresponding to the year 1980. Consequently, the extrapolated population is simply the sum of the coefficients. This can be computed by

```
p = sum(c)
```

The result is

```
P     =


  426.0950
```

which indicates a 1980 population of over 426 million. Clearly, using the seventh degree interpolating polynomial to extrapolate even a fairly short distance beyond the end of the data interval is not a good idea.

The coefficients in least squares fits by polynomials of lower degree can be computed using fewer than n columns of the matrix.

```
for k = 1:n, c = A(:,1:k)\y,  p = sum(c)
```

would produce the coefficients of these fits, as well as the resulting extrapolated population. If we do not want to print all the coefficients, we can simply generate a small table of populations predicted by polynomials of degrees zero through seven. We also compute the maximum deviation between the fitted and observed values.

```
for k = 1:n, X = A(:,1:k);  c = X\y;  ...
   d(k) = k-1;  p(k) = sum(c);  e(k) = norm(X*c-y,'inf');
<d, p, e>
```

The resulting output is

```
     0   132.7227  70.4892
     1   211.5101   9.8079
     2   227.7744   5.0354
     3   241.9574   3.8941
     4   234.2814   4.0643
     5   189.7310   2.5066
     6   118.3025   1.6741
     7   426.0950   0.0000
```

The zeroth degree fit, 132.7 million, is the result of fitting a constant to the data and is simply the average. The results obtained with polynomials of degree one through four all appear reasonable. The maximum deviation of the degree four fit is slightly greater than the degree three, even though the sum of the squares of the deviations is less. The coefficients of the highest powers in the fits of degree five and six turn out to be negative and the predicted populations of less than 200 million are probably unrealistic. The hopefully absurd prediction of the interpolating polynomial concludes the table.

We wish to emphasize that roundoff errors are not significant here. Nearly identical results would be obtained on other computers, or with other algorithms. The results simply indicate the difficulties associated with extrapolation of polynomial fits of even modest degree.

A stabilized fit by a seventh degree polynomial can be obtained using the pseudoinverse, but it requires a fairly delicate choice of a tolerance. The statement

```
   s = svd(A)
```

produces the singular values

```
   S     =

      3.4594
      2.2121
      1.0915
      0.4879
      0.1759
      0.0617
      0.0134
      0.0029
```

We see that the last three singular values are less than 0.1 , consequently, A can be approximately by a matrix of rank five with an error less than 0.1 . The Moore-Penrose pseudoinverse of this rank five matrix is obtained from the singular value decomposition with the following statements

```
   c = pinv(A,0.1)*y, p = sum(c), e = norm(a*c-y,'inf')
```

The output is

```
   C     =

    134.7972
     67.5055
```

```
  23.5523
   9.2834
   3.0174
   2.6503
  -2.8808
   3.2467

P     =

 241.1720

E     =

   3.9469
```

The resulting seventh degree polynomial has coefficients which are much smaller than those of the interpolating polynomial given earlier. The predicted population and the maximum deviation are reasonable. Any choice of the tolerance between the fifth and sixth singular values would produce the same results, but choices outside this range result in pseudoinverses of different rank and do not work as well.

The one term exponential approximation

```
  y(t) = k exp(pt)
```

can be transformed into a linear approximation by taking logarithms.

```
  log(y(t)) = log k + pt

            = c  + c t
               1    2
```

The following segment makes use of the fact that a function of a vector is the function applied to the individual components.

```
X = A(:,1:2);
c = X\log(y)
p = exp(sum(c))
e = norm(exp(X*c)-y,'inf')
```

The resulting output is

```
C     =

   4.9083
   0.5407

P     =

 232.5134
```

```
E     =

   4.9141
```

The predicted population and maximum deviation appear satisfactory and indicate that the exponential model is a reasonable one to consider.

As a curiousity, we return to the degree six polynomial. Since the coefficient of the high order term is negative and the value of the polynomial at t = 1 is positive, it must have a root at some value of t greater than one. The statements

```
X = A(:,1:7);
c = X\y;
c = c(7:-1:1);  //reverse the order of the coefficients
z = roots(c)
```

produce

```
Z     =

   1.1023-  0.0000*i
   0.3021+  0.7293*i
  -0.8790+  0.6536*i
  -1.2939-  0.0000*i
  -0.8790-  0.6536*i
   0.3021-  0.7293*i
```

There is only one real, positive root. The corresponding time on the original scale is

```
1940 + 40*real(z(1))

  =  1984.091
```

We conclude that the United States population should become zero early in February of 1984.

6 Partial differential equation example

Our second extended example is a boundary value problem for Laplace's equation. The underlying physical problem involves the conductivity of a medium with cylindrical inclusions and is considered by Keller and Sachs [7].

Find a function $u(x, y)$ satisfying Laplace's equation

$$u_{xx} + u_{yy} = 0.$$

The domain is a unit square with a quarter circle of radius rho removed from one corner. There are Neumann conditions on the top and bottom edges and Dirichlet conditions on the remainder of the boundary.

```
            u  = 0
             n

        ------------
       |            .
       |            .
       |             .
       |              .  u = 1
       |                .
       |                   .
       |                      .
u = 0  |                       |
       |                       |
       |                       |
       |                       |  u = 1
       |                       |
       |                       |
       |                       |
        -----------------------

                 u  = 0
                  n
```

The effective conductivity of an medium is then given by the integral along the left edge,

$$\sigma = \int_0^1 u_n(0, y)dy$$

It is of interest to study the relation between the radius rho and the conductivity sigma. In particular, as rho approaches one, sigma becomes infinite.

Keller and Sachs use a finite difference approximation. The following technique makes use of the fact that the equation is actually Laplace's equation and leads to a much smaller matrix problem to solve.

Consider an approximate solution of the form

$$u = \sum_{j=1}^{n} c_j r^{2j-1} \cos(2j - 1)t$$

where r,t are polar coordinates (t is theta). The coefficients are to be determined. For any set of coefficients, this function already satisfies the differential equation because the basis functions are harmonic; it satisfies the normal derivative boundary condition on the bottom edge of the domain because we used cos t in preference to sin t ; and it satisfies the boundary condition on the left edge of the domain because we use only odd multiples of t .

The computational task is to find coefficients so that the boundary conditions on the remaining edges are satisfied as well as possible. To accomplish this, pick m points (r,t) on the remaining edges. It is desirable to have m ¿ n and in practice we usually choose m to be two or three times as large as n . Typical values of n are 10 or 20 and of m are 20 to 60. An m by n matrix A is generated. The i,j element is the j-th basis function, or its normal derivative, evaluated at the i-th boundary point. A right hand side with m components is also generated. In this example, the elements of the right hand side are either zero or one. The coefficients are then found by solving the overdetermined set of equations

$$Ac = b$$

in a least squares sense.

Once the coefficients have been determined, the approximate solution is defined everywhere on the domain. It is then possible to compute the effective conductivity sigma . In fact, a very simple formula results,

$$\sigma = \sum_{j=1}^{n} (-1)^{j-1} c_j$$

To use MATLAB for this problem, the following "program" is first stored in the local computer file system, say under the name "PDE".

```
//Conductivity example.
//Parameters ---
   rho       //radius of cylindrical inclusion
   n         //number of terms in solution
   m         //number of boundary points
//initialize operation counter
   flop = <0 0>;
//initialize variables
   m1 = round(m/3);   //number of points on each straight edge
   m2 = m - m1;       //number of points with Dirichlet conditions
   pi = 4*atan(1);
//generate points in Cartesian coordinates
   //right hand edge
   for i = 1:m1, x(i) = 1; y(i) = (1-rho)*(i-1)/(m1-1);
   //top edge
   for i = m2+1:m, x(i) = (1-rho)*(m-i)/(m-m2-1); y(i) = 1;
   //circular edge
   for i = m1+1:m2, t = pi/2*(i-m1)/(m2-m1+1); ...
      x(i) = 1-rho*sin(t);  y(i) = 1-rho*cos(t);
//convert to polar coordinates
   for i = 1:m-1, th(i) = atan(y(i)/x(i));  ...
      r(i) = sqrt(x(i)**2+y(i)**2);
   th(m) = pi/2;  r(m) = 1;
//generate matrix
   //Dirichlet conditions
   for i = 1:m2, for j = 1:n, k = 2*j-1; ...
      a(i,j) = r(i)**k*cos(k*th(i));
   //Neumann conditions
   for i = m2+1:m, for j = 1:n, k = 2*j-1; ...
      a(i,j) = k*r(i)**(k-1)*sin((k-1)*th(i));
```

```
//generate right hand side
   for i = 1:m2, b(i) = 1;
   for i = m2+1:m, b(i) = 0;
//solve for coefficients
   c = A\b
//compute effective conductivity
   c(2:2:n) = -c(2:2:n);
   sigma = sum(c)
//output total operation count
   ops = flop(2)
```

The program can be used within MATLAB by setting the three parameters and then accessing the file. For example,

```
   rho = .9;
   n = 15;
   m = 30;
   exec('PDE')
```

The resulting output is

```
   RHO   =

      .9000

   N     =

    15.

   M     =

    30.

   C     =

      2.2275
     -2.2724
      1.1448
      0.1455
     -0.1678
     -0.0005
     -0.3785
      0.2299
      0.3228
     -0.2242
     -0.1311
      0.0924
      0.0310
     -0.0154
     -0.0038

   SIGM  =

      5.0895

   OPS   =

      16204.
```

A total of 16204 floating point operations were necessary to set up the matrix, solve for the coefficients and compute the conductivity. The operation count is roughly proportional to m*n**2. The results obtained for sigma as a function of rho by this approach are essentially the same as those obtained by the finite difference technique of Keller and Sachs, but the computational effort involved is much less.

7 Eigenvalue sensitivity example

In this example, we construct a matrix whose eigenvalues are moderately sensitive to perturbations and then analyze that sensitivity. We begin with the statement

```
 B = <3 0 7; 0 2 0; 0 0 1>
```

which produces

```
 B      =


      3.    0.    7.
      0.    2.    0.
      0.    0.    1.
```

Obviously, the eigenvalues of B are 1, 2 and 3 . Moreover, since B is not symmetric, these eigenvalues are slightly sensitive to perturbation. (The value b(1,3) = 7 was chosen so that the elements of the matrix A below are less than 1000.)

We now generate a similarity transformation to disguise the eigenvalues and make them more sensitive.

```
 L = <1 0 0; 2 1 0; -3 4 1>, M = L\L'

 L      =


      1.    0.    0.
      2.    1.    0.
     -3.    4.    1.


 M      =
       1.0000    2.0000   -3.0000
      -2.0000   -3.0000   10.0000
      11.0000   18.0000  -48.0000
```

The matrix M has determinant equal to 1 and is moderately badly conditioned. The similarity transformation is

```
   A = M*B/M

   A     =

     -64.0000   82.0000   21.0000
     144.0000 -178.0000  -46.0000
    -771.0000  962.0000  248.0000
```

Because det(M) = 1 , the elements of A would be exact integers if there were no roundoff. So,

```
   A = round(A)

   A     =

     -64.   82.   21.
     144. -178.  -46.
    -771.  962.  248.
```

This, then, is our test matrix. We can now forget how it was generated and analyze its eigenvalues.

```
   <X,D> = eig(A)

   D     =

       3.0000    0.0000    0.0000
       0.0000    1.0000    0.0000
       0.0000    0.0000    2.0000

   X     =

       -.0891    3.4903   41.8091
        .1782   -9.1284  -62.7136
       -.9800   46.4473  376.2818
```

Since A is similar to B, its eigenvalues are also 1, 2 and 3. They happen to be computed in another order by the EISPACK subroutines. The fact that the columns of X, which are the eigenvectors, are so far from being orthonormal is our first indication that the eigenvalues are sensitive. To see this sensitivity, we display more figures of the computed eigenvalues.

```
   long, diag(D)

   ANS   =

      2.999999999973599
      1.000000000015625
      2.000000000011505
```

We see that, on this computer, the last five significant figures are contaminated by roundoff error. A somewhat superficial explanation of this is provided by

```
   short,  cond(X)

   ANS   =

      3.2216e+05
```

The condition number of X gives an upper bound for the relative error in the computed eigenvalues. However, this condition number is affected by scaling.

```
   X = X/diag(X(3,:)),  cond(X)

   X     =

         .0909     .0751     .1111
        -.1818    -.1965    -.1667
        1.0000    1.0000    1.0000

   ANS   =

      1.7692e+03
```

Rescaling the eigenvectors so that their last components are all equal to one has two consequences. The condition of X is decreased by over two orders of magnitude. (This is about the minimum condition that can be obtained by such diagonal scaling.) Moreover, it is now apparent that the three eigenvectors are nearly parallel.

More detailed information on the sensitivity of the individual eigenvalues involves the left eigenvectors.

```
   Y = inv(X'),  Y'*A*X

   Y     =

    -511.5000  259.5000  252.0000
     616.0000 -346.0000 -270.0000
     159.5000  -86.5000  -72.0000

   ANS   =
       3.0000     .0000     .0000
        .0000    1.0000     .0000
        .0000     .0000    2.0000
```

We are now in a position to compute the sensitivities of the individual eigenvalues.

```
for j = 1:3, c(j) = norm(Y(:,j))*norm(X(:,j)); end,  C

C     =

  833.1092
  450.7228
  383.7564
```

These three numbers are the reciprocals of the cosines of the angles between the left and right eigenvectors. It can be shown that perturbation of the elements of A can result in a perturbation of the j-th eigenvalue which is c(j) times as large. In this example, the first eigenvalue has the largest sensitivity.

We now proceed to show that A is close to a matrix with a double eigenvalue. The direction of the required perturbation is given by

```
E = -1.e-6*Y(:,1)*X(:,1)'

E     =

   1.0e-03 *

     .0465    -.0930     .5115
    -.0560     .1120    -.6160
    -.0145     .0290    -.1595
```

With some trial and error which we do not show, we bracket the point where two eigenvalues of a perturbed A coalesce and then become complex.

```
eig(A + .4*E),  eig(A + .5*E)

ANS   =

    1.1500
    2.5996
    2.2504

ANS   =

   2.4067 +  .1753*i
   2.4067 -  .1753*i
   1.1866 + 0.0000*i
```

Now, a bisecting search, driven by the imaginary part of one of the eigenvalues, finds the point where two eigenvalues are nearly equal.

```
r = .4;  s = .5;

while s-r > 1.e-14, t = (r+s)/2; d = eig(A+t*E); ...
  if imag(d(1))=0, r = t; else, s = t;

long,  T

T     =

     .450380734134507
```

Finally, we display the perturbed matrix, which is obviously close to the original, and its pair of nearly equal eigenvalues. (We have dropped a few digits from the long output.)

```
A+t*E,  eig(A+t*E)

A

 -63.999979057   81.999958114   21.000230369
 143.999974778 -177.999949557  -46.000277434
-771.000006530  962.000013061  247.999928164

ANS   =

   2.415741150
   2.415740621
   1.168517777
```

The first two eigenvectors of A + t*E are almost indistinguishable indicating that the perturbed matrix is almost defective.

```
<X,D> = eig(A+t*E);  X = X/diag(X(3,:))

X     =

    .096019578     .096019586    .071608466
   -.178329614    -.178329608   -.199190520
   1.000000000    1.000000000   1.000000000

short,  cond(X)

ANS   =

   3.3997e+09
```

8 Syntax diagrams

A formal description of the language acceptable to MATLAB, as well as a flow chart of the MATLAB program, is provided by the syntax diagrams or syntax graphs of Wirth [6]. There are eleven non-terminal symbols in the language:

```
line, statement, clause, expression, term,
factor, number, integer, name, command, text .
```

The diagrams define each of the non-terminal symbols using the others and the terminal symbols:

```
letter -- A through Z,
digit  -- 0 through 9,
char   -- ( ) ; : + - * / \ = . , < >
quote  -- '
```

line

```
       |-----> statement >----|
       |                      |
       |-----> clause >-------|
       |                      |
-------|-----> expr >---------|------>
     | |                      | |
     | |-----> command >------| |
     | |                      | |
     | |-> > >-> expr >-> < >-| |
     | |                      | |
     | |----------------------| |
     |                          |
     |        |-< ; <-|         |
     |--------|       |---------|
              |-< , <-|
```

statement

```
     |-> name >--------------------------------|
     |          |                              |
     |          |         |--> : >---|         |
     |          |         |          |         |
     |          |-> ( >---|-> expr >-|---> ) >-|
     |                  |              |       |
-----|                  |-----< , <----|       |--> = >--> expr >--->
     |                                         |
     |       |--< , <---|                      |
     |       |          |                      |
     |-> < >---> name >---> > >----------------|
```

clause

```
     |---> FOR   >---> name >---> = >---> expr >--------------|
     |                                                        |
     | |-> WHILE >-|                                          |
     |-|           |-> expr >----------------------           |
     | |-> IF    >-|          |   |   |   |   |   |           |
-----|                        <   <=  =   <>  >=  >           |---->
     |                        |   |   |   |   |   |           |
     |                        ----------------------> expr >--|
     |                                                        |
     |---> ELSE  >--------------------------------------------|
     |                                                        |
     |---> END   >--------------------------------------------|
```

expr

```
       |-> + >-|
       |       |
-------|-------|-------> term >---------->
       |       |    |             |
       |-> - >-|    |  |-< + <-|  |
                    |  |       |  |
                    |--|-< - <-|--|
                       |       |
                       |-< : <-|
```

term

```
---------------------> factor >---------------------->
        |                                   |
        |             |-< * <-|             |
        |  |-------|  |       |  |-------|  |
        |--|       |--|-< / <-|--|       |--|
           |-< . <-|  |       |  |-< . <-|
                      |-< \ <-|
```

factor

```
     |----------------> number >---------------|
     |                                         |
     |-> name >--------------------------------|
     |          |                              |
     |          |         |--> : >---|         |
     |          |         |          |         |
     |          |-> ( >---|-> expr >-|---> ) >-|
     |                  |              |       |
     |                  |-----< , <----|       |
     |                                         |
-----|------------> ( >-----> expr >-----> ) >-|-|-------|----->
```

```
     |                                         | |       | |
     |                  |--------------|       | |-> ’ >-| |
     |                  |              |       |           |
     |------------> < >-|---> expr >---|-> > >-|           |
     |                    |          |         |           |
     |                    |--<   <---|         |           |
     |                    |          |         |           |
     |                    |--< ; <---|         |           |
     |                    |          |         |           |
     |                    |--< , <---|         |           |
     |                                         |           |
     |------------> > >-----> expr >-----> < >-|           |
     |                                         |           |
     |-----> factor >---> ** >---> factor >----|           |
     |                                                     |
     |------------> ’ >-----> text >-----> ’ >-------------|
```

number

```
    |----------|                          |-> + >-|
    |          |                          |       |
-----> int >-----> . >---> int >-----> E >---------> int >---->
             |                   | |      |       |        |
             |                   | |      |-> - >-|        |
             |                   | |                       |
             |---------------------------------------------|
```

int

```
------------> digit >----------->
          |           |
          |-----------|
```

name

```
                  |--< letter <--|
                  |              |
------> letter >--|--------------|----->
                  |              |
                  |--< digit  <--|
```

command

```
                        |--> name >--|
                        |            |
--------> name >--------|------------|---->
                        |            |
                        |--> char >--|
                        |            |
                        |---> ’ >----|
```

text

```
                |-> letter >--|
                |             |
                |-> digit >---|
----------------|             |-------------->
            |   |-> char >----|   |
            |   |             |   |
            |   |-> ' >-> ' >-|   |
            |                     |
            |---------------------|
```

9 The parser-interpreter

The structure of the parser-interpreter is similar to that of Wirth's compiler [6] for his simple language, PL/0 , except that MATLAB is programmed in Fortran, which does not have explicit recursion. The interrelation of the primary subroutines is shown in the following diagram.

```
MAIN
  |
MATLAB    |--CLAUSE
  |       |    |
PARSE-----|--EXPR----TERM----FACTOR
          |    |       |       |
          |    |-------|-------|
          |    |       |       |
          |  STACK1  STACK2  STACKG
          |
          |--STACKP--PRINT
          |
          |--COMAND
          |
          |
          |          |--CGECO
          |          |
          |          |--CGEFA
          |          |
          |--MATFN1--|--CGESL
          |          |
          |          |--CGEDI
          |          |
          |          |--CPOFA
          |
          |
          |          |--IMTQL2
          |          |
          |          |--HTRIDI
          |          |
          |--MATFN2--|--HTRIBK
          |          |
          |          |--CORTH
          |          |
          |          |--COMQR3
          |
          |
          |--MATFN3-----CSVDC
          |
          |
          |          |--CQRDC
          |--MATFN4--|
          |          |--CQRSL
          |
          |
          |          |--FILES
          |--MATFN5--|
                     |--SAVLOD
```

Subroutine PARSE controls the interpretation of each statement. It calls subroutines that process the various syntactic quantities such as command, expression, term and factor. A fairly simple program stack mechanism allows these subroutines to recursively ”call” each other along the lines allowed by the syntax diagrams. The four STACK subroutines manage the variable memory and perform elementary operations, such as matrix addition and transposition.

The four subroutines MATFN1 though MATFN4 are called whenever ”serious” matrix computations are required. They are interface routines which call the various LINPACK and EISPACK subroutines. MATFN5 primarily handles the file access tasks.

Two large real arrays, STKR and STKI, are used to store all the matrices. Four integer arrays are used to store the names, the row and column dimensions, and the pointers into the real stacks. The following diagram illustrates this storage scheme.

```
TOP         IDSTK     MSTK NSTK LSTK               STKR       STKI
 --      -- -- -- --   --   --   --              --------   --------
|  |--->|  |  |  |  | |  | |  | |  |----------->|        | |        |
 --      -- -- -- --   --   --   --              --------   --------
        |  |  |  |  | |  | |  | |  |            |        | |        |
         -- -- -- --   --   --   --              --------   --------
              .         .    .    .                  .          .
              .         .    .    .                  .          .
              .         .    .    .                  .          .
         -- -- -- --   --   --   --              --------   --------
BOT     |  |  |  |  | |  | |  | |  |            |        | |        |
 --      -- -- -- --   --   --   --              --------   --------
|  |--->| X|  |  |  | | 2| | 1| |  |----------->|  3.14  | |  0.00  |
--      -- -- -- --   --   --   --              --------   --------
       | A|  |  |  | | 2| | 2| |  |---------   |  0.00  | |  1.00  |
        -- -- -- --   --   --   --          \   --------   --------
       | E| P| S|  | | 1| | 1| |  |-------   ->| 11.00  | |  0.00  |
        -- -- -- --   --   --   --        \     --------   --------
       | F| L| O| P| | 1| | 2| |  |------  \   | 21.00  | |  0.00  |
        -- -- -- --   --   --   --       \  \   --------   --------
       | E| Y| E|  | |-1| |-1| |  |---    \ |  | 12.00  | |  0.00  |
        -- -- -- --   --   --   --    \   | |   --------   --------
       | R| A| N| D| | 1| | 1| |  |-   \  | |  | 22.00  | |  0.00  |
        -- -- -- --   --   --   --  \  |  \ \   --------   --------
                                    |  \   \ ->| 1.E-15 | |  0.00  |
                                    \   \   \   --------   --------
                                     \   \   ->|  0.00  | |  0.00  |
                                      \   \     --------   --------
                                       \   \   |  0.00  | |  0.00  |
                                        \   \   --------   --------
                                         \   ->|  1.00  | |  0.00  |
                                          \     --------   --------
                                           --->| URAND  | |  0.00  |
                                                --------   --------
```

The top portion of the stack is used for temporary variables and the bottom portion for saved variables. The figure shows the situation after the line

```
A = <11,12; 21,22>,  x = <3.14, sqrt(-1)>'
```

has been processed. The four permanent names, EPS, FLOP, RAND and EYE, occupy the last four positions of the variable stacks. RAND has dimensions 1 by 1, but whenever its value is requested, a random number generator is used instead. EYE has dimensions -1 by -1 to indicate that the actual dimensions must be determined later by context. The two saved variables have dimensions 2 by 2 and 2 by 1 and so take up a total of 6 locations.

Subsequent statements involving A and x will result in temporary copies being made in the top of the stack for use in the actual calculations. Whenever the top of the stack reaches the bottom, a message indicating memory has been exceeded is printed, but the current variables are not affected.

This modular structure makes it possible to implement MATLAB on a system with a limited amount of memory. The object code for the MATFN's and the LINPACK-EISPACK subroutines is rarely needed. Although it is not standard, many Fortran operating systems provide some overlay mechanism so that this code is brought into the main memory only when required. The variables, which occupy a relatively small portion of the memory, remain in place, while the subroutines which process them are loaded a few at a time.

10 The numerical algorithms

The algorithms underlying the basic MATLAB functions are described in the LINPACK and EISPACK guides [1-3]. The following list gives the subroutines used by these functions.

```
INV(A)          - CGECO,CGEDI
DET(A)          - CGECO,CGEDI
LU(A)           - CGEFA
RCOND(A)        - CGECO
CHOL(A)         - CPOFA
SVD(A)          - CSVDC
COND(A)         - CSVDC
NORM(A,2)       - CSVDC
PINV(A,eps)     - CSVDC
RANK(A,eps)     - CSVDC
QR(A)           - CQRDC,CQRSL
ORTH(A)         - CQRDC,CSQSL
\verb!A\B! and \verb!B/A!     - CGECO,CGESL if A is square.
                - CQRDC,CQRSL if A is not square.
EIG(A)          - HTRIDI,IMTQL2,HTRIBK if A is Hermitian.
                - CORTH,COMQR2         if A is not Hermitian.
SCHUR(A)        - same as EIG.
HESS(A)         - same as EIG.
```

Minor modifications were made to all these subroutines. The LINPACK routines were changed to replace the Fortran complex arithmetic with explicit references to real and imaginary parts. Since most of the floating point arithmetic is concentrated in a few low-level subroutines which perform vector operations (the Basic Linear Algebra Subprograms), this was not an extensive change. It also facilitated implementation of the FLOP and CHOP features which count and optionally truncate each floating point operation.

The EISPACK subroutine COMQR2 was modified to allow access to the Schur triangular form, ordinarily just an intermediate result. IMTQL2 was modified to make computation of the eigenvectors optional. Both subroutines were modified to eliminate the machine-dependent accuracy parameter and all the EISPACK subroutines were changed to include FLOP and CHOP.

The algorithms employed for the POLY and ROOTS functions illustrate an interesting aspect of the modern approach to eigenvalue computation. POLY(A) generates the characteristic polynomial of A and ROOTS(POLY(A)) finds the roots of that polynomial, which are, of course, the eigenvalues of A . But both POLY and ROOTS use EISPACK eigenvalues subroutines, which are based on similarity transformations. So the classical approach which characterizes eigenvalues as roots of the characteristic polynomial is actually reversed.

If A is an n by n matrix, POLY(A) produces the coefficients C(1) through C(n+1), with C(1) = 1, in

```
   DET(z*EYE-A) = C(1)*z**n + ... + C(n)*z + C(n+1) .
```

The algorithm can be expressed compactly using MATLAB:

```
   Z = EIG(A);
   C = 0*ONES(n+1,1);  C(1) = 1;
   for j = 1:n, C(2:j+1) = C(2:j+1) - Z(j)*C(1:j);
   C
```

This recursion is easily derived by expanding the product

```
      (z - z(1))*(z - z(2))* ... * (z-z(n)) .
```

It is possible to prove that POLY(A) produces the coefficients in the characteristic polynomial of a matrix within roundoff error of A . This is true even if the eigenvalues of A are badly conditioned. The traditional algorithms for obtaining the characteristic polynomial which do not use the eigenvalues do not have such satisfactory numerical properties.

If C is a vector with n+1 components, ROOTS(C) finds the roots of the polynomial of degree n ,

```
       p(z) = C(1)*z**n + ... + C(n)*z + C(n+1) .
```

The algorithm simply involves computing the eigenvalues of the companion matrix:

```
      A = 0*ONES(n,n)
      for j = 1:n, A(1,j) = -C(j+1)/C(1);
      for i = 2:n, A(i,i-1) = 1;
      EIG(A)
```

It is possible to prove that the results produced are the exact eigenvalues of a matrix within roundoff error of the companion matrix A, but this does not mean that they are the exact roots of a polynomial with coefficients within roundoff error of those in C . There are more accurate, more efficient methods for finding polynomial roots, but this approach has the crucial advantage that it does not require very much additional code.

The elementary functions EXP, LOG, SQRT, SIN, COS and ATAN are applied to square matrices by diagonalizing the matrix, applying the functions to the individual eigenvalues and then transforming back. For example, EXP(A) is computed by

```
      <X,D> = EIG(A);
      for j = 1:n, D(j,j) = EXP(D(j,j));
      X*D/X
```

This is essentially method number 14 out of the 19 'dubious' possibilities described in [8]. It is dubious because it doesn't always work. The matrix of eigenvectors X can be arbitrarily badly conditioned and all accuracy lost in the computation of X*D/X. A warning message is printed if RCOND(X) is very small, but this only catches the extreme cases. An example of a case not detected is when A has a double eigenvalue, but theoretically only one linearly independent eigenvector associated with it. The computed eigenvalues will be separated by something on the order of the square root of the roundoff level. This separation will be reflected in RCOND(X) which will probably not be small enough to trigger the error message. The computed EXP(A) will be accurate to only half precision. Better methods are known for computing EXP(A), but they do not easily extend to the other five functions and would require a considerable amount of additional code.

The expression A**p is evaluated by repeated multiplication if p is an integer greater than 1. Otherwise it is evaluated by

```
      <X,D> = EIG(A);
      for j = 1:n, D(j,j) = EXP(p*LOG(D(j,j)))
      X*D/X
```

This suffers from the same potential loss of accuracy if X is badly conditioned. It was partly for this reason that the case p = 1 is included in the general case. Comparison of A**1 with A gives some idea of the loss of accuracy for other values of p and for the elementary functions.

RREF, the reduced row echelon form, is of some interest in theoretical linear algebra, although it has little computational value. It is included in MATLAB for pedagogical reasons. The algorithm is essentially Gauss-Jordan elimination with detection of negligible columns applied to rectangular matrices.

There are three separate places in MATLAB where the rank of a matrix is implicitly computed: in RREF(A), in `A\B` for non- square A, and in the pseudoinverse PINV(A). Three different algorithms with three different criteria for negligibility are used and so it is possible that three different values could be produced for the same matrix. With RREF(A), the rank of A is the number of nonzero rows. The elimination algorithm used for RREF is the fastest of the three rank-determining algorithms, but it is the least sophisticated numerically and the least reliable. With `A\B`, the algorithm is essentially that used by example subroutine SQRST in chapter 9 of the LINPACK guide. With PINV(A), the algorithm is based on the singular value decomposition and is described in chapter 11 of the LINPACK guide. The SVD algorithm is the most time-consuming, but the most reliable and is therefore also used for RANK(A).

The uniformly distributed random numbers in RAND are obtained from the machine-independent random number generator URAND described in [9]. It is possible to switch to normally distributed random numbers, which are obtained using a transformation also described in [9].

The computation of $\sqrt{a^2 + b^2}$ is required in many matrix algorithms, particularly those involving complex arithmetic. A new approach to carrying out this operation is described by Moler and Morrison [10]. It is a cubically convergent algorithm which starts with a and b , rather than with their squares, and thereby avoids destructive arithmetic underflows and overflows. In MATLAB, the algorithm is used for complex modulus, Euclidean vector norm, plane rotations, and the shift calculation in the eigenvalue and singular value iterations.

11 FLOP and CHOP

Detailed information about the amount of work involved in matrix calculations and the resulting accuracy is provided by FLOP and CHOP. The basic unit of work is the ”flop”, or floating point operation. Roughly, one flop is one execution of a Fortran statement like

```
      S = S + X(I)*Y(I)
```

or

```
      Y(I) = Y(I) + T*X(I)
```

In other words, it consists of one floating point multiplication, together with one floating point addition and the associated indexing and storage reference operations.

MATLAB will print the number of flops required for a particular statement when the statement is terminated by an extra comma. For example, the line

```
      n = 20;  RAND(n)*RAND(n);,
```

ends with an extra comma. Two 20 by 20 random matrices are generated and multiplied together. The result is assigned to ANS, but the semicolon suppresses its printing. The only output is

```
        8800 flops
```

This is n**3 + 2*n**2 flops, n**2 for each random matrix and n**3 for the product.

FLOP is a predefined vector with two components. FLOP(1) is the number of flops used by the most recently executed statement, except that statements with zero flops are ignored. For example, after executing the previous statement,

```
      flop(1)/n**3
```

results in

```
      ANS   =

          1.1000
```

FLOP(2) is the cumulative total of all the flops used since the beginning of the MATLAB session. The statement

```
      FLOP = <0 0>
```

resets the total.

There are several difficulties associated with keeping a precise count of floating point operations. An addition or subtraction that is not paired with a multiplication is usually counted as a flop. The same is true of an isolated multiplication that is not paired with an addition. Each floating point division counts as a flop. But the number of operations required by system dependent library functions such as square root cannot be counted, so most elementary functions are arbitrarily counted as using only one flop.

The biggest difficulty occurs with complex arithmetic. Almost all operations on the real parts of matrices are counted. However, the operations on the complex parts of matrices are counted only when they involve nonzero elements. This means that simple operations on nonreal matrices require only about twice as many flops as the same operations on real matrices. This factor of two is not necessarily an accurate measure of the relative costs of real and complex arithmetic.

The result of each floating point operation may also be ”chopped” to simulate a computer with a shorter word length. The details of this chopping operation depend upon the format of the floating point word. Usually, the fraction in the floating point word can be regarded as consisting of several octal or hexadecimal digits. The least significant of these digits can be set to zero by a logical masking operation. Thus the statement

```
      CHOP(p)
```

causes the p least significant octal or hexadecimal digits in the result of each floating point operation to be set to zero. For example, if the computer being used has an IBM 360 long floating point word with 14 hexadecimal digits in the fraction, then CHOP(8) results in simulation of a computer with only 6 hexadecimal digits in the fraction, i.e. a short floating point word. On a computer such as the CDC 6600 with 16 octal digits, CHOP(8) results in about the same accuracy because the remaining 8 octal digits represent the same number of bits as 6 hexadecimal digits.

Some idea of the effect of CHOP on any particular system can be obtained by executing the following statements.

```
long,   t = 1/10
long z, t = 1/10
chop(8)
long,   t = 1/10
long z, t = 1/10
```

The following Fortran subprograms illustrate more details of FLOP and CHOP. The first subprogram is a simplified example of a system-dependent function used within MATLAB itself. The common variable FLP is essentially the first component of the variable FLOP. The common variable CHP is initially zero, but it is set to p by the statement CHOP(p). To shorten the DATA statement, we assume there are only 6 hexadecimal digits. We also assume an extension of Fortran that allows .AND. to be used as a binary operation between two real variables.

```
REAL FUNCTION FLOP(X)
REAL X
INTEGER FLP,CHP
COMMON FLP,CHP
REAL MASK(5)
DATA MASK/ZFFFFFFF0,ZFFFFFF00,ZFFFFF000,ZFFFF0000,ZFFF00000/
FLP = FLP + 1
IF (CHP .EQ. 0) FLOP = X
IF (CHP .GE. 1 .AND. CHP .LE. 5) FLOP = X .AND. MASK(CHP)
IF (CHP .GE. 6) FLOP = 0.0
RETURN
END
```

The following subroutine illustrates a typical use of the previous function within MATLAB. It is a simplified version of the Basic Linear Algebra Subprogram that adds a scalar multiple of one vector to another. We assume here that the vectors are stored with a memory increment of one.

```
   SUBROUTINE SAXPY(N,TR,TI,XR,XI,YR,YI)
   REAL TR,TI,XR(N),XI(N),YR(N),YI(N),FLOP
   IF (N .LE. 0) RETURN
   IF (TR .EQ. 0.0 .AND. TI .EQ. 0.0) RETURN
   DO 10 I = 1, N
      YR(I) = FLOP(YR(I) + TR*XR(I) - TI*XI(I))
      YI(I) = YI(I) + TR*XI(I) + TI*XR(I)
      IF (YI(I) .NE. 0.0D0) YI(I) = FLOP(YI(I))
10 CONTINUE
   RETURN
   END
```

The saxpy operation is perhaps the most fundamental operation within LINPACK. It is used in the computation of the LU, the QR and the SVD factorizations, and in several other places. We see that adding a multiple of one vector with n components to another uses n flops if the vectors are real and between n and 2*n flops if the vectors have nonzero imaginary components.

The permanent MATLAB variable EPS is reset by the statement CHOP(p). Its new value is usually the smallest inverse power of two that satisfies the Fortran logical test

```
          FLOP(1.0+EPS) .GT. 1.0
```

However, if EPS had been directly reset to a larger value, the old value is retained.

12 Communicating with other programs

There are four different ways MATLAB can be used in conjunction with other programs:

```
      -- USER,
      -- EXEC,
      -- SAVE and LOAD,
      -- MATZ, CALL and RETURN .
```

Let us illustrate each of these by the following simple example.

```
      n = 6
      for i = 1:n, for j = 1:n, a(i,j) = abs(i-j);
      A
      X = inv(A)
```

The example A could be introduced into MATLAB by writing the following Fortran subroutine.

```
         SUBROUTINE USER(A,M,N,S,T)
         DOUBLE PRECISION A(1),S,T
         N = IDINT(A(1))
         M = N
         DO 10 J = 1, N
         DO 10 I = 1, N
            K = I + (J-1)*M
            A(K) = IABS(I-J)
      10 CONTINUE
         RETURN
         END
```

This subroutine should be compiled and linked into MATLAB in place of the original version of USER. Then the MATLAB statements

```
      n = 6
      A = user(n)
      X = inv(A)
```

do the job.

The example A could be generated by storing the following text in a file named, say, EXAMPLE .

```
      for i = 1:n, for j = 1:n, a(i,j) = abs(i-j);
```

Then the MATLAB statements

```
n = 6
exec('EXAMPLE',0)
X = inv(A)
```

have the desired effect. The 0 as the optional second parameter of exec indicates that the text in the file should not be printed on the terminal.

The matrices A and X could also be stored in files. Two separate main programs would be involved. The first is:

```
      PROGRAM MAINA
      DOUBLE PRECISION A(10,10)
      N = 6
      DO 10 J = 1, N
      DO 10 I = 1, N
         A(I,J) = IABS(I-J)
   10 CONTINUE
      OPEN(UNIT=1,FILE='A')
      WRITE(1,101) N,N
  101 FORMAT('A   ',2I4)
      DO 20 J = 1, N
         WRITE(1,102) (A(I,J),I=1,N)
   20 CONTINUE
  102 FORMAT(4Z18)
      END
```

The OPEN statement may take different forms on different systems. It attaches Fortran logical unit number 1 to the file named A. The FORMAT number 102 may also be system dependent. This particular one is appropriate for hexadecimal computers with an 8 byte double precision floating point word. Check, or modify, MATLAB subroutine SAVLOD.

After this program is executed, enter MATLAB and give the following statements:

```
load('A')
X = inv(A)
save('X',X)
```

If all goes according to plan, this will read the matrix A from the file A, invert it, store the inverse in X and then write the matrix X on the file X . The following program can then access X .

```
      PROGRAM MAINX
      DOUBLE PRECISION X(10,10)
      OPEN(UNIT=1,FILE='X')
      REWIND 1
      READ (1,101) ID,M,N
  101 FORMAT(A4,2I4)
      DO 10 J = 1, N
         READ(1,102) (X(I,J),I=1,M)
   10 CONTINUE
  102 FORMAT(4Z18)
```

```
         ...
         ...
```

The most elaborate mechanism involves using MATLAB as a subroutine within another program. Communication with the MATLAB stack is accomplished using subroutine MATZ which is distributed with MATLAB, but which is not used by MATLAB itself. The preample of MATZ is:

```
      SUBROUTINE MATZ(A,LDA,M,N,IDA,JOB,IERR)
      INTEGER LDA,M,N,IDA(1),JOB,IERR
      DOUBLE PRECISION A(LDA,N)
C
C     ACCESS MATLAB VARIABLE STACK
C     A IS AN M BY N MATRIX, STORED IN AN ARRAY WITH
C         LEADING DIMENSION LDA.
C     IDA IS THE NAME OF A.
C         IF IDA IS AN INTEGER K LESS THAN 10, THEN THE NAME IS 'A'K
C         OTHERWISE, IDA(1:4) IS FOUR CHARACTERS, FORMAT 4A1.
C     JOB =  0  GET REAL A FROM MATLAB,
C         =  1  PUT REAL A INTO MATLAB,
C         = 10  GET IMAG PART OF A FROM MATLAB,
C         = 11  PUT IMAG PART OF A INTO MATLAB.
C     RETURN WITH NONZERO IERR AFTER MATLAB ERROR MESSAGE.
C
C     USES MATLAB ROUTINES STACKG, STACKP AND ERROR
```

The preample of subroutine MATLAB is:

```
      SUBROUTINE MATLAB(INIT)
C     INIT = 0 FOR FIRST ENTRY, NONZERO FOR SUBSEQUENT ENTRIES
```

To do our example, write the following program:

```
         DOUBLE PRECISION A(10,10),X(10,10)
         INTEGER IDA(4),IDX(4)
         DATA LDA/10/
         DATA IDA/'A',' ',' ',' '/, IDX/'X',' ',' ',' '/
         CALL MATLAB(0)
         N = 6
         DO 10 J = 1, N
         DO 10 I = 1, N
            A(I,J) = IABS(I-J)
      10 CONTINUE
         CALL MATZ(A,LDA,N,N,IDA,1,IERR)
         IF (IERR .NE. 0) GO TO ...
         CALL MATLAB(1)
         CALL MATZ(X,LDA,N,N,IDX,0,IERR)
         IF (IERR .NE. 0) GO TO ...
         ...
         ...
```

When this program is executed, the call to MATLAB(0) produces the MATLAB greeting, then waits for input. The command

```
return
```

sends control back to our example program. The matrix A is generated by the program and sent to the stack by the first call to MATZ. The call to MATLAB(1) produces the MATLAB prompt. Then the statements

```
X = inv(A)
return
```

will invert our matrix, put the result on the stack and go back to our program. The second call to MATZ will retrieve X .

By the way, this matrix X is interesting. Take a look at round(2*(n-1)*X).

13 Acknowledgement.

Most of the work on MATLAB has been carried out at the University of New Mexico, where it is being supported by the National Science Foundation. Additional work has been done during visits to Stanford Linear Accelerator Center, Argonne National Laboratory and Los Alamos Scientific Laboratory, where support has been provided by NSF and the Department of Energy.

14 References

[1] J. J. Dongarra, J. R. Bunch, C. B. Moler and G. W. Stewart, LINPACK Users' Guide, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[2] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, C. B. Moler, Matrix Eigensystem Routines -- EISPACK Guide, Lecture Notes in Computer Science, volume 6, second edition, Springer-Verlag, 1976.

[3] B. S. Garbow, J. M. Boyle, J. J. Dongarra, C. B. Moler, Matrix Eigensystem Routines -- EISPACK Guide Extension, Lecture Notes in Computer Science, volume 51, Springer-Verlag, 1977.

[4] S. Cohen and S. Piper, SPEAKEASY III Reference Manual, Speakeasy Computing Corp., Chicago, Ill., 1979.

[5] J. H. Wilkinson and C. Reinsch, Handbook for Automatic Computation, volume II, Linear Algebra, Springer-Verlag, 1971.

[6] Niklaus Wirth, Algorithms + Data Structures = Programs, Prentice-Hall, 1976.

[7] H. B. Keller and D. Sachs, "Calculations of the Conductivity of a Medium Containing Cylindrical Inclusions", J. Applied Physics 35, 537-538, 1964.

[8] C. B. Moler and C. F. Van Loan, Nineteen Dubious Ways to Compute the Exponential of a Matrix, SIAM Review 20, 801-836, 1979.

[9] G. E. Forsythe, M. A. Malcolm and C. B. Moler, Computer Methods for Mathematical Computations, Prentice-Hall, 1977.

[10] C. B. Moler and D. R. Morrison, "Replacing square roots by Pythagorean sums", University of New Mexico, Computer Science Department, technical report, submitted for publication, 1980.

15 Appendix. The HELP document

```
NEWS  MATLAB NEWS dated May, 1981.
      This describes recent or local changes.
      The new features added since the November,  1980,  printing
      of the Users' Guide include DIARY, EDIT, KRON, MACRO, PLOT,
      RAT, TRIL, TRIU and six element-by-element operations:
            .*   ./   .\   .*.   ./.   .\.
      Some additional  capabilities  have  been  added  to  EXIT,
      RANDOM, RCOND, SIZE and SVD.

INTRO Welcome to MATLAB.

      Here are a few sample statements:

      A = <1 2; 3 4>
      b = <5 6>'
      x = A\b
      <V,D> = eig(A),  norm(A-V*D/V)
      help \ , help eig
      exec('demo',7)

      For more information, see the MATLAB Users' Guide which  is
      contained in file ...  or may be obtained from ... .

HELP  HELP gives assistance.
```

```
      HELP HELP obviously prints this message.
      To see all the HELP messages, list the file ... .

<     < > Brackets used in forming vectors and matrices.
      <6.9 9.64 SQRT(-1)> is a vector with three elements
      separated by blanks.  <6.9, 9.64, sqrt(-1)> is the same
      thing.  <1+I 2-I 3>  and  <1 +I 2 -I 3>  are not the  same.
      The first has three elements, the second has five.
      <11 12 13; 21 22 23>  is a 2 by 3 matrix .   The  semicolon
      ends the first row.

      Vectors and matrices can be used inside < > brackets.
      <A B; C>  is allowed if the number of rows  of   A   equals
      the  number  of rows of  B  and the number of columns of  A
      plus the number of columns of   B   equals  the  number  of
      columns  of   C  .   This  rule  generalizes in a hopefully
      obvious way to allow fairly complicated constructions.

      A = < >  stores an empty matrix in  A , thereby removing it
      from the list of current variables.

      For the use of < and > on the left of  the  =  in  multiple
      assignment statements, see LU, EIG, SVD and so on.

      In WHILE and IF clauses, <>  means  less  than  or  greater
      than,  i.e.  not  equal, < means less than, > means greater
      than, <= means less than or equal, >= means greater than or
      equal.

      For the use of > and < to delineate macros, see MACRO.

>     See < .  Also see MACRO.

(     ( ) Used to indicate precedence in  arithmetic  expressions
      in  the  usual way.  Used to enclose arguments of functions
      in the usual way.  Used to enclose  subscripts  of  vectors
      and  matrices  in  a  manner somewhat more general than the
      usual way.  If  X   and   V  are  vectors,  then   X(V)  is
      <X(V(1)),  X(V(2)),  ...,  X(V(N))> .  The components of  V
      are rounded to nearest integers and used as subscripts.  An
      error  occurs  if  any  such  subscript  is  less than 1 or
      greater than the dimension of  X .  Some examples:
      X(3)  is the third element of  X .
      X(<1 2 3>)  is the first three elements of  X .  So is
      X(<SQRT(2), SQRT(3), 4*ATAN(1)>)  .
```

If X has N components, X(N:-1:1) reverses them.
The same indirect subscripting is used in matrices. If V has M components and W has N components, then A(V,W) is the M by N matrix formed from the elements of A whose subscripts are the elements of V and W . For example...
A(<1,5>,:) = A(<5,1>,:) interchanges rows 1 and 5 of A .

) See (.

= Used in assignment statements and to mean equality in WHILE and IF clauses.

. Decimal point. 314/100, 3.14 and .314E1 are all the same.

Element-by-element multiplicative operations are obtained using .* , ./ , or .\ . For example, C = A ./ B is the matrix with elements c(i,j) = a(i,j)/b(i,j) .

Kronecker tensor products and quotients are obtained with .*. , ./. and .\. . See KRON.

Two or more points at the end of the line indicate continuation. The total line length limit is 1024 characters.

, Used to separate matrix subscripts and function arguments. Used at the end of FOR, WHILE and IF clauses. Used to separate statements in multi-statement lines. In this situation, it may be replaced by semicolon to suppress printing.

; Used inside brackets to end rows.
Used after an expression or statement to suppress printing.
See SEMI.

\ Backslash or matrix left division. \verb!A\B! is roughly the same as INV(A)*B , except it is computed in a different way. If A is an N by N matrix and B is a column vector with N components, or a matrix with several such columns, then X = \verb!A\B! is the solution to the equation A*X = B computed by Gaussian elimination. A warning message is printed if A is badly scaled or nearly singular.
A\EYE produces the inverse of A .

If A is an M by N matrix with M < or > N and B is a column vector with M components, or a matrix with several such columns, then X = \verb!A\B! is the solution in the least squares sense to the under- or overdetermined system of equations A*X = B . The effective rank, K, of A is determined from the QR decomposition with pivoting. A solution X is computed which has at most K nonzero components per column. If K < N this will usually not be the same solution as PINV(A)*B .
A\EYE produces a generalized inverse of A .

If A and B have the same dimensions, then A .\ B has elements a(i,j)\b(i,j) .

Also, see EDIT.

/ Slash or matrix right division. \verb!B/A! is roughly the same as \verb!B*INV(A)!. More precisely, \verb!B/A = (A'\B')'!. See \verb!\!.

IF A and B have the same dimensions, then A ./ B has elements a(i,j)/b(i,j) .

Two or more slashes together on a line indicate a logical end of line. Any following text is ignored.

' Transpose. X' is the complex conjugate transpose of X .
Quote. 'ANY TEXT' is a vector whose components are the MATLAB internal codes for the characters. A quote within the text is indicated by two quotes. See DISP and FILE .

+ Addition. X + Y . X and Y must have the same dimensions.

- Subtraction. X - Y . X and Y must have the same dimensions.

* Matrix multiplication, X*Y . Any scalar (1 by 1 matrix) may multiply anything. Otherwise, the number of columns of X must equal the number of rows of Y .

Element-by-element multiplication is obtained with X .* Y .

The Kronecker tensor product is denoted by X .*. Y .

Powers. X**p is X to the p power. p must be a scalar. If X is a matrix, see FUN .

: Colon. Used in subscripts, FOR iterations and possibly elsewhere.
J:K is the same as <J, J+1, ..., K>
J:K is empty if J > K .
J:I:K is the same as <J, J+I, J+2I, ..., K>
J:I:K is empty if I > 0 and J > K or if I < 0 and J < K .
The colon notation can be used to pick out selected rows, columns and elements of vectors and matrices.
A(:) is all the elements of A, regarded as a single column.
A(:,J) is the J-th column of A
A(J:K) is A(J),A(J+1),...,A(K)
A(:,J:K) is A(:,J),A(:,J+1),...,A(:,K) and so on.
For the use of the colon in the FOR statement, See FOR .

ABS ABS(X) is the absolute value, or complex modulus, of the elements of X .

ANS Variable created automatically when expressions are not assigned to anything else.

ATAN ATAN(X) is the arctangent of X . See FUN .

BASE BASE(X,B) is a vector containing the base B representation of X . This is often used in conjunction with DISPLAY. DISPLAY(X,B) is the same as DISPLAY(BASE(X,B)). For example, DISP(4*ATAN(1),16) prints the hexadecimal representation of pi.

CHAR CHAR(K) requests an input line containing a single character to replace MATLAB character number K in the following table. For example, CHAR(45) replaces backslash. CHAR(-K) replaces the alternate character number K.

K	character	alternate	name
0 - 9	0 - 9	0 - 9	digits
10 - 35	A - Z	a - z	letters
36			blank
37	(	(	lparen
38	)	)	rparen
39	;	;	semi
40	:	|	colon
41	+	+	plus
42	-	-	minus

43	*	*	star
44	/	/	slash
45	\	$	backslash
46	=	=	equal
47	.	.	dot
48	,	,	comma
49	’	"	quote
50	<	[	less
51	>	]	great

CHOL Cholesky factorization. CHOL(X) uses only the diagonal and upper triangle of X . The lower triangular is assumed to be the (complex conjugate) transpose of the upper. If X is positive definite, then R = CHOL(X) produces an upper triangular R so that R’*R = X . If X is not positive definite, an error message is printed.

CHOP Truncate arithmetic. CHOP(P) causes P places to be chopped off after each arithmetic operation in subsequent computations. This means P hexadecimal digits on some computers and P octal digits on others. CHOP(0) restores full precision.

CLEAR Erases all variables, except EPS, FLOP, EYE and RAND.
X = <> erases only variable X . So does CLEAR X .

COND Condition number in 2-norm. COND(X) is the ratio of the largest singular value of X to the smallest.

CONJG CONJG(X) is the complex conjugate of X .

COS COS(X) is the cosine of X . See FUN .

DET DET(X) is the determinant of the square matrix X .

DIAG If V is a row or column vector with N components, DIAG(V,K) is a square matrix of order N+ABS(K) with the elements of V on the K-th diagonal. K = 0 is the main diagonal, K > 0 is above the main diagonal and K < 0 is below the main diagonal. DIAG(V) simply puts V on the main diagonal.
eg. DIAG(-M:M) + DIAG(ONES(2*M,1),1) + DIAG(ONES(2*M,1),-1)
produces a tridiagonal matrix of order 2*M+1 .
IF X is a matrix, DIAG(X,K) is a column vector formed from the elements of the K-th diagonal of X .

DIAG(X) is the main diagonal of X .
DIAG(DIAG(X)) is a diagonal matrix .

DIARY DIARY(’file’) causes a copy of all subsequent terminal input and most of the resulting output to be written on the file. DIARY(0) turns it off. See FILE.

DISP DISPLAY(X) prints X in a compact format. If all the elements of X are integers between 0 and 51, then X is interpreted as MATLAB text and printed accordingly. Otherwise, + , - and blank are printed for positive, negative and zero elements. Imaginary parts are ignored.
DISP(X,B) is the same as DISP(BASE(X,B)).

EDIT There are no editing features available on most installations and EDIT is not a command. However, on a few systems a command line consisting of a single backslash \ will cause the local file editor to be called with a copy of the previous input line. When the editor returns control to MATLAB, it will execute the line again.

EIG Eigenvalues and eigenvectors.
EIG(X) is a vector containing the eigenvalues of a square matrix X .
<V,D> = EIG(X) produces a diagonal matrix D of eigenvalues and a full matrix V whose columns are the corresponding eigenvectors so that X*V = V*D .

ELSE Used with IF .

END Terminates the scope of FOR, WHILE and IF statements. Without END’s, FOR and WHILE repeat all statements up to the end of the line. Each END is paired with the closest previous unpaired FOR or WHILE and serves to terminate its scope. The line

```
FOR I=1:N, FOR J=1:N, A(I,J)=1/(I+J-1); A
```

would cause A to be printed N**2 times, once for each new element. On the other hand, the line

```
FOR I=1:N, FOR J=1:N, A(I,J)=1/(I+J-1); END, END, A
```

will lead to only the final printing of A .
Similar considerations apply to WHILE.
EXIT terminates execution of loops or of MATLAB itself.

EPS Floating point relative accuracy. A permanent variable whose value is initially the distance from 1.0 to the next largest floating point number. The value is changed by CHOP, and other values may be assigned. EPS is used as a default tolerance by PINV and RANK.

EXEC EXEC(’file’,k) obtains subsequent MATLAB input from an external file. The printing of input is controlled by the optional parameter k .
If k = 1 , the input is echoed.
If k = 2 , the MATLAB prompt <> is printed.
If k = 4 , MATLAB pauses before each prompt and waits for a null line to continue.
If k = 0 , there is no echo, prompt or pause. This is the default if the exec command is followed by a semicolon.
If k = 7 , there will be echos, prompts and pauses. This is useful for demonstrations on video terminals.
If k = 3 , there will be echos and prompts, but no pauses. This is the the default if the exec command is not followed by a semicolon.
EXEC(0) causes subsequent input to be obtained from the terminal. An end-of-file has the same effect.
EXEC’s may be nested, i.e. the text in the file may contain EXEC of another file. EXEC’s may also be driven by FOR and WHILE loops.

EXIT Causes termination of a FOR or WHILE loop.
If not in a loop, terminates execution of MATLAB.

EXP EXP(X) is the exponential of X , e to the X . See FUN .

EYE Identity matrix. EYE(N) is the N by N identity matrix. EYE(M,N) is an M by N matrix with 1’s on the diagonal and zeros elsewhere. EYE(A) is the same size as A . EYE with no arguments is an identity matrix of whatever order is appropriate in the context. For example, A + 3*EYE adds 3 to each diagonal element of A .

FILE The EXEC, SAVE, LOAD, PRINT and DIARY functions access files. The ’file’ parameter takes different forms for different operating systems. On most systems, ’file’ may be a string of up to 32 characters in quotes. For example, SAVE(’A’) or EXEC(’matlab/demo.exec’) . The string will be used as the name of a file in the local operating system.
On all systems, ’file’ may be a positive integer k less than 10 which will be used as a FORTRAN logical unit

number. Some systems then automatically access a file with a name like FORT.k or FORk.DAT. Other systems require a file with a name like FT0kF001 to be assigned to unit k before MATLAB is executed. Check your local installation for details.

FLOPS Count of floating point operations.

FLOPS is a permanently defined row vector with two elements. FLOPS(1) is the number of floating point operations counted during the previous statement. FLOPS(2) is a cumulative total. FLOPS can be used in the same way as any other vector. FLOPS(2) = 0 resets the cumulative total. In addition, FLOPS(1) will be printed whenever a statement is terminated by an extra comma. For example,

```
X = INV(A);,
or
COND(A),   (as the last statement on the line).
```

HELP FLPS gives more details.

FLPS More detail on FLOPS.

It is not feasible to count absolutely all floating point operations, but most of the important ones are counted. Each multiply and add in a real vector operation such as a dot product or a 'saxpy' counts one flop. Each multiply and add in a complex vector operation counts two flops. Other additions, subtractions and multiplications count one flop each if the result is real and two flops if it is not. Real divisions count one and complex divisions count two. Elementary functions count one if real and two if complex. Some examples. If A and B are real N by N matrices, then

```
A + B  counts N**2 flops,
A*B    counts N**3 flops,
A**100 counts 99*N**3 flops,
LU(A)  counts roughly (1/3)*N**3 flops.
```

FOR Repeat statements a specific number of times.

```
FOR variable = expr, statement, ..., statement, END
```

The END at the end of a line may be omitted. The comma before the END may also be omitted. The columns of the expression are stored one at a time in the variable and then the following statements, up to the END, are executed. The expression is often of the form X:Y, in which case its columns are simply scalars. Some examples (assume N has already been assigned a value).

```
FOR I = 1:N, FOR J = 1:N, A(I,J) = 1/(I+J-1);
```

```
     FOR J = 2:N-1, A(J,J) = J; END; A
     FOR S = 1.0: -0.1: 0.0, ...  steps S with increments of -0.1 .
     FOR E = EYE(N), ...   sets  E  to the unit N-vectors.
     FOR V = A, ...   has the same effect as
     FOR J = 1:N, V = A(:,J); ...  except J is also set here.
```

FUN   For matrix arguments X , the functions SIN, COS, ATAN, SQRT, LOG, EXP and X**p are computed using eigenvalues D and eigenvectors V . If <V,D> = EIG(X) then f(X) = V*f(D)/V . This method may give inaccurate results if V is badly conditioned. Some idea of the accuracy can be obtained by comparing X**1 with X .
For vector arguments, the function is applied to each component.

HESS Hessenberg form. The Hessenberg form of a matrix is zero below the first subdiagonal. If the matrix is symmetric or Hermitian, the form is tridiagonal. <P,H> = HESS(A) produces a unitary matrix P and a Hessenberg matrix H so that A = P*H*P'. By itself, HESS(A) returns H.

HILB Inverse Hilbert matrix. HILB(N) is the inverse of the N by N matrix with elements 1/(i+j-1), which is a famous example of a badly conditioned matrix. The result is exact for N less than about 15, depending upon the computer.

IF   Conditionally execute statements. Simple form...
IF expression rop expression, statements
where rop is =, <, >, <=, >=, or <> (not equal) . The statements are executed once if the indicated comparison between the real parts of the first components of the two expressions is true, otherwise the statements are skipped.
Example.

```
     IF ABS(I-J) = 1, A(I,J) = -1;
```

More complicated forms use END in the same way it is used with FOR and WHILE and use ELSE as an abbreviation for END,
IF expression not rop expression . Example:

```
     FOR I = 1:N, FOR J = 1:N, ...
        IF I = J, A(I,J) = 2; ELSE IF ABS(I-J) = 1, A(I,J) = -1; ...
        ELSE A(I,J) = 0;
```

An easier way to accomplish the same thing is

```
     A = 2*EYE(N);
     FOR I = 1:N-1, A(I,I+1) = -1; A(I+1,I) = -1;
```

IMAG IMAG(X) is the imaginary part of X .

INV INV(X) is the inverse of the square matrix X . A warning message is printed if X is badly scaled or nearly singular.

KRON KRON(X,Y) is the Kronecker tensor product of X and Y . It is also denoted by X .*. Y . The result is a large matrix formed by taking all possible products between the elements of X and those of Y . For example, if X is 2 by 3, then X .*. Y is

```
      < x(1,1)*Y  x(1,2)*Y  x(1,3)*Y
        x(2,1)*Y  x(2,2)*Y  x(2,3)*Y >
```

The five-point discrete Laplacian for an n-by-n grid can be generated by

```
      T = diag(ones(n-1,1),1);  T = T + T';  I = EYE(T);
      A = T.*.I + I.*.T - 4*EYE;
```

Just in case they might be useful, MATLAB includes constructions called Kronecker tensor quotients, denoted by X ./. Y and X .\. Y . They are obtained by replacing the elementwise multiplications in X .*. Y with divisions.

LINES An internal count is kept of the number of lines of output since the last input. Whenever this count approaches a limit, the user is asked whether or not to suppress printing until the next input. Initially the limit is 25. LINES(N) resets the limit to N .

LOAD LOAD('file') retrieves all the variables from the file . See FILE and SAVE for more details. To prepare your own file for LOADing, change the READs to WRITEs in the code given under SAVE.

LOG LOG(X) is the natural logarithm of X . See FUN . Complex results are produced if X is not positive, or has nonpositive eigenvalues.

LONG Determine output format. All computations are done in complex arithmetic and double precision if it is available. SHORT and LONG merely switch between different output formats.
SHORT Scaled fixed point format with about 5 digits.

```
      LONG     Scaled fixed point format with about 15 digits.
      SHORT E  Floating point format with about 5 digits.
      LONG E   Floating point format with about 15 digits.
      LONG Z   System dependent format, often hexadecimal.

LU    Factors from Gaussian elimination.  <L,U> = LU(X)  stores a
      upper triangular matrix in  U  and a 'psychologically lower
      triangular matrix', i.e. a product of lower triangular  and
      permutation matrices, in L , so that  X = L*U .  By itself,
      LU(X) returns the output from CGEFA .

MACRO The macro facility involves text and inward pointing  angle
      brackets.  If  STRING  is  the  source  text for any MATLAB
      expression or statement, then
            t = 'STRING';
      encodes the text as a vector of integers  and  stores  that
      vector in  t .  DISP(t) will print the text and
            >t<
      causes the text to be interpreted, either as a statement or
      as a factor in an expression.  For example
            t = '1/(i+j-1)';
            disp(t)
            for i = 1:n, for j = 1:n, a(i,j) = >t<;
      generates the Hilbert matrix of order n.
      Another example showing indexed text,
            S = <'x = 3            '
                 'y = 4            '
                 'z = sqrt(x*x+y*y)'>
            for k = 1:3, >S(k,:)<
      It is necessary that the strings making up  the  "rows"  of
      the "matrix"  S  have the same lengths.

MAGIC Magic square.  MAGIC(N) is an N  by  N  matrix  constructed
      from  the integers 1 through N**2 with equal row and column
      sums.

NORM  For matrices..
      NORM(X)  is the largest singular value of  X .
      NORM(X,1)  is the 1-norm of  X .
      NORM(X,2)  is the same as NORM(X) .
      NORM(X,'INF')  is the infinity norm of  X .
      NORM(X,'FRO')  is the F-norm, i.e.  SQRT(SUM(DIAG(X'*X))) .
      For vectors..
      NORM(V,P) = (SUM(V(I)**P))**(1/P) .
      NORM(V) = NORM(V,2) .
```

NORM(V,'INF') = MAX(ABS(V(I))) .

ONES All ones. ONES(N) is an N by N matrix of ones. ONES(M,N) is an M by N matrix of ones . ONES(A) is the same size as A and all ones .

ORTH Orthogonalization. Q = ORTH(X) is a matrix with orthonormal columns, i.e. Q'*Q = EYE, which span the same space as the columns of X .

PINV Pseudoinverse. X = PINV(A) produces a matrix X of the same dimensions as A' so that A*X*A = A , X*A*X = X and AX and XA are Hermitian . The computation is based on SVD(A) and any singular values less than a tolerance are treated as zero. The default tolerance is NORM(SIZE(A),'inf')*NORM(A)*EPS. This tolerance may be overridden with X = PINV(A,tol). See RANK.

PLOT PLOT(X,Y) produces a plot of the elements of Y against those of X . PLOT(Y) is the same as PLOT(1:n,Y) where n is the number of elements in Y . PLOT(X,Y,P) or PLOT(X,Y,p1,...,pk) passes the optional parameter vector P or scalars p1 through pk to the plot routine. The default plot routine is a crude printer-plot. It is hoped that an interface to local graphics equipment can be provided.
An interesting example is

```
      t = 0:50;
      PLOT( t.*cos(t), t.*sin(t) )
```

POLY Characteristic polynomial.
If A is an N by N matrix, POLY(A) is a column vector with N+1 elements which are the coefficients of the characteristic polynomial, DET(lambda*EYE - A) .
If V is a vector, POLY(V) is a vector whose elements are the coefficients of the polynomial whose roots are the elements of V . For vectors, ROOTS and POLY are inverse functions of each other, up to ordering, scaling, and roundoff error.
ROOTS(POLY(1:20)) generates Wilkinson's famous example.

PRINT PRINT('file',X) prints X on the file using the current format determined by SHORT, LONG Z, etc. See FILE.

PROD PROD(X) is the product of all the elements of X .

QR Orthogonal-triangular decomposition.
<Q,R> = QR(X) produces an upper triangular matrix R of the same dimension as X and a unitary matrix Q so that X = Q*R .
<Q,R,E> = QR(X) produces a permutation matrix E , an upper triangular R with decreasing diagonal elements and a unitary Q so that X*E = Q*R .
By itself, QR(X) returns the output of CQRDC . TRIU(QR(X)) is R .

RAND Random numbers and matrices. RAND(N) is an N by N matrix with random entries. RAND(M,N) is an M by N matrix with random entries. RAND(A) is the same size as A . RAND with no arguments is a scalar whose value changes each time it is referenced.
Ordinarily, random numbers are uniformly distributed in the interval (0.0,1.0) . RAND('NORMAL') switches to a normal distribution with mean 0.0 and variance 1.0 . RAND('UNIFORM') switches back to the uniform distribution. RAND('SEED') returns the current value of the seed for the generator. RAND('SEED',n) sets the seed to n . RAND('SEED',0) resets the seed to 0, its value when MATLAB is first entered.

RANK Rank. K = RANK(X) is the number of singular values of X that are larger than NORM(SIZE(X),'inf')*NORM(X)*EPS.
K = RANK(X,tol) is the number of singular values of X that are larger than tol .

RCOND RCOND(X) is an estimate for the reciprocal of the condition of X in the 1-norm obtained by the LINPACK condition estimator. If X is well conditioned, RCOND(X) is near 1.0 . If X is badly conditioned, RCOND(X) is near 0.0 .
<R, Z> = RCOND(A) sets R to RCOND(A) and also produces a vector Z so that

NORM(A*Z,1) = R*NORM(A,1)*NORM(Z,1)

So, if RCOND(A) is small, then Z is an approximate null vector.

RAT An experimental function which attempts to remove the roundoff error from results that should be "simple" rational numbers.
RAT(X) approximates each element of X by a continued fraction of the form

```
          a/b = d1 + 1/(d2 + 1/(d3 + ... + 1/dk))
```

with k <= len, integer di and abs(di) <= max . The default values of the parameters are len = 5 and max = 100. RAT(len,max) changes the default values. Increasing either len or max increases the number of possible fractions. <A,B> = RAT(X) produces integer matrices A and B so that

```
          A ./ B  =  RAT(X)
```

Some examples:

```
      long
      T = hilb(6), X = inv(T)
      <A,B> = rat(X)
      H = A ./ B, S = inv(H)

      short e
      d = 1:8,  e = ones(d),  A = abs(d'*e - e'*d)
      X = inv(A)
      rat(X)
      display(ans)
```

REAL REAL(X) is the real part of X .

RETURN From the terminal, causes return to the operating system or other program which invoked MATLAB. From inside an EXEC, causes return to the invoking EXEC, or to the terminal.

RREF RREF(A) is the reduced row echelon form of the rectangular matrix. RREF(A,B) is the same as RREF(<A,B>) .

ROOTS Find polynomial roots. ROOTS(C) computes the roots of the polynomial whose coefficients are the elements of the vector C . If C has N+1 components, the polynomial is C(1)*X**N + ... + C(N)*X + C(N+1) . See POLY.

ROUND ROUND(X) rounds the elements of X to the nearest integers.

SAVE SAVE('file') stores all the current variables in a file. SAVE('file',X) saves only X . See FILE .

The variables may be retrieved later by LOAD('file') or by your own program using the following code for each matrix. The lines involving XIMAG may be eliminated if everything is known to be real.

```
      attach lunit to 'file'
      REAL or DOUBLE PRECISION XREAL(MMAX,NMAX)
      REAL or DOUBLE PRECISION XIMAG(MMAX,NMAX)
      READ(lunit,101) ID,M,N,IMG
      DO 10 J = 1, N
         READ(lunit,102) (XREAL(I,J), I=1,M)
         IF (IMG .NE. 0) READ(lunit,102) (XIMAG(I,J),I=1,M)
   10 CONTINUE
```

The formats used are system dependent. The following are typical. See SUBROUTINE SAVLOD in your local implementation of MATLAB.

```
  101 FORMAT(4A1,3I4)
  102 FORMAT(4Z18)
  102 FORMAT(4O20)
  102 FORMAT(4D25.18)
```

SCHUR Schur decomposition. <U,T> = SCHUR(X) produces an upper triangular matrix T , with the eigenvalues of X on the diagonal, and a unitary matrix U so that X = U*T*U' and U'*U = EYE . By itself, SCHUR(X) returns T .

SHORT See LONG .

SEMI Semicolons at the end of lines will cause, rather than suppress, printing. A second SEMI restores the initial interpretation.

SIN SIN(X) is the sine of X . See FUN .

SIZE If X is an M by N matrix, then SIZE(X) is <M, N> .
Can also be used with a multiple assignment,

```
      <M, N> = SIZE(X) .
```

SQRT SQRT(X) is the square root of X . See FUN . Complex results are produced if X is not positive, or has nonpositive eigenvalues.

STOP Use EXIT instead.

SUM SUM(X) is the sum of all the elements of X . SUM(DIAG(X)) is the trace of X .

SVD Singular value decomposition. <U,S,V> = SVD(X) produces a diagonal matrix S , of the same dimension as X and with nonnegative diagonal elements in decreasing order, and unitary matrices U and V so that X = U*S*V' .
By itself, SVD(X) returns a vector containing the singular values.
<U,S,V> = SVD(X,0) produces the "economy size" decomposition. If X is m by n with m > n, then only the first n columns of U are computed and S is n by n .

TRIL Lower triangle. TRIL(X) is the lower triangular part of X. TRIL(X,K) is the elements on and below the K-th diagonal of X. K = 0 is the main diagonal, K > 0 is above the main diagonal and K < 0 is below the main diagonal.

TRIU Upper triangle. TRIU(X) is the upper triangular part of X. TRIU(X,K) is the elements on and above the K-th diagonal of X. K = 0 is the main diagonal, K > 0 is above the main diagonal and K < 0 is below the main diagonal.

USER Allows personal Fortran subroutines to be linked into MATLAB . The subroutine should have the heading

```
         SUBROUTINE USER(A,M,N,S,T)
         REAL or DOUBLE PRECISION A(M,N),S,T
```

The MATLAB statement Y = USER(X,s,t) results in a call to the subroutine with a copy of the matrix X stored in the argument A , its column and row dimensions in M and N , and the scalar parameters s and t stored in S and T . If s and t are omitted, they are set to 0.0 . After the return, A is stored in Y . The dimensions M and N may be reset within the subroutine. The statement Y = USER(K) results in a call with M = 1, N = 1 and A(1,1) = FLOAT(K) . After the subroutine has been written, it must be compiled and linked to the MATLAB object code within the local operating system.

WHAT Lists commands and functions currently available.

WHILE Repeat statements an indefinite number of times.

WHILE expr rop expr, statement, ..., statement, END
where rop is =, <, >, <=, >=, or <> (not equal) . The END at the end of a line may be omitted. The comma before the END may also be omitted. The commas may be replaced by semicolons to avoid printing. The statements are repeatedly executed as long as the indicated comparison between the real parts of the first components of the two expressions is true. Example (assume a matrix A is already defined).

```
E = 0*A; F = E + EYE; N = 1;
WHILE NORM(E+F-E,1) > 0, E = E + F; F = A*F/N; N = N + 1;
E
```

WHO   Lists current variables.

WHY   Provides succinct answers to any questions.

16 Update to MATLAB Users' Guide, May, 1981

The following additions to the MATLAB HELP file have been made since the November, 1980, printing of the Users' Guide.

NEWS MATLAB NEWS dated May, 1981.
The new features added since the November, 1980, printing of the Users' Guide include DIARY, EDIT, KRON, MACRO, PLOT, RAT, TRIL, TRIU and six element-by-element operations:

```
.*   ./   .\   .*.   ./.   .\.
```

Some additional capabilities have been added to EXIT, RANDOM, RCOND, SIZE and SVD.

.   Decimal point. 314/100, 3.14 and .314E1 are all the same.

Element-by-element multiplicative operations are obtained using .* , ./ , or .\ . For example, C = A ./ B is the matrix with elements c(i,j) = a(i,j)/b(i,j) .

Kronecker tensor products and quotients are obtained with .*. , ./. and .\. . See KRON.

Two or more points at the end of the line indicate continuation. The total line length limit is 1024 characters.

DIARY DIARY(’file’) causes a copy of all subsequent terminal input and most of the resulting output to be written on the file. DIARY(0) turns it off. See FILE.

EDIT There are no editing features available on most installations and EDIT is not a command. However, on a few systems a command line consisting of a single backslash \ will cause the local file editor to be called with a copy of the previous input line. When the editor returns control to MATLAB, it will execute the line again.

EXIT Causes termination of a FOR or WHILE loop.
If not in a loop, terminates execution of MATLAB.

KRON KRON(X,Y) is the Kronecker tensor product of X and Y . It is also denoted by X .*. Y . The result is a large matrix formed by taking all possible products between the elements of X and those of Y . For example, if X is 2 by 3, then X .*. Y is

```
< x(1,1)*Y  x(1,2)*Y  x(1,3)*Y
  x(2,1)*Y  x(2,2)*Y  x(2,3)*Y >
```

The five-point discrete Laplacian for an n-by-n grid can be generated by

```
T = diag(ones(n-1,1),1);  T = T + T’;  I = EYE(T);
A = T.*.I + I.*.T - 4*EYE;
```

Just in case they might be useful, MATLAB includes constructions called Kronecker tensor quotients, denoted by X ./. Y and X .\. Y . They are obtained by replacing the elementwise multiplications in X .*. Y with divisions.

MACRO The macro facility involves text and inward pointing angle brackets. If STRING is the source text for any MATLAB expression or statement, then

```
t = ’STRING’;
```

encodes the text as a vector of integers and stores that vector in t . DISP(t) will print the text and

```
>t<
```

causes the text to be interpreted, either as a statement or as a factor in an expression. For example

```
t = ’1/(i+j-1)’;
disp(t)
```

```
      for i = 1:n, for j = 1:n, a(i,j) = >t<;
```

generates the Hilbert matrix of order n.
Another example showing indexed text,

```
      S = <'x = 3           '
           'y = 4           '
           'z = sqrt(x*x+y*y)'>
      for k = 1:3, >S(k,:)<
```

It is necessary that the strings making up the "rows" of the "matrix" S have the same lengths.

PLOT PLOT(X,Y) produces a plot of the elements of Y against those of X . PLOT(Y) is the same as PLOT(1:n,Y) where n is the number of elements in Y . PLOT(X,Y,P) or PLOT(X,Y,p1,...,pk) passes the optional parameter vector P or scalars p1 through pk to the plot routine. The default plot routine is a crude printer-plot. It is hoped that an interface to local graphics equipment can be provided.
An interesting example is

```
      t = 0:50;
      PLOT( t.*cos(t), t.*sin(t) )
```

RAND Random numbers and matrices. RAND(N) is an N by N matrix with random entries. RAND(M,N) is an M by N matrix with random entries. RAND(A) is the same size as A . RAND with no arguments is a scalar whose value changes each time it is referenced.
Ordinarily, random numbers are uniformly distributed in the interval (0.0,1.0) . RAND('NORMAL') switches to a normal distribution with mean 0.0 and variance 1.0 . RAND('UNIFORM') switches back to the uniform distribution. RAND('SEED') returns the current value of the seed for the generator. RAND('SEED',n) sets the seed to n . RAND('SEED',0) resets the seed to 0, its value when MATLAB is first entered.

RAT An experimental function which attempts to remove the roundoff error from results that should be "simple" rational numbers.
RAT(X) approximates each element of X by a continued fraction of the form

```
          a/b = d1 + 1/(d2 + 1/(d3 + ... + 1/dk))
```

with k <= len, integer di and abs(di) <= max . The default values of the parameters are len = 5 and max = 100.

RAT(len,max) changes the default values. Increasing either len or max increases the number of possible fractions.
<A,B> = RAT(X) produces integer matrices A and B so that

```
          A ./ B  =  RAT(X)
```

Some examples:

```
      long
      T = hilb(6), X = inv(T)
      <A,B> = rat(X)
      H = A ./ B, S = inv(H)

      short e
      d = 1:8,  e = ones(d),  A = abs(d'*e - e'*d)
      X = inv(A)
      rat(X)
      display(ans)
```

RCOND RCOND(X) is an estimate for the reciprocal of the condition of X in the 1-norm obtained by the LINPACK condition estimator. If X is well conditioned, RCOND(X) is near 1.0 . If X is badly conditioned, RCOND(X) is near 0.0 .
<R, Z> = RCOND(A) sets R to RCOND(A) and also produces a vector Z so that

```
          NORM(A*Z,1) = R*NORM(A,1)*NORM(Z,1)
```

So, if RCOND(A) is small, then Z is an approximate null vector.

SIZE If X is an M by N matrix, then SIZE(X) is <M, N> .
Can also be used with a multiple assignment,

```
      <M, N> = SIZE(X) .
```

SVD Singular value decomposition. <U,S,V> = SVD(X) produces a diagonal matrix S , of the same dimension as X and with nonnegative diagonal elements in decreasing order, and unitary matrices U and V so that X = U*S*V' .
By itself, SVD(X) returns a vector containing the singular values.
<U,S,V> = SVD(X,0) produces the "economy size" decomposition. If X is m by n with m > n, then only the first n columns of U are computed and S is n by n .

```
TRIL  Lower triangle.  TRIL(X) is the lower triangular part of X.
      TRIL(X,K) is the elements on and below the K-th diagonal of
      X.  K = 0 is the main diagonal, K > 0  is  above  the  main
      diagonal and K < 0 is below the main diagonal.

TRIU  Upper triangle.  TRIU(X) is the upper triangular part of X.
      TRIU(X,K) is the elements on and above the K-th diagonal of
      X.  K = 0 is the main diagonal, K > 0  is  above  the  main
      diagonal and K < 0 is below the main diagonal.
```