6371

Burroughs

Large Systems NUMERALS

NUMERICAL ANALYSIS PROGRAM LIBRARY REFERENCE MANUAL

PRICED ITEM

Printed in U.S.A. July 1977 1076981

Burroughs

Hans Vlems
nov '83

Large Systems
NUMERALS

NUMERICAL ANALYSIS PROGRAM LIBRARY REFERENCE MANUAL

Copyright © 1973, 1974, 1975, Burroughs Corporation, Detroit, Michigan 48232
AA 430061 AA 482489 AA 603654 AA 709294

PRICED ITEM

Printed in U.S.A. July 1977 1076981

Burroughs believes that the application package described in this manual is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, can be accepted for any consequences arising out of the use of this material, including loss of profit, indirect, special, or consequential damages. There are no warranties which extend beyond the program specification.

The Customer should exercise care to assure that use of the application package will be in full compliance with laws, rules and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change. Revisions may be issued from time to time to advise of changes and/or additions.

This reprint includes information released under the following PCN:

PCN 1076981-001 (9-23-75)

Any comments or suggestions regarding this publication should be forwarded to Documentation Department, Business Management and Scientific Systems, Burroughs Corporation, Burroughs Place, Detroit, Michigan 48232.

Table of Contents

Section	Title	Page
	INTRODUCTION	ix
	Contents of the Library	ix
	Use of the Library	x
1	MATRICES	1-1
	Linear Equations, Matrix Inversion, and Determinants	1-1
	Real Matrices - LINPAC	1-1
	LINEQN Procedure	1-4
	DETLINEQN Procedure	1-5
	LINEQNIMPRV Procedure	1-6
	DETLINEQNIMPRV Procedure	1-7
	LINEQNS Procedure	1-8
	DETLINEQNS Procedure	1-9
	INVERSE Procedure	1-10
	INVWITHIMPRV Procedure	1-11
	DETOFA Procedure	1-12
	INVERSEOVERA Procedure	1-13
	LEASTSQUARES Procedure	1-14
	LEASTSQUARESIMPRV Procedure	1-16
	Symmetric Matrices - SYMLINPAC	1-18
	SYMLINEQN Procedure	1-19
	SYMLINEQNIMPRV Procedure	1-20
	SYMLINEQNS Procedure	1-21
	SYMINVWITHIMPRV Procedure	1-22
	SYMBANDLINEQN Procedure	1-23
	SYMINVERSEOVERA Procedure	1-25
	Complex Matrices	1-26
	CXLINEQN Procedure	1-27
	Eigenvalues and Eigenvectors	1-29
	Real Matrices	1-31
	NONSYMEIGENVALUES Procedure	1-31
	NONSYMEIGENVECTORS Procedure	1-33
	Symmetric Matrices	1-35
	SYMEIGENVALUES Procedure	1-35
	SYMEIGENVECTORS Procedure	1-36
	SYMBANDEIGENVALUES Procedure	1-37
	Complex Matrices	1-39
	COMPLEXEIGENVALUES Procedure	1-39
	COMPLEXEIGENVECTORS Procedure	1-41
	INVARIANTSUBSPACES Procedure	1-43
	Matrix Arithmetic	1-45
	Square Matrices	1-45
	MATPLUSMAT Procedure	1-45
	MATMINUSMAT Procedure	1-46
	MATTIMESMAT Procedure	1-47
	TRANSPOSE Procedure	1-48
	ATIMESATRANSPOSE Procedure	1-49
	ATIMESBTRANSPOSE Procedure	1-50
	MATTIMESVEC Procedure	1-51

Table of Contents (cont)

Section	Title	Page
1 (cont)	VECTIMESMAT Procedure	1-52
	VECPLUSRVEC Procedure	1-53
	MATPLUSDIAG Procedure	1-54
	MATMINUSDIAG Procedure	1-55
	MATTIMESDIAG Procedure	1-56
	DIAGTIMESMAT Procedure	1-57
	RPLUSMAT Procedure	1-58
	RMINUSMAT Procedure	1-59
	RTIMESMAT Procedure	1-60
	MATCOPY Procedure	1-61
	Rectangular Matrices	1-62
	RECMATPLUSMAT Procedure	1-62
	RECMATMINUSMAT Procedure	1-63
	RECMATTIMESMAT Procedure	1-64
2	POLYNOMIALS	2-1
	Roots of Polynomials	2-1
	REALPOLYZEROFINDER Procedure	2-1
	COMPLEXPOLYZEROFINDER Procedure	2-2
	Operations on Polynomials	2-3
	POLYADD Procedure	2-3
	POLYSUB Procedure	2-4
	POLYMUL Procedure	2-5
	POLYDIV Procedure	2-6
	POLYVAL Procedure	2-7
	POLYDER Procedure	2-8
	POLYINT Procedure	2-9
	POLYPOP Procedure	2-10
	POLYMAD Procedure	2-11
	POLYHCR Procedure	2-12
	POLYGCD Procedure	2-13
	POLYNCRM Procedure	2-14
	POLYMCV Procedure	2-15
	POLYSOR Procedure	2-16
3	ORDINARY DIFFERENTIAL EQUATIONS	3-1
	DIFEQNS Procedure	3-1
	SIMPLEDIFEQNS Procedure	3-5
4	NUMERICAL INTEGRATION	4-1
	Single Integration	4-1
	ROMBERGINT Procedure	4-1
	SIMPSONINT Procedure	4-2
	BULIRSCHINT Procedure	4-3
	Multiple Integration	4-5
	SIMPS Procedure	4-5
	DOUBLEINTEGRAL Procedure	4-6
	TRIPLEINTEGRAL Procedure	4-7

Section	Title	Page
5	SPECIAL FUNCTIONS	5-1
	JBESS Procedure	5-1
	IBESS Procedure	5-2
	CXJBES Procedure	5-3
	EL1 Procedure	5-4
	CEL1 Procedure	5-5
	EL2 Procedure	5-6
	CEL2 Procedure	5-7
	EL3 Procedure	5-8
	CEL3 Procedure	5-9
	CEL Procedure	5-10
6	MISCELLANEOUS	6-1
	PLOTTER Procedure	6-1
	READMATRIX Procedure	6-3
	READSYMMATRIX Procedure	6-4
7	FORTRAN COMPATIBLE PROCEDURES	7-1
	FRLPLY Procedure	7-2
	FCXPLY Procedure	7-3
	FRMBRG Procedure	7-4
	FSMPSN Procedure	7-5
	FBULIR Procedure	7-6
	FDBINT Procedure	7-7
	FTRINT Procedure	7-8
8	FORTRAN MATRICES	8-1
	Linear Equations, Matrix Inversion, and Determinants	8-1
	LINEQN Subroutine	8-2
	LNQPRV Subroutine	8-3
	LNEQNS Subroutine	8-5
	SYMBLQ Subroutine	8-6
	SYMLNQ Subroutine	8-6C
	SYMLQP Subroutine	8-6E
	LLSQSE Subroutine	8-6G
	LLSQPR Subroutine	8-6J
	DTLQPR Subroutine	8-6M
	SYMLQS Subroutine	8-6N
	SYINPR Subroutine	8-6P
	LINQNS Subroutine	8-6Q
	CXLINQ Subroutine	8-6R
	INVERT Subroutine	8-6T
	INVERA Subroutine	8-6U
	INVEPR Subroutine	8-6V
	DETOFA Subroutine	8-6W
	SINVRA Subroutine	8-6X
	Eigenvalues and Eigenvectors	8-7

Table of Contents (cont)

Section	Title	Page
8 (cont)	SYMVAL Subroutine	8-8
	SYMVEC Subroutine	8-9
	EIGVAL Subroutine	8-11
	EIGVEC Subroutine	8-13
	CEIGVL Subroutine	8-15
	CEIGVC Subroutine	8-17
	HREGVL Subroutine	8-19
	HREGVC Subroutine	8-20
	MXEGVL Subroutine	8-22
	MNEGVL Subroutine	8-24
	Matrix Arithmetic	8-26
	MMULTM Subroutine	8-27
	TRANSP Subroutine	8-28
	AMULTR Subroutine	8-29
	AMULBT Subroutine	8-30
	MATVEC Subroutine	8-31
	VECMAT Subroutine	8-32
	VPLURV Subroutine	8-33
	MMINDG Subroutine	8-34
	MMULTD Subroutine	8-35
	DMULTM Subroutine	8-36
	RPLUSM Subroutine	8-37
	RMINM Subroutine	8-38
	RMULTM Subroutine	8-39
	MCOPY Subroutine	8-40
	RECMPM Subroutine	8-41
	RECMMM Subroutine	8-42
	RECMMU Subroutine	8-43
9	INTERPOLATION	9-1
	General	9-1
	Naming Convention	9-2
	CSDDE Subroutine	9-4
	CSIE Subroutine	9-6
	CSDDU Subroutine	9-8
	CSIU Subroutine	9-10
	CSDDE2 Subroutine	9-12
	CSIE2 Subroutine	9-14
	CSDDU2 Subroutine	9-16
	CSIU2 Subroutine	9-18
10	DIFFERENTIAL EQUATIONS, NUMERICAL INTEGRATION, AND DIFFERENTIATION	10-1
	Differential Equations	10-1
	DESP Subroutine	10-1
	Numerical Integration and Differentiation	10-3
	NUMINT Subroutine	10-4
	NUMDIF Subroutine	10-5

Section	Title	Page
11	ZEROES AND EXTREMA	11-1
	Solution of Nonlinear Equations	11-1
	NLNEQN Subroutine	11-1
	Zeroes of Functions of a Single Variable	11-3
	ZEROML Subroutine	11-3
	ZEROSS Subroutine	11-5
	ZERONT Subroutine	11-7
	Extrema of Functions	11-9
	OPTHKJ Subroutine	11-9
	Nonlinear Programming	11-11
	PENHKJ Subroutine	11-11
12	FORTRAN POLYNOMIALS	12-1
	Roots of Polynomials	12-1
	RLPOLY Subroutine	12-1
	CXPOLY Subroutine	12-3
	Operations on Polynomials	12-5
	POLADD Subroutine	12-5
	POLSUB Subroutine	12-6
	POLMUL Subroutine	12-7
	POLDIV Subroutine	12-8
	POLDER Subroutine	12-9
	POLINT Subroutine	12-10
	POLHOR Subroutine	12-11
	POLMAD Subroutine	12-12
	POLPOP Subroutine	12-13
	POLMOV Subroutine	12-14
	POLNOR Subroutine	12-15
	POLSOR Subroutine	12-16
	POLGCD Subroutine	12-17
13	FORTRAN SPECIAL FUNCTIONS	13-1
	CXBESJ Subroutine	13-1
	BESSFJ Function	13-3
14	FORTRAN NUMERICAL INTEGRATION	14-1
	Single Integration	14-1
	SIMINT Function	14-1
	BULINT Function	14-2
	ROMINT Function	14-3
	Multiple Integration	14-4
	DBLINT Function	14-4
	TPLINT Function	14-5
15	FOURIER TRANSFORMS	15-1
	General	15-1
	FFTF Procedure	15-2

Section	Title	Page
15 (cont)	FFTR Procedure	15-3
	SINCOS Procedure	15-4
	BITREV2 Procedure	15-5
APPENDIX A	PROGRAM EXAMPLES	A-1
APPENDIX B	BIBLIOGRAPHY	B-1

INTRODUCTION

The Burroughs B 7700/B 6700 NUMERALS Numerical Analysis Program Library is a collection of more than 200 FORTRAN subroutines and ALGOL procedures in the area of numerical mathematics. These programs provide a basic computational capability which may be easily interfaced with user programs to solve many problems in engineering, science, and industry. Program calling sequences are designed for maximum ease of use and are standardized in each section. A number of programs utilize the hardware-based double precision capability. The areas of numerical mathematics included are:

a. Matrices.

b. Polynomials.

c. Numerical solution of systems of ordinary differential equations.

d. Numerical integration and differentiation.

e. Standard mathematical functions.

f. Interpolation and curve fitting.

g. Zeroes and extrema of nonlinear functions.

Where feasible, the library includes only one method per mathematical problem area. This is most notably true in the areas of differential equations, roots of polynomials, and eigenvalues and eigenvectors of matrices, for which there are algorithms recently devised that are demonstrably superior to older methods. In other cases, where no single program has all the desired characteristics, such as accuracy or efficiency, an effort is made to ensure that each desirable characteristic is represented by one or more programs. For example, there are three ALGOL matrix inversion routines: INVERSE, the fastest; INVWITHIMPRV, which computes the inverse to machine accuracy; and INVERSECVERA, which minimizes storage.

The question of what to do if a program determines that a given problem is beyond its capacity, for example, LINEQN encountering a singular matrix, is generally resolved in favor of printing an error message and exiting. However, the programs of SYMLINPAC, as well as DIFEQNS, return to a user-supplied label if an error condition is encountered. Discounting error messages, all programs except DIFEQNS, SIMPLEDIFEQNS, and DESP are input-output free.

CONTENTS OF THE LIBRARY.
The library is supplied as a collection of the following files:

a. FORTLIB/SYMBOL - FORTRANSYMBOL file that contains the source codes of all FORTRAN subroutines.

b. MATHLIB/SYMBOL - ALGOLSYMBOL file that contains the source codes of all ALGOL procedures.

c. FORTLIB/INDEX - DATA file that lists the sequence numbers in the file FORTLIB/SYMBOL at which the individual subroutines occur.

d. MATHLIB/INDEX - DATA file that lists the sequence numbers in the file MATHLIB/SYMBOL at which the individual procedures occur.

e. FORTLIB/CATALOG - DATA file that lists the subroutines according to abilities.

f. MATHLIB/CATALOG - DATA file that lists the procedures according to abilities.

g. FORTLIB/DOCUMENT - DATA file that describes the syntax, calling secuence, and input/output requirements of the subroutines in FORTLIB/SYMBOL.

h. MATHLIB/DOCUMENT - DATA file that describes the syntax, calling secuence, and input/output requirements of the procedures in MATHLIB/SYMBOL.

i. LIBRARY/SYNTAX - ALGOLCODE file of a program that prints out, when executed, the section of file FORTLIB/DOCUMENT or MATHLIB/DOCUMENT pertaining to a specified routine. The FORTLIB/DOCUMENT and MATHLIB/DOCUMENT files are expected on disk at the time the program is executed. If a label equation to another storage device is desired, the internal name of the file is FIN. The program is executed as follows:

```
? EXECUTE LIBRARY/SYNTAX("file name,routine name")
? END
```

where

file name = FORTLIB or MATHLIB

routine name = name of a routine as it occurs on the index files

USE OF THE LIBRARY.

To solve a mathematical problem, the user may adopt the following stepwise procedure:

a. Refer to the catalogs for the FORTRAN or ALGOL libraries (FORTLIB/CATALOG or MATHLIB/CATALOG) and select appropriate routine. If no routine is available, go to step j.

b. Use the prcgram LIBRARY/SYNTAX to obtain the syntax for the routine.

c. Decide whether to use the object code or the source code. Proceed to step g if object code binding is selected.

d. Refer to the corresponding irdex (FORTLIB/INDEX or MATHLIB/INDEX) fcr the lccation of the selected routine in the source file (FORTLIB/SYMBOL or MATHLIB/SYMBOL).

e. Use the $ INCLUDE option to insert the proper routine in the main program.

f. If the rcutine performs as expected and solves the problem, go to step k; otherwise go to step i.

g. Use the BINDER Program to bird the object code of the selected routine to the main program. The compile time option $ AUTOBIND may also be used.

h. If the rcutine performs as expected and solves the problem, go to step k; otherwise go to step i.

i. Serd all pertinent informaticn regarding the problem to the Project Manager, NUMERALS. Go to step k.

j. Send an ability request to the Project Manager, NUMERALS.

k. END.

SECTION 1

MATRICES

All matrices used in the following programs are assumed to be stored in locations 1:N, 1:N of their respective arrays, and vectors in locations 1:N; computed matrices and vectors are similarly stored. That is, row and column 0 are not used for matrix storage, and location 0 is not used for vector storage. The only exceptions to this are banded, symmetric matrices, which must be input in a special form. However, all arrays passed to matrix procedures must be declared with lower bounds of 0, a requirement dictated by hardware efficiency. For both linear systems and eigenvalues, only the lower triangular half of a symmetric matrix is needed.

Operations on rows (treated as array rows) rather than columns are used wherever possible for increased speed, except in procedure INVERSEOVERA where storage limitations preclude this. INVERSEOVERA is therefore slower than INVERSE, but requires only one-half as much storage. Additional features for linear systems are the use of double precision arithmetic in an iterative method of improving solutions to machine accuracy, and a method for computing the determinant which avoids exponent overflow or underflow. Many procedures for linear systems appear in two versions, the first for real matrices and the second for symmetric matrices. In addition, there is a separate procedure to solve complex linear equations.

Under the paragraph heading EIGENVALUES AND EIGENVECTORS, in this section, are the most recently published procedures for the computation of eigenvalues of real, symmetric, and complex matrices. Here, also, double precision arithmetic is used for key computations. Each procedure is thoroughly tested.

Common arithmetic operations on square matrices as well as addition, subtraction, and multiplication of rectangular matrices are provided for under the paragraph heading MATRIX ARITHMETIC, in this section.

LINEAR EQUATIONS, MATRIX INVERSION, AND DETERMINANTS.

REAL MATRICES - LINPAC.

LINPAC is a collection of procedures to solve a system of linear equations, invert matricies, and compute determinants. A full accuracy option allows solutions and inverses to be computed to full machine accuracy (11 + decimal digits). The different procedures given allow for different combinations of options.

The basic method employed is Gaussian elimination with partial pivoting (row interchanges), also known as LU decomposition. The versions for linear equations use combinations of two or more basic working procedures, of which there are four; DECOMPOSE, SOLVE, IMPROVE, and DETERMINANT. A description of these four follows.

DECOMPOSE transforms a copy of the input matrix A (some procedures allow A itself to be transformed, at the user's option) into the product of an upper triangular matrix U and a lower triangular matrix L

with 1's along the main diagonal. U is stored in the upper triangular half of the array LU and L is stored in the lower triangular half with the 1's omitted. The elimination is carried out by using a form of scaling in addition to choosing maximum column pivots. It is done as follows: Before the elimination begins, the reciprocal of the maximum magnitude element in each row is recorded. The elimination proceeds by choosing as the pivot element the largest entry, taking into account the scale factors, on or below the main diagonal. Rather than actually interchange the rows, the row indices stored in the array PS are interchanged. The LU decomposition actually satisfies the equation LU = PA, where P is a permutation matrix because of the row interchanges.

The solution is computed in SOLVE by solving successively the systems LY = B and UX = Y (the back solution).

IMPROVE iterates on the initial solution provided by SOLVE until full machine accuracy is achieved. This is done as follows: The residual vector R = AX-B is computed in double precision arithmetic, and the correction DX1 satisfying (approximately) ADX1 = R is computed by SOLVE. The process is repeated for the new residual vector R1 = A(X+DX1)-B and continues until the correction becomes negligible. Iterative improvement requires only a copy of A and a modest amount of additional time because the LU decomposition of A need only be computed for the initial solution.

DETERMINANT computes the significant digits and exponent of det(A) separately to avoid possible exponent overflow or underflow. Det(A) is stored as a real number, DETBASE, and an integer power of 10, DETEXP: det(A) = DETBASE X 10.0 @ DETEXP. If det(A) lies within floating point range, DETBASE is set to the value det(A) and DETEXP is set to 0. Otherwise, DETBASE satisfies 1 LEQ ABS(DETBASE) LSS 8.0@13 and DETEXP is the appropriate power of 10 for DETBASE.

If a singular matrix is detected, procedure SINGULAR is called to print an error message; the computation is then continued, resulting in a divide by zero in SOLVE. This may be avoided by declaring a LABEL, perhaps ERREXIT, in the user program, inserting the statement GO TO ERREXIT at the end of procedure SINGULAR, and labeling an appropriate statement in the calling program with the LABEL ERREXIT.

Procedure INVERSE obtains the LU decomposition from A by calling DECOMPOSE, and then solves the N systems of equations LUX = B[I], with B[I] the Ith column of the Identity matrix, I = 1,2,...,N. Rather than make N calls to SOLVE, INVERSE takes advantage of the special structure of the columns of the Identity matrix to achieve full efficiency. It is the fastest of the three inversion routines.

INVWITHIMPRV uses DECOMPOSE, SOLVE, and IMPROVE to compute each column of the inverse to full machine accuracy. No advantage is taken of the special structure of the Identity matrix. This procedure is four to five times slower than INVERSE and should only be used when a fully accurate inverse is necessary.

INVERSEOVERA also computes the LU decomposition of A, but inverts L and U in place, i.e., without requiring a separate array for storage of the inverse. It therefore requires only half as much storage as INVERSE, but is slower.

Reference.

Forsythe, George E. and Moler, Cleve B. 1967. Computer Solution of Linear Algebraic Systems. Englewood Cliffs, N. J.:Prentice-Hall.

LINEQN PROCEDURE. The syntax for the LINEQN procedure is:

```
PROCEDURE LINEQN(N,A,LU,B,X);
VALUE N; INTEGER N;
ARRAY A,LU[0,0], B,X[0];
```

Purpose.
LINEQN solves the linear system AX = B.

Input.
Input for LINEQN consists of:

N - dimension of coefficient matrix A
A - coefficient matrix
B - right-hand-side vector

Output.
Output for LINEQN consists of:

LU - matrix containing the triangular decomposition of A
X - solution vector

Remarks.
LINEQN(N,A,A,B,X) overwrites A with its triangular decomposition and saves NxN words of storage.

LINEQN uses procedures DECOMPOSE, SOLVE, and, indirectly, INNERPROD, ELIM, and SINGULAR.

DETLINEQN PROCEDURE. The syntax for the DETLINEQN procedure is:

```
PROCEDURE DETLINEQN(N,A,LU,B,X,DETBASE,DETEXP);
VALUE N; INTEGER N,DETEXP;
REAL DETBASE;
ARRAY A,LU[0,0], B,X[0];
```

Purpose.
DETLINEQN solves the linear system AX = B and calculates the determinant of the coefficient matrix A.

Input.
Input to DETLINEQN consists of:

N - dimension of coefficient matrix A
A - coefficient matrix
B - right-hand-side vector

Output.
Output for DETLINEQN consists of:

LU - matrix containing the triangular decomposition of A
X - solution vector
DETBASE - contains the significant digits of det(A)
DETEXP - contains the appropriate power of 10 for DETBASE

Remarks.
DETLINEQN(N,A,A,B,X,DETBASE,DETEXP) overwrites A with its triangular decomposition and saves NxN words of storage.

DETLINEQN uses procedures DECOMPOSE, SOLVE, DETERMINANT, and, indirectly, INNERPROD, ELIM, and SINGULAR.

LINEQNIMPRV PROCEDURE. The syntax is as follows:

```
PROCEDURE LINEQNIMPRV(N,A,B,X,DIGITS);
VALUE N; INTEGER N;
REAL DIGITS;
ARRAY A[0,0], B,X[0];
```

Purpose.
LINEQNIMPRV solves the linear system AX = B and improves the solution X to machine accuracy.

Input.
Input to LINEQNIMPRV consists of:

N - dimension of coefficient matrix A
A - coefficient matrix
B - right-hand-side vector

Output.
Output for LINEQNIMPRV consists of:

X - solution vector
DIGITS - number of correct digits in initial solution

Remarks.
LINEQNIMPRV uses procedures DECOMPOSE, SOLVE, IMPROVE, and, indirectly, INNERPROD, DOUBLEDOTPROD, ELIM, and SINGULAR.

DETLINEQNIMPRV PROCEDURE. The syntax is as follows:

```
PROCEDURE DETLINEQNIMPRV(N,A,B,X,DIGITS,DETBASE,DETEXP);
VALUE N; INTEGER N,DETEXP;
REAL DIGITS,DETBASE;
ARRAY A[0,0], B,X[0];
```

Purpose.
DETLINEQNIMPRV solves the linear system AX = B, improves the solution to machine accuracy, and computes the determinant of the coefficient matrix A.

Input.
Input to DETLINEQNIMPRV consists of:

N - dimension of coefficient matrix A
A - coefficient matrix
B - right-hand-side vector

Output.
Output for DETLINEQNIMPRV consists of:

X - solution vector
DIGITS - number of correct digits in initial solution
DETBASE - contains the significant digits of det(A)
DETEXP - contains the appropriate power of 10 for DETBASE

Remarks.
DETLINEQNIMPRV uses procedures DECOMPOSE, SOLVE, IMPROVE, DETERMINANT, and, indirectly, INNERPROD, DOUBLEDOTPROD, ELIM, and SINGULAR.

LINEQNS PROCEDURE. The syntax is as follows:

```
PROCEDURE LINEQNS(N,A,K,C,Y);
VALUE N,K; INTEGER N,K;
ARRAY A,C,Y[0,0];
```

Purpose.
LINEQNS solves the K systems of equations AY = C, where the K right-hand sides are stored in rows 1 through K of the KxN matrix C, and the corresponding K solutions are placed in rows 1 through K of the KxN matrix Y.

Input.
Input to LINEQNS consists of:

N - dimension of coefficient matrix A

A - coefficient matrix

K - row dimension of the matrices C and Y

C - the matrix used to store the set of right-hand-sides as rows, dimensioned C[0:K,0:N]

Output.
Output for LINEQNS consists of:

Y - the matrix used to store the set of solutions as rows, dimensioned Y[0:K,0:N]

Method.
After A is decomposed into triangular form, SOLVE is called successively for each row of C. The rows of C and Y are treated as 1-dimensional arrays using the array row feature of Burroughs ALGOL so that no efficiency is lost.

Remarks.
A is overwritten by its LU decomposition in the process.

LINEQNS uses procedures DECOMPOSE, SOLVE, and, indirectly, INNERPROD, ELIM, and SINGULAR.

DETLINEQNS PROCEDURE. The syntax is as follows:

```
PROCEDURE DETLINEQNS(N,A,K,C,Y,DETBASE,DETEXP);
VALUE N,K; INTEGER N,K,DETEXP;
REAL DETBASE;
ARRAY A,C,Y[0,0];
```

Purpose.
The purpose of DETLINEQNS is the same as LINEQNS except that in addition the determinant of A is calculated.

Input.
Input to DETLINEQNS consists of:

N - dimension of A

A - coefficient matrix

K - row dimension of matrices C and Y

C - matrix used to store the set of right-hand sides as rows, dimensioned C[0:K,0:N]

Output.
Output for DETLINEQNS consists of:

Y - matrix used to store the set of solutions as rows, dimensioned Y[0:K,0:N]

DETBASE - contains the significant digits of det(A)

DETEXP - contains the appropriate power of 10 for DETBASE

Remarks.
DETLINEQNS uses procedures DECOMPOSE, SOLVE, DETERMINANT, and, indirectly, INNERPROD, ELIM, and SINGULAR.

INVERSE PROCEDURE. The syntax is as follows:

```
PROCEDURE INVERSE(N,A,AINVERSE,DETBASE,DETEXP);
VALUE N; INTEGER N,DETEXP;
REAL DETBASE;
ARRAY A,AINVERSE[0,0];
```

Purpose.
INVERSE computes the inverse of the matrix A, stores it in AINVERSE, and computes the determinant of A.

Input.
Input to INVERSE consists of:

N - dimension of A
A - matrix to be inverted

Output.
Output for INVERSE consists of:

AINVERSE - array containing the inverse of A
DETBASE - contains the significant digits of det(A)
DETEXP - contains the appropriate power of 10 for DETBASE

Method.
The inverse is computed by solving successively the N systems of equations represented by A x AINVERSE = I, taking account of the special structure of the columns of the identity matrix I. No iterative improvement is used.

Remarks.
A and AINVERSE must occupy separate storage. A is overwritten by its LU decomposition in the process.

INVERSE uses procedures DECOMPOSE, SOLVE, DETERMINANT, and, indirectly, INNERPROD, ELIM, and SINGULAR.

INVWITHIMPRV PROCEDURE. The syntax is as follows:

```
PROCEDURE INVWITHIMPRV(N,A,AINVERSE,DETBASE,DETEXP);
VALUE N; INTEGER N,DETEXP;
REAL DETBASE;
ARRAY A,AINVERSE[0,0];
```

Purpose.
INVWITHIMPRV computes the inverse of the matrix A, stores it in AINVERSE, and computes the determinant of A.

Input.
Input to INVWITHIMPRV consists of:

N - dimension of A
A - matrix to be inverted

Output.
Output for INVWITHIMPRV consists of:

AINVERSE - array containing the matrix of A
DETBASE - contains the significant digits of det(A)
DETEXP - contains the appropriate power of 10 for DETBASE

Method.
The inverse is calculated by solving successively the N systems of equations represented by A x AINVERSE = I, where I is the NxN identity matrix. Each column of AINVERSE is calculated to machine precision using iterative refinement. No account of the special structure of I is taken. Since A must be preserved for the iterative improvement, INVWITHIMPRV requires NxN more words of storage than INVERSE and takes four to five times longer.

Remarks.
A and AINVERSE must occupy separate storage.

INVWITHIMPRV uses procedures DECOMPOSE, SOLVE, IMPROVE, DETERMINANT, and, indirectly, INNERPROD, DOUBLEPROD, ELIM, and SINGULAR.

DETOFA PROCEDURE. The syntax is as follows:

```
PROCEDURE DETOFA(N,A,DETBASE,DETEXP);
VALUE N; INTEGER N,DETEXP;
REAL DETBASE;
ARRAY A[0,0];
```

Purpose.
DETOFA computes the determinant of A.

Input.
Input to DETOFA consists of:

N - dimension of A
A - matrix whose determinant is to be computed

Output.
Output for DETOFA consists of:

DETBASE - contains the significant digits of det(A)
DETEXP - contains the appropriate power of 10 for DETBASE

Method.
DETOFA computes the significant digits and exponent of det(A) separately, to avoid possible exponent overflow or underflow. Det(A) is stored as a real number, DETBASE, and as an integer power of 10, DETEXP: det(A) = DETBASE x 10*DETEXP. If det(A) lies within floating point range, DETBASE is set to the value det(A) and DETEXP is set to 0. Otherwise, DETBASE satisfies LEQ ABS(DETBASE) LSS 8*13 and DETEXP is the appropriate power of 10 for DETBASE.

Remarks.
A is overwritten with its LU decomposition.

DETOFA uses procedures DECOMPOSE, DETERMINANT, and, indirectly, INNERPROD, ELIM, and SINGULAR.

INVERSEOVERA PROCEDURE. The syntax is as follows:

```
PROCEDURE INVERSEOVERA(N,A);
VALUE N; INTEGER N;
ARRAY A[0,0];
```

Purpose.
INVERSEOVERA replaces the matrix A by its inverse.

Input.
Input to INVERSEOVERA consists of:

N - dimension of A
A - A[1:N,1:N] contains the matrix to be inverted

Output.
Output for INVERSEOVERA consists of:

A - the original matrix is overwritten by its inverse

METHOD.
INVERSEOVERA inverts the matrix A "in place." That is, as the elements of the inverse of A are computed, they replace the corresponding elements of A, thus saving NxN words of storage over procedure INVERSE. INVERSEOVERA is, however, slower. A is first decomposed into triangular form by procedure TRIANGLDECOMP which is the same as procedure DECOMPOSE in LINPAC except that actual row interchanges are used. The triangular form is then inverted by procedure INVERT, as follows: The inverse, X, of the product LU (the triangular form) is calculated from the conditions that XL be an upper triangular matrix and UX be a unit lower triangular matrix. The columns of X are then interchanged in the reverse order as the row interchanges in TRIANGDECOMP to get the inverse of A.

Remarks.
A must be declared with lower bounds of zero in the calling program.

Reference.
Fox, L. 1951. Practical Solution of Linear Equations and Inversion of Matrices. In Nat. Bur. Stand. AMS 39 ed. O. Taussky, pp. 1-54.

LEASTSQUARES PROCEDURE. The syntax is as follows:

```
PROCEDURE LEASTSQUARES(M1,M,N,A,B,X,RES,FIRST,SINGULAR);
VALUE M1,M,N,FIRST; INTEGER M1,M,N;
ARRAY A[0,0], B,X,RES[0];
BOOLEAN FIRST;
LABEL SINGULAR;
```

Purpose.
LEASTSQUARES solves an overdetermined system of M equations in N unknowns. A specified number of the equations may be satisfied exactly.

Input.
Input to LEASTSQUARES consists of:

M1 - the first M1 equations are satisfied exactly. M1 must satisfy: 0 LEQ M1 LEQ N.

M - number of equations (rows of the matrix)

N - number of unknowns (columns of the matrix)

A - coefficient matrix is input in locations 1:M,1:N of the array A[0:M,0:N]

B - right-hand-side vector is input in locations 1:M of the array B[0:M]

FIRST - Boolean switch which must be true on the first call to LEASTSQUARES and false on subsequent calls with the same coefficient matrix and different right-hand sides

SINGULAR - label to which control is transferred if the QR decomposition cannot be completed (if A has rank less than N or M1 is chosen too large)

Output.
Output for LEASTSQUARES consists of:

A - A is overwritten with its QR decomposition

X - the solution is output in locations 1:N of X[0:N]

RES - the residuals are output in locations M1+1-M of the array RES[0:M]

Method.
If FIRST is false, A is decomposed into the product Q X R of an orthogonal matrix Q and an upper triangular matrix R using a sequence of elementary Householder transformations. Pivoting is accomplished by choosing among the remaining columns. The one with the largest sum

of squares to be reduced is next. If FIRST is false, the array QR is already assumed to contain the QR decomposition of A. Control proceeds directly to ACCSOLVE. In this case the value of M1 must be the same as that used on the first call to LEASTSQUARES with the current matrix. The solution vector X and the residual vector R are computed in ACCSOLVE.

Reference.
Bjorck, Ake and Golub, Gene. 1967. Iterative Refinement of Linear Least Square Solutions by Householder Transformation. BIT 7:322-337.

LEASTSQUARESIMPRV PROCEDURE. The syntax is as follows:

```
PROCEDURE LEASTSQUARESIMPRV(M1,M,N,A,QR,B,X,RES,FIRST,SINGULAR,
                            FAIL);
VALUE M1,M,N,FIRST; INTEGER M1,M,N;
ARRAY A,QR[0,0],B,X,RES[0];
BOOLEAN FIRST;
LABEL FAIL,SINGULAR;
```

Purpose.
Solves an overdetermined system of M equations in N unknowns. A specified number of the equations may be satisfied exactly. The solution is improved to machine accuracy.

Input.
Input to LEASTSQUARESIMPRV consists of:

M1 - the first M1 equations are satisfied exactly. M1 must satisfy: 0 LEQ M1 LEQ N.

M - number of equations (rows of the matrix)

N - number of unknowns (columns of the matrix)

A - coefficient matrix is input in locations 1:M,1:N of the array A[0:M,0:N]

B - right-hand-side vector is input in locations 1:M of the array B[0:M]

FIRST - Boolean switch which must be true on the first call to LEASTSQUARES and false on subsequent calls with the same coefficient matrix and different right-hand sides

SINGULAR - label to which control is transferred if the QR decomposition cannot be completed (if A has rank less than N or M1 is chosen too large)

FAIL - label to which control is transferred if the iterative refinement fails to improve the solution sufficiently

Output.
Output for LEASTSQUARESIMPRV consists of:

A - A is preserved by the procedure

QR - the QR decomposition of A is output in the array QR[0:M,0:N]

X - the solution is output in locations 1:N of X[0:N]

RES - the residuals are output in locations M1+1-M of

the array RES[0:M]

Method.
If FIRST is true, A is decomposed into the product QXR of an orthogonal matrix Q and an upper triangular matrix R, using a sequence of elementary Householder transformations. Pivoting is accomplished by choosing, among the remaining columns the one with the largest sum of squares to be reduced next. If FIRST is false, the array QR is assumed to already contain the QR decomposition of A and control proceeds directly to ACCSOLVE. In this case the value of M1 must be the same as that used on the first call to LEASTSQUARESIMPRV with the current matrix.

In ACCSOLVE the first solution is taken to be a null vector, and the first two iteration steps are always executed. The iteration for the current right-hand side is terminated when the conditions (1) and (2) below are simultaneously satisfied:

(1) N(DX(S)) GEQ .125xN(DX(S-1)) OR N(DX(S)) LEQ ETAxN(X(1))
(2) N(DR(S)) GEQ .125xN(DR(S-1)) OR N(DR(S)) LEQ ETAxN(R(1))

N stands for the Euclidean norm, and DX(S) means the value of DX after the Sth iteration. If the iteration is terminated and at the same time

N(DX(S)) GTR 2xETAxN(X(1)) AND N(DR(S)) GTR 2xETAxN(R(1))

the exit FAIL is used.

Reference.
This is a modified version of procedures appearing in: Bjorck, Ake and Golub, Gene. 1967. Iterative Refinement of Linear Least Square Solutions by Householder Transformation. BIT 7:322-337.

SYMMETRIC MATRICES - SYMLINPAC.
SYMLINPAC is a set of procedures, analogous to LINPAC, used to solve systems of equations with symmetric coefficient matrices and to invert such matrices. Full advantage is taken of the symmetry by requiring only the lower triangle of the input matrix for each SYMLINPAC procedure. Full accuracy options, which allow solutions and inverses to be computed to 11+ decimal digits, are included. If determinants are desired, the relevant procedures of LINPAC must be utilized instead.

A variant of the Cholesky decomposition allowing nonsingular symmetric matrices which are not necessarily positive-definite is used. However, the method guarantees numerical stability only for equations and inverses involving positive-definite matrices, and for general symmetric matrices serious loss of accuracy may occur because pivoting (row interchanges) is not performed. The matrix is decomposed into the product A = LDU, where L is a lower triangular matrix with ones on the diagonal, U is its transpose, and D is diagonal. If the matrix is not required for iterative improvement (it is not in SYMLINEQN, SYMLINEQNS, and SYMINVERSEOVERA), the lower triangle of A is overwritten with L. In SYMLINEQNIMPRV and SYMINVWITHIMPRV, A is preserved and L is stored separately. The reciprocals of the elements of D are stored instead of the elements themselves. Iterative improvement, when desired, is accomplished with essentially the same procedure as IMPROVE in LINPAC.

Should a singular matrix be encountered, control is returned to the statement in the calling program whose LABEL is passed as the formal parameter SINGULARMATRIX (FAIL in SYMBANDLINEQN).

SYMBANDLINEQN uses the standard Cholesky decomposition, which requires positive-definiteness, to solve a band symmetric system of linear equations. Full advantage of the band structure is taken by storing only the non-zero elements of the input matrix. A Boolean switch, FIRST, is provided for efficient calculation of systems with several right-hand sides. On the first call to SYMBANDLINEQN, FIRST must be set TRUE, in which case the input matrix is overwritten with its Cholesky decomposition. On subsequent calls with the same coefficient matrix and different right-hand sides, FIRST should be set FALSE, in which case the decomposition is skipped. The inverse matrix can be calculated with reasonable efficiency by making N calls, with B the successive columns of the identity matrix.

References.
Martin, R. S.; Peters, G.; and Wilkinson, J. H. 1965. Symmetric Decomposition of a Positive Definite Matrix. Numerische Mathematik 7:362-383.

Wilkinson, J. H. and Reinsch, C. eds. 1971. Linear Algebra. Handbook for Automatic Computation. Vol. 2 New York:Springer-Verlag.

SYMLINEQN PROCEDURE. The syntax is as follows:

```
PROCEDURE SYMLINEQN(N,A,B,X,SINGULARMATRIX);
VALUE N; INTEGER N;
ARRAY A[0,0], B,X[0];
LABEL SINGULARMATRIX;
```

Purpose.
SYMLINEQN solves the symmetric linear system AX = B.

Input.
Input to SYMLINEQN consists of:

N - order of system

A - array containing the coefficient matrix which must be symmetric. It is only necessary to store the lower triangle of the matrix; the strict upper triangle of A is not used.

B - right-hand-side vector

SINGULARMATRIX - label in the calling program to which control is transferred if a singular matrix is encountered

Output.
Output for SYMLINEQN consists of:

A - the strict lower triangle of A is overwritten with its triangular decomposition

X - solution vector

Method.
A is decomposed into triangular form without using pivoting; if A is not positive-definite, serious loss of accuracy may occur.

Remarks.
The call SYMLINEQN(N,A,B,B,SINGULARMATRIX) overwrites B with the solution X.

SYMLINEQN uses procedures SYMDECOMPOSE, SYMSOLVE, and, indirectly, SYMELIM and INNERPROD.

SYMLINEQNIMPRV PROCEDURE. The syntax is as follows:

```
PROCEDURE SYMLINEQNIMPRV(N,A,B,X,DIGITS,SINGULARMATRIX);
VALUE N; INTEGER N;
REAL DIGITS;
ARRAY A[0,0], B,X[0];
LABEL SINGULARMATRIX;
```

Purpose.
SYMLINEQNIMPRV solves the symmetric linear system AX = B and improves the solution X to machine accuracy.

Input.
Input to SYMLINEQNIMPRV consists of:

N - order of system

A - array containing the coefficient matrix which must be symmetric. It is only necessary to store the lower triangle of the matrix. However, if the full matrix is stored, the call to SYMMETRIZE is superfluous and may be removed.

B - right-hand-side vector

SINGULARMATRIX - label in the calling program to which control is transferred if a singular matrix is encountered

Output.
Output for SYMLINEQNIMPRV consists of:

X - solution vector
DIGITS - number of correct digits in initial solution

Method.
A copy of A is decomposed into triangular form without using pivoting. The solution is improved to machine accuracy unless the initial solution is too inaccurate for the iterative improvement to converge, in which case an error message is written and control is transferred to the statement labeled SINGULARMATRIX in the calling program.

Remarks.
B and X must occupy separate storage.

SYMLINEQNIMPRV uses procedures SYMDECOMPOSE, SYMSOLVE, SYMMETRIZE, SYMIMPROVE, and, indirectly, SYMELIM, DOUBLEDOTPROD, and INNERPROD.

SYMLINEQNS PROCEDURE. The syntax is as follows:

```
PROCEDURE SYMLINEQNS(N,A,K,C,Y,SINGULARMATRIX);
VALUE N,K; INTEGER N,K;
ARRAY A,C,Y[0,0];
LABEL SINGULARMATRIX;
```

Purpose.
SYMLINEQNS solves the K symmetric systems of equations AY = C, where the K right-hand sides are stored in rows 1-K of the array C, and the corresponding K solutions are placed in rows 1-K of Y.

Input.
Input to SYMLINEQNS consists of:

N - dimension of coefficient matrix

A - array containing the coefficient matrix which must be symmetric. It is only necessary to store the lower triangle of the matrix; the strict upper triangle of A is not used.

K - the number of sets of equations to be solved

C - right-hand sides of the equations are stored in rows 1-K of array C[0:K,0:N]

SINGULARMATRIX - label in the calling program to which control is transferred if a singular matrix is encountered

Output.
Output for SYMLINEQNS consists of:

A - the strict lower triangle of A is overwritten with its triangular decomposition

Y - solutions of the K systems are stored in rows 1-K of Y[0:K,0:N]

Method.
A is decomposed into triangular form without using pivoting so that if A is not positive-definite, serious loss of accuracy may occur. SYMSOLVE is called successively for each row of C. The rows of C and Y are treated as 1-D arrays (array rows) so that efficiency is not lost.

Remarks.
The call SYMLINEQNS(N,A,K,C,C,SINGULARMATRIX) overwrites the right-hand-side matrix C with the solution matrix Y.

SYMLINEQNS uses procedures SYMDECOMPOSE, SYMSOLVE, and, indirectly, SYMELIM and INNERPROD.

SYMINVWITHIMPRV PROCEDURE. The syntax is as follows:

```
PROCEDURE SYMINVWITHIMPRV(N,A,AINVERSE,SINGULARMATRIX);
VALUE N; INTEGER N;
ARRAY A,AINVERSE[0,0];
LABEL SINGULARMATRIX;
```

Purpose.
SYMINVWITHIMPRV computes the complete inverse of the symmetric matrix A, improves it to machine accuracy, and stores it in AINVERSE[1:N,1:N].

Input.
Input to SYMINVWITHIMPRV consists of:

N - order of A

A - array containing the matrix to be inverted; only the lower triangle of A is required. However, if the full matrix is stored, the call to SYMMETRIZE is superfluous and may be removed.

SINGULARMATRIX - label in the calling program to which control is transferred if a singular matrix is encountered

Output.
Output for SYMINVWITHIMPRV consists of:

AINVERSE - array containing the full inverse of A

Method.
A copy of A is decomposed into triangular form without using pivoting. The inverse is calculated and improved column by column, taking account of the special form of the columns of the identity matrix. If the iterative improvement of a column fails to converge, an error message is written and control is transferred to the statement labeled SINGULARMATRIX in the calling program.

Remarks.
This procedure is much slower than SYMINVERSEOVERA and should be used only if a fully accurate inverse is desired.

A and AINVERSE must occupy separate storage.

SYMINVWITHIMPRV uses procedures SYMDECOMPOSE, SYMSOLVE, SYMMETRIZE, SYMIMPROVE, and, indirectly, SYMELIM, DOUBLEDOTPROD, and INNERPROD.

SYMBANDLINEQN PROCEDURE. The syntax is as follows:

```
SYMBANDLINEQN(N,M,A,B,X,FIRST,FAIL);
VALUE N,M,FIRST; INTEGER M,N; BOOLEAN FIRST;
ARRAY A[0,0], B,X[0];
LABEL FAIL;
```

Purpose.
SYMBANDLINEQN solves the linear system AX = B, where A is a symmetric positive-definite band matrix.

Input.
Input to SYMBANDLINEQN consists of:

N - order of system

M - number of lines on either side of the main diagonal

A - A[1:N,0:M] contains the lower half of the positive-definite band coefficient matrix. The subdiagonals of the matrix are stored in the columns of A, with the main diagonal in A[I,M], I = 1 (1) N. The elements of the stored array occupy the positions illustrated below for N=5, M=2 of a conventional array.

Stored Array			Lower Triangle of Conventional Array				
X	X	A[1,2]	A[1,2]				
X	A[2,1]	A[2,2]	A[2,1]	A[2,2]			
A[3,0]	A[3,1]	A[3,2]	A[3,0]	A[3,1]	A[3,2]		
A[4,0]	A[4,1]	A[4,2]		A[4,0]	A[4,1]	A[4,2]	
A[5,0]	A[5,1]	A[5,2]			A[5,0]	A[5,1]	A[5,2]

The elements indicated by X's in the upper left-hand corner of the stored array may be arbitrary since they are not used. A is preserved by the program.

B - B[1:N] contains the right-hand side of the system

FIRST - a TRUE/FALSE switch. The call SYMBANDLINEQN(N,M,A,B,X, TRUE,FAIL) is used for the first set of equations, whereas additional sets with the same coefficient matrix and different right-hand sides are solved by the call SYMBANDLINEQN(N,M,A,B,X,FALSE,FAIL).

FAIL - a label in the calling program to which control is transferred if the matrix A is not positive-definite

Output.
Output for SYMBANDLINEQN consists of:

A - A is overwritten with the lower triangular part of the Cholesky decomposition A = LU

X - X[1:N] contains the solution

Method.
The Cholesky decomposition A = LU, where U is the transpose of L, is performed and L is stored in the same form as A. The reciprocals of the diagonal elements of L are stored instead of the elements themselves. If A is not positive-definite, the procedure fails and control is transferred to the statement labeled FAIL in the calling program.

Remarks.
The call SYMBANDLINEQN(N,M,A,B,B,FIRST,FAIL) overwrites B with the solution X.

Reference.
Martin, R. S. and Wilkinson, J. H. 1965. Symmetric Decomposition of Positive Band Matrices. Numerische Mathematik 7:355-361.

SYMINVERSEOVERA PROCEDURE. The syntax is as follows:

```
PROCEDURE SYMINVERSEOVERA(N,A,SINGULARMAT);
VALUE N; INTEGER N;
ARRAY A[0,0];
LABEL SINGULARMAT;
```

Purpose.
SYMINVERSEOVERA overwrites the symmetric matrix stored in the lower triangle of A[1:N,1:N] with its complete inverse.

Input.
Input to SYMINVERSEOVERA consists of:

N - order of A

A - array containing the matrix to be inverted. It is only necessary to store the lower triangle of the matrix. The strict upper triangle is not used on input.

SINGULARMAT - label in the calling program to which control is transferred in case of a singular matrix

Output.
Output for SYMINVERSEOVERA consists of:

A - A is overwritten with the entire inverse matrix which is also symmetric

Method.
The decomposition A = LDU, where U is the transpose of L, is performed. L with the unit diagonal omitted is overwritten on the strict lower triangle of A, and the reciprocals of the elements of D are stored on the main diagonal. The transpose of the inverse of L (omitting its unit diagonal) is then written on the strict upper triangle of A, after which L inverse and D are replaced by the upper triangle of the inverse matrix. Lastly, the remainder of the inverse is overwritten on the strict lower triangle of A.

Pivoting is not used; if A is not positive-definite, serious loss of accuracy may occur.

Reference.
This is a modification of procedure SYMINVERSION which appears in: Martin, R. S.; Peters, G; and Wilkinson, J. H. 1965. Symmetric Decomposition of a Positive Definite Matrix. Numerische Mathematik 7:362-383.

COMPLEX MATRICES.
A single procedure, an extension to complex arithmetic of the Gaussian elimination method employed in LINPAC, is given for complex linear equations. The Boolean switches FIRST and IMPROVE allow for efficient solution of systems with multiple right-hand sides and iterative improvement, respectively. As outlined for SYMBANDLINEQN, the inverse of a complex matrix can be computed by successive calls to CXLINEQN. For example, the following statements produce the inverse of A stored in RAINVERSE and IAINVERSE.

```
RB[1] := 0;
CXLINEQN(N,RA,IA,RLU,ILU,RB,IB,RX,IX,TRUE,FALSE);
FOR J := 2 STEP 1 UNTIL N DO
BEGIN
    RB[J] := 1; RB[J-1] := 0;
    CXLINEQN(N,RA,IA,RLU,ILU,RB,IB,RX,IX,FALSE,FALSE);
    FOR I := 1 STEP 1 UNTIL N DO
    BEGIN
        RAINVERSE[I,J] := RX[I];
        IAINVERSE[I,J] := IX[I];
    END;
END;
```

It is assumed here that the arrays RB and IB are initially set to zero. As another example, the following code placed after CXDECOMPOSE with the appropriate declarations computes the determinant of the complex matrix A.

```
RDET := RLU[1,1]; IDET := ILU[1,1];
FOR I := 2 STEP 1 UNTIL N DO
BEGIN
    RDET := RDETxRLU[I,I] - IDETxILU[I,I];
    IDET := RDETxILU[I,I] + IDETxRLU[I,I];
END;
```

CXLINEQN PROCEDURE. The syntax is as follows:

```
PROCEDURE CXLINEQN(N,RA,IA,RLU,ILU,RB,IB,RX,IX,FIRST,IMPROVE);
VALUE N,FIRST,IMPROVE; INTEGER N; BOOLEAN FIRST,IMPROVE;
ARRAY RA,IA,RLU,ILU[0,0], RB,IB,RX,IX[0];
```

Purpose.
CXLINEQN solves a system of complex linear equations (RA + IxIA) (RX + IxIX) = RB + IxIB. The solution may optionally be improved to machine accuracy.

Input.
Input to CXLINEQN consists of:

N - complex dimension of coefficient matrix A

RA - real part of coefficient matrix A

IA - imaginary part of coefficient matrix A

RB - real part of right-hand-side vector

IB - imaginary part of right-hand-side vector

FIRST - Boolean value. FIRST should be TRUE the first time CXLINEQN is called with a given coefficient matrix, and FALSE for further calls.

IMPROVE - Boolean value to specify whether or not iterative improvement is desired

Output.
Output for CXLINEQN consists of:

RLU - real part of triangular decomposition of A
ILU - imaginary part of triangular decomposition of A
RX - real part of solution vector
IX - imaginary part of solution vector

Method.
If FIRST is TRUE, a copy of A is reduced to triangular form by CXDECOMPOSE which uses Gaussian elimination with partial pivoting. The pivoting is accomplished by interchanging the row indices stored in the array PS, instead of physical row interchanges. If FIRST is FALSE, the triangular form of A is assumed to be already stored in the arrays RLU and ILU. The solution is computed in CXSOLVE and, if IMPROVE is TRUE, refined to machine accuracy in procedure CXIMPROVE. This is an extension to complex arithmetic of the method employed by LINPAC.

If a singular matrix is detected, an error message is printed and the procedure continued, resulting in a divide-by-zero. If the IMPROVE option is requested and the solution fails to converge after 23 iterations, an error message is printed and the last iteration is taken as a solution.

Remarks.
CXLINEQN(N,RA,IA,RA, IA, RB,IB,RX,IX,FIRST,CALSE) overwrites RA and IA with the triangular decomposition of A and saves 2xNxN words of storage.

Reference.
Forsythe, George E. and Moler, Cleve B. 1967. Computer Solution of Linear Algebraic Systems, Englewood Cliffs, N. J.:Prentice-Hall.

EIGENVALUES AND EIGENVECTORS.

The following paragraphs include procedures for calculating the full set of eigenvalues and eigenvectors of symmetric, band symmetric, and general real and complex matrices. Because of the computational effort required to find eigenvectors, the user is offered the choice between a procedure finding only the eigenvalues and one computing both eigenvalues and eigenvectors.

All procedures employ the QR or closely related LR algorithm and all except INVARIANTSUBSPACES are based on the work of Wilkinson and his colleagues published in Numerische Mathematik. Two types of preliminary transformations are applied to significantly improve the results of or decrease the computational effort involved in the QR or LR iterations. The first type is known as "balancing," a method of scaling without introducing rounding errors which equalizes the norms of corresponding rows and columns of the matrix and also reduces the norm of the matrix. Having as its major effect a substantial improvement in the accuracy of computed eigenvalues and eigenvectors of poorly conditioned matrices, balancing also detects isolated eigenvalues, effectively reducing the order of the matrix when they are present. Because of the volume of work required per QR or LR iteration on a full matrix, it is necessary to employ a second type of transformation to reduce the matrix to a simpler form. The transformation used depends on the kind of matrix, but each is numerically stable so that a minimum of rounding error is introduced. Although the eigenvalues of the original matrix are preserved by both types of transformations, the eigenvectors are not. Consequently, when eigenvectors are desired, the reduction procedures store sufficient information to allow back-transformation of the eigenvectors of the reduced matrix.

Symmetric matrices are initially balanced so that the only preliminary transformation required is a reduction to tri-diagonal form. For full symmetric matrices this is accomplished by Householder (orthogonal) transformations, whereas for band symmetric matrices a sequence of Jacobi rotations is used. The QL algorithm (essentially the same as the QR) is applied to the tri-diagonal matrix in each case. If eigenvectors of a full symmetric matrix are wanted, they are computed and back-transformed inside the QL procedure.

Balancing is carried out for real and complex matrices without user option because the amount of computation required is small. Both real and complex matrices are then reduced to upper Hessenberg form (zeroes below the first subdiagonal), the former by Householder transformations and the latter by elementary row and column operations with pivoting for stability. The Francis Double-Step QR algorithm, which avoids complex arithmetic, is applied to the real Hessenberg matrices, and the somewhat simpler LR algorithm is applied to the complex ones. The eigenvectors of the quasi-triangular (triangular) matrix produced by the QR (LR) algorithm are computed by solving the appropriate set of linear equations. Back-transformation of the eigenvectors proceeds slightly differently in the two cases. For real matrices, the transformation matrix is generated separately by procedure ORTRANS, then updated and applied to the eigenvectors inside HQR2. To account for balancing, a further back-transformation is performed by BALBAK. With

complex matrices the transformation matrix is generated, updated, and applied, all inside COMLR2. Back-transformation of the eigenvectors is accomplished by two calls to BALBAK.

INVARIANTSUBSPACES should only be applied to those matrices, real or complex, which are known or observed to have multiple eigenvalues whose corresponding eigenvectors are nearly linearly dependent. For such matrices, it produces accurate basis vectors for the subspace spanned by all of the eigenvectors corresponding to the multiple (or clustered) eigenvalue(s).

Only a few general comments are made about the QR and related LR algorithms because an adequate literature exists. The fastest of all known methods when all of the eigenvalues or eigenvectors are desired, it is, for example, several times faster than the Jacobi method, except for matrices of order less than 10. Failures are simply not encountered on matrices of order less than 50 (perhaps much higher). With regard to accuracy, the QR algorithm always produces an eigensystem which is exact for a matrix which differs only slightly from the input matrix.

Reference.
Reinsch, C. and Wilkinson, J. H., eds. 1971. Linear Algebra. Handbook for Automatic Computation. Vol. 2. New York:Springer-Verlag.

REAL MATRICES.

NONSYMEIGENVALUES PROCEDURE. The syntax is as follows:

```
PROCEDURE NONSYMEIGENVALUES(N,A,RTR,RTI);
VALUE N; INTEGER N;
ARRAY A[0,0], RTR,RTI[0];
```

Purpose.
NONSYMEIGENVALUES finds all N eigenvalues of the general real matrix A. Complex conjugate eigenvalues are stored consecutively.

Input.
Input to NONSYMEIGENVALUES consists of:

N - dimension of A

A - the matrix A is input in locations 1:N,1:N of the array A[0:N,0:N]

Output.
Output for NONSYMEIGENVALUES consists of:

A - the given matrix is destroyed in the process

RTR - the real parts of the eigenvalues are placed in locations 1:N of the array RTR[0:N]

RTI - the imaginary parts of the eigenvalues are placed in locations 1:N of the array RTI[0:N]

Method.
A is first prepared by procedure BALANCE, which detects any isolated eigenvalues and reduces the norm of A without introducing any round-off errors. ORTHES is then called to reduce A to upper Hessenberg form using elementary orthogonal (Householder) transformations. Procedure HQR, which utilizes the Francis Double-Step QR algorithm, is employed to find the eigenvalues of A.

Remarks.
Balancing is strongly recommended since it generally improves accuracy at a small additional cost in time. However, it is not necessary to balance Orthogonal or Normal matrices. If BALANCE is not used, LOW must be set to 1 and UPP to N.

If the original matrix is already of upper Hessenberg form (zeroes below the first subdiagonal), the call to ORTHES may be removed (procedure BALANCE preserves the Hessenberg form).

The procedure fails if an eigenvalue requires more than 30 iterations, in which case an error message is printed. Failures are extremely rare.

Reference.
Martin, R. S.; Peters, G.; and Wilkinson, J. H. 1970. The QR Algorithm for Real Hessenberg Matrices. Numerische Mathematik 14:219-231.

NONSYMEIGENVECTORS PROCEDURE. The syntax is as follows:

```
PROCEDURE NONSYMEIGENVECTORS(N,A,RTR,RTI,U);
VALUE N; INTEGER N;
ARRAY A,U[0,0], RTR,RTI[0];
```

Purpose.
NONSYMEIGENVECTORS finds all N eigenvalues and normalized column eigenvectors of the general real matrix stored in locations 1:N,1:N of the array A[0:N,0:N]. Complex conjugate eigenvalues are stored consecutively. The eigenvectors are stored in the columns of U; if the Kth eigenvalue is complex with positive imaginary part, column K contains the real part and column K+1 contains the imaginary part of the Kth eigenvector. The complex conjugate eigenvector is not stored.

Input.
Input to NONSYMEIGENVECTORS consists of:

N - order of A
A - general real matrix stored in A[1:N,1:N]

Output.
Output for NONSYMEIGENVECTORS consists of:

A - the upper triangular part of A contains the eigenvectors of the quasi-triangular matrix produced by the QR algorithm

RTR - the real parts of the eigenvalues are placed in locations 1:N of the array RTI[0:N]

RTI - the imaginary parts of the eigenvalues are placed in locations 1:N of the array RTI[0:N]

U - normalized eigenvectors are placed in locations 1:N of the columns of the array U[0:N,0:N]

Method.
A is first prepared by procedure BALANCE, which detects any isolated eigenvalues and reduces the norm of A without introducing any round-off errors. ORTHES is then called to reduce A to upper Hessenberg form, using elementary orthogonal (Householder) transformations. The matrix defining the similarity reduction of A to upper Hessenberg form is then formed and stored in the array U by procedure ORTRANS using the information left below the subdiagonal of A by ORTHES, completing the preliminary reductions and transformations. Procedure HQR2 is employed to obtain the eigenvalues of A and the eigenvectors of the balanced matrix. HQR2 utilizes the Francis Double-Step QR algorithm to compute the eigenvalues and a method of back substitution for the eigenvectors. The eigenvectors of the balanced matrix are next transformed into eigenvectors of the original matrix by procedure BALBAK. Finally, the eigenvectors of A are normalized so that a real eigenvector has its component of maximum magnitude equal to 1.0, and a complex eigenvector has its maximum modulus component equal to 1.0 + 0*I.

Remarks.
Balancing is strongly recommended since it generally improves accuracy at a small additional cost in time. However, it is not necessary to balance orthogonal or normal matrices. If BALANCE is not used, LOW must be set to 1 and UPP must be set to N.

If the original matrix is already of upper Hessenberg form (zeros below the first subdiagonal), the calls to ORTHES and ORTRANS may be removed (procedure BALANCE preserves the Hessenberg form). In this case, U must be set to the identity matrix before HQR2 is entered (it is only necessary to set the diagonal elements to 1).

If unnormalized eigenvectors are sufficient, the call to procedure NORMALIZE may be removed.

The procedure fails if any eigenvalue requires more than 30 iterations, in which case an error message is printed. Failures are extremely rare.

Reference.
Peters, G. and Wilkinson, J. H. 1970. Eigenvectors of Real and Complex Matrices by LR and QR Triangularization. Numerische Mathematik 16:181-204.

SYMMETRIC MATRICES.

SYMEIGENVALUES PROCEDURE. The syntax is as follows:

```
PROCEDURE SYMEIGENVALUES(N,A,VALUES);
VALUE N; INTEGER N;
ARRAY A[0,0], VALUES[0];
```

Purpose.
SYMEIGENVALUES finds all N eigenvalues of a real symmetric matrix. Only the lower triangle of the matrix is required.

Input.
Input to SYMEIGENVALUES consists of:

N - order of matrix

A - the lower triangle of the matrix is stored in the lower triangle of locations 1:N,1:N of the array A[0:N,0:N]. The strict upper triangle of A is unused.

Output.
Output for SYMEIGENVALUES consists of:

A - the lower triangle of A is destroyed

VALUES - the eigenvalues are placed in locations 1:N of the array VALUES[0:N] in ascending order

Method.
The matrix is first reduced to tridiagonal form using the Householder reduction. The eigenvalues of the tridiagonal matrix are found with QL transformations.

Remarks.
If the iteration fails to converge, an error message is printed and the procedure is exited. Failures are exceedingly rare.

Reference.
This is a modified version of procedures appearing in: Bowdler, H.; Martin, R. S.; Reinsch, C.; and Wilkinson, J. H. 1968. The QR and QL Algorithms for Symmetric Matrices. Numerische Mathematik 11:293-306.

SYMEIGENVECTORS PROCEDURE. The syntax is as follows:

```
PROCEDURE SYMEIGENVECTORS(N,A,VALUES,VECTORS);
VALUE N; INTEGER N;
ARRAY A,VECTORS[0,0], VALUES[0];
```

Purpose.
SYMEIGENVECTORS finds all N eigenvalues and normalized column eigenvectors of a real symmetric matrix. Only the lower triangle of the matrix is required.

Input.
Input to SYMEIGENVECTORS consists of:

N - order of matrix

A - the lower triangle of the matrix is stored in the lower triangle of locations 1:N,1:N of the array A[0:N,0:N]. The strict upper triangle of A is unused.

Output.
Output for SYMEIGENVECTORS consists of:

A - the lower triangle of A is destroyed

VALUES - the eigenvalues are placed in locations 1:N of the array VALUES[0:N] in ascending order

VECTORS - the normalized eigenvectors are placed in locations 1:N of the columns of the array VECTORS[0:N,0:N]

Method.
The matrix is first reduced to tridiagonal form using the Householder reduction. The eigenvalues and eigenvectors of the tridiagonal matrix are found with QL transformations. These eigenvectors are then transformed into eigenvectors of the original matrix using information stored from the Householder reduction and normalized such that each eigenvector has its component of maximum magnitude equal to 1.0.

Remarks.
If the iteration fails to converge, an error message is printed and the procedure is exited. Failures are exceedingly rare.

Reference.
This is a modified version of procedures appearing in: Bowdler, H.; Martin, R. S.; Reinsch, C.; and Wilkinson, J. H. 1968. The QR and QL Algorithms for Symmetric Matrices. Numerische Mathematik 11:293-306.

SYMBANDEIGENVALUES PROCEDURE. The syntax is as follows:

```
PROCEDURE SYMBANDEIGENVALUES(N,M,A,VALUES);
VALUE N,M; INTEGER N,M;
ARRAY A[0,0], VALUES[0];
```

Purpose.
SYMBANDEIGENVALUES finds all N eigenvalues of a symmetric positive-definite band matrix. A special storage arrangement is used.

Input.
Input to SYMBANDEIGENVALUES consists of:

N - order of system

M - number of lines on either side of the main diagonal

A - the lower half of the matrix is stored in locations 1:N,0:M of the array A[0:N,0:M]. The subdiagonals of the matrix are stored in the columns of A, with the main diagonal in A[I,M], I = 1 (1) N. The elements of the stored array occupy the positions illustrated below for N=5, M=2 of a conventional array.

Stored Array			Lower Triangle of Conventional Array				
X	X	A[1,2]	A[1,2]				
X	A[2,1]	A[2,2]	A[2,1]	A[2,2]			
A[3,0]	A[3,1]	A[3,2]	A[3,0]	A[3,1]	A[3,2]		
A[4,0]	A[4,1]	A[4,2]		A[4,0]	A[4,1]	A[4,2]	
A[5,0]	A[5,1]	A[5,2]			A[5,0]	A[5,1]	A[5,2]

The elements indicated by X's in the upper left-hand corner of the stored array may be arbitrary since they are not used. This storage arrangement is the same as that used in SYMBANDLINEQN.

Output.
Output for SYMBANDEIGENVALUES consists of:

A - A[M,I], I=1(1)N contains the main diagonal of the reduced matrix, and A[M-1,I], I=2(1)N contains the first lower subdiagonal

VALUES - the eigenvalues are placed in locations 1:N of the array VALUES[0:N] in ascending order

Method.
The matrix is reduced to tridiagonal form by a sequence of Jacobi rotations which preserve the eigenvalues and the band structure. The eigenvalues of the tridiagonal matrix are found using QL transformations; the procedure employed inside SYMEIGENVALUES is also used here.

Remarks.
If the eigenvalue iteration fails to converge, an error message is printed and the procedure is exited. Failures are exceedingly rare.

References.
Bowdler, H.; Martin, R. S.; Reinsch, C.; and Wilkinson, J. H. 1968. QR and QL Algorithms for Symmetric Matrices. Numerische Mathematik 11:293-306.

Schwarz, H. R. 1968. The Tridiagonalization of a Symmetric Band Matrix. Numerische Mathematik 12:231-241.

COMPLEX MATRICES.

COMPLEXEIGENVALUES PROCEDURE. The syntax is as follows:

```
PROCEDURE COMPLEXEIGENVALUES(N,AR,AI,WR,WI);
VALUE N; INTEGER N;
ARRAY AR,AI[0,0], WR,WI[0];
```

Purpose.
COMPLEXEIGENVALUES finds all N eigenvalues of the general complex matrix A.

Input.
Input to COMPLEXEIGENVALUES consists of:

N - complex order of the complex matrix A

AR,AI - the real and imaginary parts of A are input in locations 1:N, 1:N of the arrays AR,AI[0:N,0:N]

Output.
Output for COMPLEXEIGENVALUES consists of:

AR,AI - the given matrix is destroyed in the process

WR,WI - the real and imaginary parts of the eigenvalues are placed in locations 1:N of the arrays WR,WI[0:N], respectively.

Method.
A is first prepared by procedure COMPLEXBALANCE, which detects any isolated eigenvalues, reduces the norm of the matrix, and equalizes the L1 norm of corresponding rows and columns, all without introducing round-off error. The balanced matrix is next reduced to upper Hessenberg form by procedure COMHES utilizing stabilized elementary similarity transformations. The eigenvalues of the upper Hessenberg matrix (which are also the eigenvalues of A) are computed by procedure COMLR which employs the LR algorithm with partial pivoting.

Remarks.
Balancing is strongly recommended, since it generally improves accuracy at a small additional cost in time. However, it is not necessary to balance normal matrices. If balancing is not used, LOW must be set to 1 and UPP must be set to N.

If the original matrix is already of upper Hessenberg form (zeroes below the first subdiagonal), the call to COMHES may be removed (procedure COMPLEXBALANCE preserves the Hessenberg form).

The procedure fails if any eigenvalue requires more than 30 iterations, in which case an error message is printed. Failures are extremely rare.

Reference.
Martin, R. S. and Wilkinson, J. H. 1968. The Modified LR Algorithm for Complex Hessenberg Matrices. Numerische Mathematik 12:369-376.

COMPLEXEIGENVECTORS PROCEDURE. The syntax is as follows:

```
PROCEDURE COMPLEXEIGENVECTORS(N,AR,AI,WR,WI,VR,VI);
VALUE N; INTEGER N;
ARRAY AR,AI,VR,VI[0,0], WR,WI[0];
```

Purpose.
COMPLEXEIGENVECTORS finds all N eigenvalues and normalized column eigenvectors of the general complex matrix A.

Input.
Input to COMPLEXEIGENVECTORS consists of:

N - complex order of the complex matrix A

AR,AI - the real and imaginary parts of A are input in locations 1:N,1:N of the arrays AR,AI[0:N,0:N]

Output.
Output for COMPLEXEIGENVECTORS consists of:

AR,AI - the strict upper triangles of AR,AI are overwritten with the eigenvectors of the triangular matrix produced by the LR algorithm

WR,WI - the real and imaginary parts of the eigenvalues are placed in location 1:N of the arrays WR,WI[0:N]

VR,VI - the real and imaginary parts of the normalized eigenvectors are stored in the columns (in locations 1:N) of the arrays VR,VI[0:N,0:N]

Method.
A is first prepared by procedure COMPLEXBALANCE, which detects any isolated eigenvalues, reduces the norm of the matrix, and equalizes the L1 norms of corresponding rows and columns, all without introducing round-off error. The balanced matrix is next reduced to upper Hessenberg form by procedure COMHES utilizing stabilized elementary similarity transformations.

After completing the preliminary reductions, LR iterations with partial pivoting and a back-substitution process are employed by COMLR2 to compute the eigenvalues of A and the eigenvectors of the balanced matrix, which are then transformed into eigenvectors of the original matrix A by two calls to procedure BALBAK. Finally, each eigenvector is normalized so that its component of maximum modulus is equal to 1.0 + 0xI.

Remarks.
Balancing is strongly recommended since it generally improves accuracy at a small additional cost in time. However, it is not necessary to balance normal matrices. If balancing is not used, LOW must be set to 1 and UPP must be set to N.

If the original matrix is already of upper Hessenberg form (zeroes below the first subdiagonal), the call to COMHES may be removed (procedure COMPLEXBALANCE preserves the Hessenberg form). If unnormalized eigenvectors are sufficient, the call to procedure COMPLEXNORMALIZE may be omitted.

The procedure fails if any eigenvalue requires more than 30 iterations, in which case an error message is printed. Failures are extremely rare.

Reference.

Peters, G. and Wilkinson, J. H. 1970. Eigenvectors of Real and Complex Matrices by LR and QR Triangularizations. Numerische Mathematik 16:181-204.

INVARIANTSUBSPACES PROCEDURE. The syntax is as follows:

```
PROCEDURE INVARIANTSUBSPACES(N,A,RTR,RTI,X,CL1);
VALUE N; INTEGER N;
ARRAY A,X[0,0], RTR,RTI[0];
INTEGER ARRAY CL1[0];
```

Purpose.
INVARIANTSUBSPACES finds all N eigenvalues of the complex matrix A and the invariant subspace of dimension K associated with each eigenvalue of multiplicity K (K may be 1, in which case the 1-dimensional invariant subspace is an eigenvector).

Input.
Input to INVARIANTSUBSPACES consists of:

N - complex dimension of complex matrix A

A - complex general matrix whose I,Jth element is stored in locations A[I,2xJ - 1] and A[I,2xJ] of the array A[0:N,0:2xN], i.e., real and imaginary parts are stored in consecutive columns of A

Output.
Output for INVARIANTSUBSPACES consists of:

A - given matrix is destroyed in the process

RTR - the real parts of the eigenvalues are placed in locations 1:N of the array RTR[0:N]

RTI - the imaginary parts of the eigenvalues are placed in locations 1:N of the array RTI[0:N]

X - the basis vectors of the invariant subspaces of A are placed in the columns of X[0:N,0:2xN] with the real and imaginary parts of each basis vector stored consecutively, beginning in column 1

CL1 - the dimensions of the invariant subspaces (the multiplicity of the eigenvalues) are stored in locations 1:N of the array CL1[0:N]; e.g., if CL1[1] = CL1[2] = 2, the first eigenvalue is double and columns 1-4 of X span the 2-dimensional invariant subspace corresponding to it

Method.
A is first reduced to upper Hessenberg form. The eigenvalues of A are found by applying the LR algorithm to the reduced matrix using procedure COMLR. The invariant subspaces are found by generalized inverse iteration. The procedure fails if any eigenvalue takes more than 30 iterations, in which case an error message is printed and the procedure is exited.

Remarks.
This procedure should be used only if A is known to have multiple eigenvalues and only then if the eigenvectors associated with a multiple eigenvalue are nearly parallel. In the absence of such knowledge, procedure COMPLEXEIGENVECTORS should be used first.

Reference.
Varak, James M. 1967. The Computation of Bounds for the Invariant Subspaces of a General Matrix Operator. Technical Report No. CS 66. Stanford, Calif.:Computer Science Dept., Stanford University.

COMLR is taken from procedure COMPLEXEIGENVALUES.

MATRIX_ARITHMETIC.

SQUARE MATRICES.

MATPLUSMAT PROCEDURE. The syntax is as follows:

```
PROCEDURE MATPLUSMAT(N,A,B,C);
VALUE N; INTEGER N;
ARRAY A,B,C[0,0];
```

Purpose.
MATPLUSMAT computes sum A+B of the matrices A[1:N,1:N] and B[1:N,1:N] and stores the result in C[1:N,1:N].

Parameters.
Parameters for MATPLUSMAT consist of:

N - dimension of A, B, and C
A - first input matrix
B - second input matrix
C - output matrix

Method.
The matrices are added element by element.

Remarks.
Either A or B may be overwritten with C by the procedure calls MATPLUSMAT(N,A,B,A) or MATPLUSMAT(N,A,B,B), respectively.

MATMINUSMAT PROCEDURE. The syntax is as follows:

```
PROCEDURE MATMINUSMAT(N,A,B,C);
VALUE N; INTEGER N;
ARRAY A,B,C[0,0];
```

Purpose.
MATMINUSMAT computes difference A-B of the matrices A[1:N,1:N] and B[1:N,1:N] and stores the result in C[1:N,1:N].

Parameters.
Parameters for MATMINUSMAT consist of:

N - dimension of A, B, and C
A - first input matrix
B - second input matrix
C - output matrix

Method.
Elements of matrix B are subtracted from the corresponding elements of matrix A.

Remarks.
Either A or B may be overwritten with C by the procedure calls MATMINUSMAT(N,A,B,A) or MATMINUSMAT(N,A,B,B), respectively.

MATTIMESMAT PROCEDURE. The syntax is as follows:

```
PROCEDURE MATTIMESMAT(N,A,B,C);
VALUE N; INTEGER N;
ARRAY A,B,C[0,0];
```

Purpose.
MATTIMESMAT computes the matrix product C = AxB, where the matrices A and B are stored in A[1:N,1:N] and B[1:N,1:N]. The product is stored in C[1:N,1:N].

Parameters.
Parameters for MATTIMESMAT consist of:

N - dimension of A, B, and C
A - first input matrix
B - second input matrix
C - output matrix

Method.
C[I,J] is defined to be the innerproduct of the Ith row of A with the Jth column of B.

Remarks.
A, B, and C must occupy separate storage.

TRANSPOSE PROCEDURE. The syntax is as follows:

```
PROCEDURE TRANSPOSE(N,A);
VALUE N; INTEGER N;
ARRAY A[0,0];
```

Purpose.
TRANSPOSE replaces the matrix stored in A[1:N,1:N] by its transpose.

Parameters.
Parameters for TRANSPOSE consist of:

N - dimension of A
A - matrix to be transposed

Method.
The I,Jth element of A is replaced by the J,Ith element.

ATIMESATRANSPOSE PROCEDURE. The syntax is as follows:

```
PROCEDURE ATIMESATRANSPOSE(N,A,C);
VALUE N; INTEGER N;
ARRAY A,C[0,0];
```

Purpose.
ATIMESATRANSPOSE computes the product of the matrix stored in A[1:N,1:N] with its transpose and stores the result in C[1:N,1:N], i.e., C = AxATRANSPOSE.

Parameters.
Parameters for ATIMESATRANSPOSE consist of:

N - dimension of A and C
A - input matrix
C - output matrix

Method.
C[I,J] is defined to be the inner-product of the Ith row of A with the Jth column of A.

Remarks.
A and C must occupy separate storage.

ATIMESBTRANSPOSE PROCEDURE. The syntax is as follows:

```
PROCEDURE ATIMESBTRANSPOSE(N,A,B,C);
VALUE N; INTEGER N;
ARRAY A,B,C[0,0];
```

Purpose.
ATIMESBTRANSPOSE computes the product of the matrix stored in A[1:N,1:N] with the transpose of the matrix stored in B[1:N,1:N], and stores the result in C[1:N,1:N], i.e., C = AxBTRANSPOSE.

Parameters.
Parameters for ATIMESBTRANSPOSE consist of:

N - dimension of A, B, and C
A - first input matrix
B - second input matrix
C - output matrix

Method.
The transpose of matrix B is not actually calculated. Instead, the elements of matrix B are taken by rows rather than by columns for pre-multiplication by matrix A.

Remarks.
A, B, and C must all occupy separate storage.

MATTIMESVEC PROCEDURE. The syntax is as follows:

```
PROCEDURE MATTIMESVEC(N,A,X,Y);
VALUE N; INTEGER N;
ARRAY A[0,0],X,Y[0];
```

Purpose.
MATTIMESVEC computes the product Y = AxX of the NxN matrix stored in A[1:N,1:N] and the Nx1 column vector stored in X[1:N]. The result is stored in Y[1:N].

Parameters.
Parameters for MATTIMESVEC consist of:

N - square dimension of A and linear dimension of X and Y
A - input matrix
X - input vector
Y - output vector

Method.
Y[I] is defined to be the innerproduct of the Ith row of A with X.

Remarks.
X and Y must occupy separate storage.

VECTIMESMAT PROCEDURE. The syntax is as follows:

```
PROCEDURE VECTIMESMAT(N,A,X,Y);
VALUE N; INTEGER N;
ARRAY A[0,0], X,Y[0];
```

Purpose.
VECTIMESMAT computes the product Y = XxA of the 1xN row vector stored in X[1:N] with the matrix stored in A[1:N,1:N]. The result is stored in Y[1:N].

Parameters.
Parameters for VECTIMESMAT consist of:

N - square dimension of A and linear dimension of X and Y
A - input matrix
X - input vector
Y - output vector

Method.
The Ith component of Y is defined to be the innerproduct of X with the Ith column of A.

Remarks.
X and Y must occupy separate storage.

VECPLUSRVEC PROCEDURE. The syntax is as follows:

```
PROCEDURE VECPLUSRVEC(N,AI,R,X,BI);
VALUE N,R; INTEGER N; REAL R;
ARRAY AI,X,BI[0];
```

Purpose.
VECPLUSRVEC adds the scalar multiple RxX of the vector stored in X[1:N] by the real number R to the vector stored in AI[1:N]. The result is stored in BI[1:N]. The procedure call VECPLUSRVEC(N,A[I,*],R,X,A[I,*]) may be used to operate on the Ith row of a matrix A, and the call VECPLUSRVEC(N,A[I,*],R,A[K,*],A[I,*]) may be used to add a multiple of the Kth row of a matrix to the Ith row.

Parameters.
Parameters for VECPLUSRVEC consist of:

N - dimension of AI, X, and BI
AI - first input vector
R - real multiplier
X - second input vector
BI - output vector

Method.
The vector BI is defined by BI[J] = AI[J] + RxX[J].

MATPLUSDIAG PROCEDURE. The syntax is as follows:

```
PROCEDURE MATPLUSDIAG(N,D,A,C);
VALUE N; INTEGER N;
ARRAY A,C[0,0], D[0];
```

Purpose.
MATPLUSDIAG adds the diagonal matrix which is stored as a 1-dimensional array in D[1:N] to the matrix stored in A[1:N,1:N] and stores the result in C[1:N,1:N], i.e., C = A+D.

Parameters.
Parameters for MATPLUSDIAG consist of:

- N - square dimension of A and linear dimension of D
- D - input diagonal matrix stored as a vector
- A - input matrix
- C - output matrix

Method.
The elements of D are added to the corresponding diagonal elements of A.

Remarks.
A may be overwritten with C using the procedure call MATPLUSDIAG(N,D,A,A).

MATMINUSDIAG PROCEDURE. The syntax is as follows:

```
PROCEDURE MATMINUSDIAG(N,D,A,C);
VALUE INTEGER N;
ARRAY A,C[0,0], D[0];
```

Purpose.
MATMINUSDIAG subtracts the diagonal matrix which is stored as a 1-dimensional array in D[1:N] from the matrix stored in A[1:N,1:N] and stores the result in C[1:N,1:N], i.e., C =A-D.

Parameters.
Parameters for MATMINUSDIAG consist of:

N - square dimension of A and linear dimension of D
D - input diagonal matrix stored as a vector
A - input matrix
C - output matrix

Method.
The elements of D are subtracted from the corresponding diagonal elements of A.

Remarks.
A may be overwritten with C using the procedure call MATMINUSDIAG(N,D,A,A).

MATTIMESDIAG PROCEDURE. The syntax is as follows:

```
PROCEDURE MATTIMESDIAG(N,D,A,C);
VALUE N; INTEGER N;
ARRAY A,C[0,0], D[0];
```

Purpose.
MATTIMESDIAG postmultiplies the matrix stored in A[1:N,1:N] by the diagonal matrix which is stored as a 1-dimensional array in D[1:N] and stores the result in C[1:N,1:N], i.e., C = AxD.

Parameters.
Parameters for MATTIMESDIAG consist of:

N - square dimension of A and linear dimension of D
D - input diagonal matrix stored as a vector
A - input matrix
C - output matrix

Method.
The elements of the Jth column of A are multiplied by D[J] to obtain the Jth column of C.

Remarks.
A may be overwritten with C using the procedure call MATTIMESDIAG(N,D,A,A).

DIAGTIMESMAT PROCEDURE. The syntax is as follows:

```
PROCEDURE DIAGTIMESMAT(N,D,A,C);
VALUE N; INTEGER N;
ARRAY A,C[0,0], D[0];
```

Purpose.
DIAGTIMESMAT premultiplies the matrix stored in A[1:N,1:N] by the diagonal matrix which is stored as a 1-dimensional array in D[1:N] and stores the result in C[1:N,1:N], i.e., C = DxA.

Parameters.
Parameters for DIAGTIMESMAT consist of:

N - square dimension of A and linear dimension of D
D - input diagonal matrix stored as a vector
A - input matrix
C - output matrix

Method.
The elements of the Ith row of A are multiplied by D[I] to obtain the Ith row of C.

Remarks.
A may be overwritten with C using the procedure call DIAGTIMESMAT(N,D,A,A).

RPLUSMAT PROCEDURE. The syntax is as follows:

```
PROCEDURE RPLUSMAT(N,R,A,B);
VALUE N,R; INTEGER N; REAL R;
ARRAY A,B[0,0];
```

Purpose.
RPLUSMAT adds the real number R to each component of the matrix A[1:N,1:N] and stores the result in B[1:N,1:N], i.e., B = R+A.

Parameters.
Parameters for RPLUSMAT consist of:

N - dimension of A and B
R - input real number
A - input matrix
B - output matrix

Remarks.
A may be overwritten with B using the procedure call RPLUSMAT(N,R,A,A).

RMINUSMAT PROCEDURE. The syntax is as follows:

```
PROCEDURE RMINUSMAT(N,R,A,B);
VALUE N,R; INTEGER N; REAL R;
ARRAY A,B[0,0];
```

Purpose.
RMINUSMAT subtracts each component of the matrix stored in A[1:N,1:N] from the real number R and stores the result in B[1:N,1:N], i.e., B = R-A.

Parameters.
Parameters for RMINUSMAT consist of:

N - dimension of A and B
R - input real number
A - input matrix
B - output matrix

Remarks.
A may be overwritten with B using the procedure call RMINUSMAT(N,R,A,A).

RTIMESMAT PROCEDURE. The syntax is as follows:

```
PROCEDURE RTIMESMAT(N,R,A,B);
VALUE N,R; INTEGER N; REAL R;
ARRAY A,B[0,0];
```

Purpose.
RTIMESMAT multiplies each component of the matrix stored in A[1:N,1:N] by the real number R and stores the result in B[1:N,1:N], i.e., B = RxA.

Parameters.
Parameters for RTIMESMAT consist of:

N - dimension of A and B
R - input real number
A - input matrix
B - output matrix

Remarks.
A may be overwritten with B using the procedure call RTIMESMAT(N,R,A,A).

MATCOPY PROCEDURE. The syntax is as follows:

```
PROCEDURE MATCOPY(N,A,B);
VALUE N; INTEGER N;
ARRAY A,B[0,0];
```

Purpose.
MATCOPY copies the matrix stored in A[1:N,1:N] onto the matrix stored in B[1:N,1:N].

Parameters.
Parameters for MATCOPY consist of:

N - dimension of A and B
A - matrix to be copied
B - output matrix

Method.
A is copied onto B one row at a time.

RECTANGULAR MATRICES.

RECMATPLUSMAT PROCEDURE. The syntax is as follows:

```
PROCEDURE RECMATPLUSMAT(N,M,A,B,C);
VALUE N,M; INTEGER N,M;
ARRAY A,B,C[0,0];
```

Purpose.
RECMATPLUSMAT computes the sum A+B of the rectangular matrices A[1:N,1:M] and B[1:N,1:M] and stores the result in C[1:N,1:M].

Parameters.
Parameters for RECMATPLUSMAT consist of:

N - row dimension of A, B, and C
M - column dimension of A, B, and C
A - first input matrix
B - second input matrix
C - output matrix

Method.
The matrices are added element by element.

Remarks.
Either A or B may be overwritten with C by the procedure call RECMATPLUSMAT(N,M,A,B,A) or RECMATPLUSMAT(N,M,A,B,B).

RECMATMINUSMAT PROCEDURE. The syntax is as follows:

```
PROCEDURE RECMATMINUSMAT(N,M,A,B,C);
VALUE N,M; INTEGER N,M;
ARRAY A,B,C[0,0];
```

Purpose.
RECMATMINUSMAT computes the difference A-B of the rectangular matrices A[1:N,1:M] and B[1:N,1:M] and stores the result in C[1:N,1:M].

Parameters.
Parameters for RECMATMINUSMAT consist of:

N - row dimension of A, B, and C
M - column dimension of A, B, and C
A - first input matrix
B - second input matrix
C - output matrix

Method.
The matrices are subtracted element by element.

Remarks.
Either A or B may be overwritten with C by the procedure call RECMATMINUSMAT(N,M,A,B,A) or RECMATMINUSMAT(N,M,A,B,B).

RECMATTIMESMAT PROCEDURE. The syntax is as follows:

```
PROCEDURE RECMATTIMESMAT(N,K,M,A,B,C);
VALUE N,K,M; INTEGER N,K,M;
ARRAY A,B,C[0,0];
```

Purpose.
RECMATTIMESMAT computes the product of the N-rowed, K-columned matrix A[1:N,1:K] and the K-rowed, M-columned matrix B[1:K,1:M], and stores the result in C[1:N,1:M].

Parameters.
Parameters for RECMATTIMESMAT consist of:

N - row dimension of A and C
K - column dimension of A and row dimension of B
M - column dimension of B and C
A - first input matrix
B - second input matrix
C - output matrix

Method.
The I,Jth element of C is defined to be the innerproduct of the Ith row of A and the Jth column of B.

Remarks.
A, B, and C must occupy separate storage.

SECTION 2

POLYNOMIALS

ROOTS OF POLYNOMIALS.

REALPOLYZEROFINDER PROCEDURE.
The syntax is as follows:

```
PROCEDURE REALPOLYZEROFINDER(DEGREE,A,ZEROR,ZEROI);
VALUE DEGREE;
INTEGER DEGREE;
ARRAY P,ZEROR,ZEROI[0];
```

PURPOSE. REALPOLYZEROFINDER finds all the zeroes of a real polynomial P.

DESCRIPTION OF PARAMETERS. The parameters are:

DEGREE - degree of the polynomial

P - array of coefficients of the polynomial, the coefficients of Z**J being stored in P[N-J]; dimension P[0:DEGREE]

ZEROR - array of real part of zeroes of the polynomial, the real part of the Kth zero being stored in ZEROR[K]; dimension ZEROR[0:DEGREE]

ZEROI - array of imaginary parts of zeroes of the polynomial, the imaginary part of the Kth zero being stored in ZEROI[0:DEGREE]

PARAMETERS RETURNED. The parameters returned are:

ZEROR
ZEROI

METHOD. A 3-stage, variable-shift iteration due to M. A. Jenkins and J. F. Traub is used. The zeroes are calculated one or two at a time in roughly increasing order of magnitude, using real arithmetic wherever possible. Failure of the iteration to converge for a root causes an error message to be printed and the procedure to be exited with the zeroes already found stored in the arrays ZEROR and ZEROI. Failures are rare.

REFERENCE. Jenkins, M. A. 1969. Three Stage Variable-Shift Iterations for the Solution of Polynomial Equations with a Posteriori Error Bounds for the Zeros. Technical Report No. CS 138. Stanford, Calif.:Computer Science Dept., Stanford University.

COMPLEXPOLYZEROFINDER PROCEDURE.
The syntax is as follows:

```
PROCEDURE COMPLEXPOLYZERORFINDER(DEGREE,PR,PI,ZEROR,ZEROI);
VALUE DEGREE;
INTEGER DEGREE;
ARRAY PR,PI,ZEROR,ZEROI[0];
```

PURPOSE. COMPLEXPOLYZERORFINDER finds all the zeroes of the complex polynomial P(Z).

DESCRIPTION OF PARAMETERS. The parameters are:

DEGREE - degree of the polynomial

PR - array of real part of coefficients of the polynomial, the real part of the coefficient of Z**J being stored in P[N-J]; dimension PR[0:DEGREE]

PI - array of imaginary part of coefficients of the polynomial, the imaginary part of the coefficient of Z**J being stored in P[N-J]; dimension PI[0:DEGREE]

ZEROR - array of real part of zeroes of the polynomial, the real part of the Kth zero being stored in ZERORR[K]; dimension ZERORR[0:DEGREE]

ZEROI - array of imaginary part of zeroes of the polynomial, the imaginary part of the Kth zero being stored in ZEROI[K]; dimension ZEROI[0:DEGREE]

PARAMETERS RETURNED. The parameters returned are:

ZEROR
ZEROI

METHOD. A 3-stage, variable-shift iteration due to M. A. Jenkins and J. F. Traub is used. The zeroes are calculated one at a time in roughly increasing order of magnitude using complex arithmetic. Failure of the iteration to converge for a root causes an error message to be printed and the procedure to be exited with the zeroes already found stored in the arrays ZEROR and ZEROI. Failures are rare.

REFERENCE. Jenkins, M. A. 1969. Three Stage Variable-Shift Iterations for the Solution of Polynomial Equations with a Posteriori Error Bounds for the Zeros. Technical Report No. CS 138. Stanford, Calif.:Computer Science Dept., Stanford University.

OPERATIONS ON POLYNOMIALS.

POLYADD PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYADD(N1DEG,COEF1,N2DEG,COEF2,NRDEG,COEFR);
VALUE N1DEG,N2DEG;
INTEGER N1DEG,N2DEG,NRDEG;
ARRAY COEF1,COEF2,COEFR[0];
```

PURPOSE. POLYADD adds two polynomials.

DESCRIPTION OF PARAMETERS. The parameters are:

N1DEG - degree of first polynomial

COEF1 - array of coefficients for first polynomial ordered from largest to smallest power; dimension [0:N1DEG]

N2DEG - degree of second polynomial

COEF2 - array of coefficients for second polynomial ordered from largest to smallest power; dimension [0:N2DEG]

NRDEG - degree of resultant polynomial

COEFR - array of coefficients for resultant polynomial ordered from largest to smallest power; dimension [0:NRDEG]

PARAMETERS RETURNED. The parameters returned are:

NRDEG
COEFR

METHOD. The degree of resultant polynomial NRDEG is calculated as the larger of the two degrees N1DEG,N2DEG. Corresponding coefficients are then added to form COEFR.

POLYSUB PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYSUB(N1DEG,COEF1,N2DEG,COEF2,NRDEG,COEFR);
VALUE N1DEG,N2DEG;
INTEGER N1DEG,N2DEG,NRDEG;
ARRAY COEF1,COEF2,COEFR[0];
```

PURPOSE. POLYSUB subtracts one polynomial from another.

DESCRIPTION OF PARAMETERS. The parameters are:

N1DEG - array of coefficients for first polynomial ordered from largest to smallest power; dimension [0:N1DEG]

N2DEG - degree of second polynomial

COEF2 - array of coefficients for second polynomial ordered from largest to smallest power; dimension [0:N2DEG]

NRDEG - degree of resultant polynomial

COEFR - array of coefficients for resultant polynomial ordered from largest to smallest power; dimension [0:NRDEG]

PARAMETERS RETURNED. The parameters returned are:

NRDEG
COEFR

METHOD. The degree of resultant polynomial NRDEG is calculated as the larger of the two degrees N1DEG,N2DEG. Coefficients in array COEF2 are then subtracted from corresponding coefficients in array COEF1.

POLYMUL, PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYMUL(N1DEG,COEF1,N2DEG,COEF2,NRDEG,COEFR);
VALUE N1DEG,N2DEG;
INTEGER N1DEG,N2DEG,NRDEG;
ARRAY COEF1,COEF2,COEFR[0];
```

PURPOSE. POLYMUL multiplies two polynomials.

DESCRIPTION OF PARAMETERS. The parameters are:

N1DEG - degree of first polynomial

COEF1 - array of coefficients for first polynomial ordered from largest to smallest power; dimension [0:N1DEG]

N2DEG - degree of second polynomial

COEF2 - array of coefficients for second polynomial ordered from largest to smallest power; dimension [0:N2DEG]

NRDEG - degree of resultant polynomial

COEFR - array of coefficients for resultant polynomial ordered from largest to smallest power; dimension [0:NRDEG]

PARAMETERS RETURNED. The parameters returned are:

NRDEG
COEFR

METHOD. The degree of resultant polynomial, NRDEG, is calculated as N1DEG+N2DEG. The coefficients of COEFR are calculated as the sum of products of coefficients of COEF1 and COEF2 whose exponents add up to the corresponding exponent of COEFR.

REMARKS. COEFR cannot be in the same location as COEF1 or COEF2.

POLYDIV PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYDIV(N1DEG,COEF1,N2DEG,COEF2,NQDEG,COEFQ,TOL,ERRCOD);
VALUE N1DEG,N2DEG,TOL;
INTEGER N1DEG,N2DEG,NQDEG,ERRCOD;
REAL TOL;
ARRAY COEF1,COEF2,COEFQ[0];
```

PURPOSE. POLYDIV divides one polynomial by another.

DESCRIPTION OF PARAMETERS. The parameters are:

N1DEG - dividend polynomial degree, replaced by degree of remainder

COEF1 - array of coefficients for dividend polynomial ordered from largest to smallest power; it is replaced by remainder after division; dimension [0:N1DEG]

N2DEG - degree of divisor polynomial

COEF2 - array of coefficients for divisor polynomial ordered from largest to smallest power; dimension [0:N2DEG]

NQDEG - degree of quotient polynomial

COEFQ - array of coefficients for quotient polynomial ordered from largest to smallest power; dimension [0:NQDEG]

TOL - tolerance value below which coefficients are eliminated during normalization

ERRCOD - error code; 0 is normal; 1 is for zero divisor

PARAMETERS RETURNED. Parameters returned are:

N1DEG
COEF1
NQDEG
COEFQ
ERRCOD

METHOD. Polynomial COEF1 is divided by polynomial COEF2, giving quotient part COEFQ and a remainder such that COEF1 = COEFQxCOEF2 + remainder. Divisor COEF2 and the remainder array get normalized.

REMARKS. The remainder replaces COEF1. If N2DEG=0, N1DEG is set to -1. The devisor COEF2 is unchanged. If the degree of the devisor polynomial exceeds the dividend, NQDEG is set to -1 and the calculation is bypassed.

REQUIREMENTS. The other procedure required is POLYNORM.

POLYVAL PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYVAL(NDEG,COEF,ARG,RES);
VALUE NDEG,ARG;
INTEGER NDEG;
REAL ARG,RES;
ARRAY COEF[0];
```

PURPOSE. POLYVAL evaluates a polynomial for a given value of the variable.

DESCRIPTION OF PARAMETERS. The parameters are:

NDEG - degree of polynomial

COEF - array of coefficients ordered from largest to smallest power; dimension [0:NDEG]

ARG - given value of variable

RES - resultant value of polynomial

PARAMETER RETURNED. The parameter returned is:

RES

METHOD. Evaluation is done by means by nested multiplication.

POLYDER PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYDER(NDEG,COEF,NDDEG,COEFD);
VALUE NDEG;
INTEGER NDEG,NDDEG;
ARRAY COEF,COEFD[0];
```

PURPOSE. POLYDER finds the derivative of a polynomial.

DESCRIPTION OF PARAMETERS. The parameters are:

NDEG - degree of original polynomial

COEF - array of coefficients for original polynomial, ordered from largest to smallest power; dimension [0:NDEG]

NDDEG - degree of derivative (equal to NDEG-1)

COEFD - array of coefficients for derivative, ordered from largest to smallest power; dimension [0:NDDEG]

PARAMETERS RETURNED. Parameters returned are:

NDDEG
COEFD

METHOD. NDDEG is set at NDEG-1. The derivative is then calculated by multiplying the coefficients by their respective exponents.

POLYINT PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYINT(NDEG,COEF,C,NIDEG,COEFI);
VALUE NDEG,C;
INTEGER NDEG,NIDEG;
REAL C;
ARRAY COEF,COEFI[0];
```

PURPOSE. POLYINT finds the integral of a polynomial.

DESCRIPTION OF PARAMETERS. The parameters are:

C - constant of integration

NDEG - degree of original polynomial

COEF - array of coefficients for original polynomial, ordered from largest to smallest power; dimension [0:NDEG]

NIDEG - degree of integral (equal to NDEG+1)

COEFI - array of coefficients for integral, ordered from largest to smallest power; dimension [0:N1DEG]

PARAMETERS RETURNED. Parameters returned are:

NIDEG
COEFI

METHOD. NIDEG is set at NDEG+1, and the constant term is set at C. The integral is then calculated by dividing the coefficients by their respective exponents.

POLYPOP PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYPOP(N1DEG,COEF1,N2DEG,COEF2,NRDEG,COEFR);
VALUE N1DEG,N2DEG;
INTEGER N1DEG,N2DEG,NRDEG;
ARRAY COEF1,COEF2,COEFR[0];
```

PURPOSE. POLYPOP substitutes the variable of a polynomial by another polynomial.

DESCRIPTION OF PARAMETERS. The parameters are:

N1DEG - degree of original polynomial

COEF1 - array of coefficients for original polynomial, ordered from highest to smallest power; dimension [0:N1DEG]

N2DEG - degree of polynomial which is substituted for variable

COEF2 - array of coefficients for polynomial which is substituted for variable, ordered from highest to smallest power; dimension [0:N2DEG]

NRDEG - degree of resultant polynomial

COEFR - array of coefficients for resultant polynomial ordered from highest to smallest power; dimension [0:NRDEG]

PARAMETERS RETURNED. Parameters returned are:

NRDEG
COEFR

METHOD. The variable of polynomial COEF1 is substituted by polynomial COEF2 to form polynomial COEFR. The degree of the resultant polynomial is N1DEGxN2DEG.

REMARKS. If either N1DEG or N2DEG is negative, NRDEG is set at -1 and the calculation is bypassed.

OTHER PROCEDURES REQUIRED. Other procedures required are:

NRDEG
COEFR

POLYMAD PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYMAD(N1DEG,COEF1,C,N2DEG,COEF2,NRDEG,COEFR);
VALUE C,N1DEG,N2DEG;
INTEGER N1DEG,N2DEG,NRDEG;
REAL C;
ARRAY COEF1,COEF2,COEFR[0];
```

PURPOSE. POLYMAD adds coefficients of one polynomial to the product of a factor by coefficients of another polynomial.

DESCRIPTION OF PARAMETERS. The parameters are:

N1DEG - degree of first polynomial

COEF1 - array of coefficients for first polynomial, ordered from largest to smallest power; dimension [0:N1DEG]

N2DEG - degree of second polynomial

COEF2 - array of coefficients for second polynomial, ordered from largest to smallest power; dimension [0:N2DEG]

C - factor to be multiplied by array COEF2

NRDEG - degree of resultant polynomial

COEFR - array of resultant coefficients, ordered from largest to smallest power; dimension [0:NRDEG]

METHOD. The degree of the resultant polynomial NRDEG is calculated as the larger of the two degrees N1DEG,N2DEG. Coefficients in array COEF1 are then added to corresponding coefficients in array COEF2 and multiplied by C to form COEFR.

POLYHOR PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYHOR(NDEG,COEF,BOOL,ARG,R,K);
VALUE NDEG,K,ARG;
INTEGER NDEG,K;
REAL ARG;
ARRAY COEF,R[0];
BOOLEAN BOOL;
```

PURPOSE. POLYHOR finds the values of the first K derivatives of a polynomial for a given value of the variable, or calculates the values of the first K+1 coefficients of the expansion of the polynomial in a power series in the neighborhood of a given point.

DESCRIPTION OF PARAMETERS. The parameters are:

NDEG - degree of polynomial

COEF - array of coefficients ordered from largest to smallest power; dimension [0:NDEG]

BOOL - Boolean variable whose value is TRUE if the values of the first K derivatives are desired. If the expansion of the polynomial in a power series is desired, its value is FALSE.

ARG - value of variable if BOOL is TRUE; point at which power series expansion is desired if BOOL is FALSE

R - array of values of derivatives if BOOL is TRUE, the Jth entry being the value of the Jth derivative; array of values of the first K+1 coefficients of the power series if BOOL is FALSE

K - highest order of the derivative if BOOL is TRUE; NDEG if BOOL is FALSE

PARAMETER RETURNED. The parameter returned is:

R

METHOD. The Horner scheme is used.

REFERENCE. Pankiewicz, W. 1968. ALGORITHM 337, Calculation of a Polynomial and Its Derivative Values by Horner Scheme. Comm. ACM 11:663.

POLYGCD PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYGCD(N1DEG,COEF1,N2DEG,COEF2,TOL,ERRCOD);
VALUE TOL;
INTEGER N1DEG,N2DEG,ERRCOD;
REAL TOL;
ARRAY COEF1,COEF2[0];
```

PURPOSE. POLYGCD finds the greatest common divisor of two polynomials.

DESCRIPTION OF PARAMETERS. The parameters are:

N1DEG - degree of first polynomial

COEF1 - array of coefficients for first polynomial, ordered from largest to smallest power; dimension [0:N1DEG]

N2DEG - degree of second polynomial

COEF2 - array of coefficients for second polynomial, ordered from largest to smallest power; it is replaced by the greatest common divisor; dimension [0:N2DEG]

TOL - tolerance value below which the coefficient is eliminated during normalization

ERRCOD - resultant error code, where:
0 indicates no error
1 indicates either COEF1 or COEF2 is a zero polynomial

PARAMETERS RETURNED. Parameters returned are:

N2DEG
COEF2
ERRCOD

METHOD. The greatest common divisor of two polynomials, COEF1 and COEF2, is determined by the Euclidean algorithm. Coefficient arrays, COEF1 and COEF2, are destroyed and the greatest common divisor is generated in COEF2.

REMARKS. N1DEG must be greater than N2DEG. N2DEG = 0 on return means COEF1 and COEF2 are prime. The GCD is a constant. N1DEG is destroyed during computation.

OTHER PROCEDURE REQUIRED. The other procedure required is:

POLYDIV

POLYNORM PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYNORM(NDEG,COEF,TOL):
VALUE TOL;
INTEGER NDEG;
REAL TOL;
ARRAY COEF[0];
```

PURPOSE. POLYNORM normalizes the coefficient array of a polynomial.

DESCRIPTION OF PARAMETERS. The parameters are:

NDEG - degree of polynomial; replaced by final degree

COEF - array of original coefficient, ordered from largest to smallest power; replaced by final coefficients; dimension [0:NDEG]

TOL - tolerance below which coefficient is eliminated

PARAMETERS RETURNED. The parameters returned are:

NDEG
COEF

METHOD. The degree of the polynomial, NDEG, is reduced by one for each leading coefficient with an absolute value less than or equal to TOL. Coefficients are then moved to the left so that the leading coefficient is in COEF[0].

REMARKS. If all coefficients including the constant term are less than TOL, the result is a zero polynomial with NDEG = -1. If only the constant term is greater than TOL, the resulting value of NDEG = 0.

POLYMOV PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYMOV(N1DEG,COEF1,N2DEG,COEF2);
VALUE N1DEG;
INTEGER N1DEG,N2DEG;
ARRAY COEF1,COEF2[0];
```

PURPOSE. POLYMOV moves one polynomial to another.

DESCRIPTION OF PARAMETERS. The parameters are:

N1DEG - degree of polynomial to be moved

COEF1 - array of coefficients of polynomial to be moved, ordered from largest to smallest power; dimension [0:N1DEG]

N2DEG - degree of resultant polynomial

COEF2 - array of coefficients of resultant polynomial, ordered from largest to smallest power; dimension [0:N2DEG]

PARAMETERS RETURNED. The parameters returned are:

N2DEG
COEF2

METHOD. N2DEG is replaced by N1DEG, and array COEF1 is moved to COEF2.

POLYSOR PROCEDURE.
The syntax is as follows:

```
PROCEDURE POLYSOR(NDEG,COEF,U);
VALUE NDEG,U;
INTEGER NDEG;
REAL U;
ARRAY COEF[0];
```

PURPOSE. POLYSOR provides a shift of origin.

DESCRIPTION OF PARAMETERS. The parameters are:

NDEG - degree of polynomial

COEF - array of coefficients ordered from largest to smallest power; replaced by array of transformed coefficients

U - shift parameter

PARAMETER RETURNED. The parameter returned is:

COEF

METHOD. Coefficient array COEF[I] of P(X) is transformed such that Q(X)=P(X-U), where Q(X) denotes the polynomial with the transformed coefficient array.

SECTION 3

ORDINARY DIFFERENTIAL EQUATIONS

DIFEQNS PROCEDURE.
The syntax is as follows:

```
PROCEDURE DIFEQNS(SP,XFX,N,XI,XF,Y,HI,EPS,ERR,ERRLBL);
VALUE SP,N,XI,XF,HI,EPS;
PROCEDURE XFX;
INTEGER N;
REAL SP,XI,XF,HI,EPS,ERR;
ARRAY Y[0];
LABEL ERRLBL;
```

PURPOSE.
DIFEQNS solves a system of first order ordinary differential equations and prints the solution at increments on the system line printer.

INPUT.
Input to DIFEQNS consists of:

SP - a positive value for SP causes output at increments of SP in addition to program controlled output; if SP = 0, only program-controlled output occurs

XFX - procedure to generate the right-hand side of the differential equation; it must have a heading of the form:

```
PROCEDURE NAME(X,Y,DY);
VALUE X; REAL X;
ARRAY Y,DY[0];
```

where given the value of the independent variable X and the values of the dependent variables at X, stored in Y[1:N], NAME returns the values of the derivatives at X in DY[1:N]

N - number of differential equations in the system; N should be less than or equal to 10

XI - initial value of the independent variable

XF - final value of the independent variable; XF may be less than XI in which case HI must be negative

Y - array containing the initial values of the dependent variables, which must be stored in Y[1] through Y[N]; these are not preserved; Y must be dimensioned [0:N]

HI - initial step-size; HI may be negative and must satisfy ABS(XF-XI)/(2*15)<ABS(H)

EPS - error tolerance (1.0@-10 LEQ EPS LEQ 1.0@-2)

ERR - parameter to specify which convergence criterion to use:

SE - standard error
RE - relative error
AE - absolute error

ERRLBL - a label in the calling program to which a return is made in case of an input error or in case the iteration fails to converge

OUTPUT.
Output for DIFEQNS consists of:

Y - contains the solution values at the final solution point

METHOD.
DIFEQNS uses the rational extrapolation method of R. Bulirsch and J. Stoer which has been found to be superior to all other methods both in speed and accuracy for small systems of differential equations, say N less than five.

REFERENCE.
DIFEQNS is an adaptation of the FORTRAN package DESUB which is Vol. 2, Issue 1, of the Numerical Mathematics Program Library Project, Bell Telephone Laboratories, Murray Hill, New Jersey.

EXAMPLES.
The following sample program could be used to solve the differential equation Y"+Y=0 with the initial conditions, Y(0)=1, Y'(0)=0, over the range X=0 to X=10 with output at program selected points.

```
BEGIN
ARRAY YSTART[0:2];
LABEL FAIL;
PROCEDURE FUNCT(X,Z,DZ);
VALUE X; REAL X;
ARRAY Z,DZ[0];
    BEGIN
        DZ[1] := Z[2]; DZ[2] := -Z[1];
    END FUNCT;
            YSTART[1] := 1; YSTART[2] := 0;
            DIFEQNS(0,FUNCT,2,0,10,YSTART,.1,@-9,"RE",FAIL);
FAIL: END.
```

In order to solve the same system with output at increments of .1 (in addition to output at program selected points) and with an error of 10-9, the procedure call

```
DIFEQNS(.1,FUNCT,2,0,10,YSTART,.1,@-9,"RE",FAIL);
```

could be used instead. No relationship between SP and HI is necessary

DISCUSSION.

INITIAL STEP-SIZE - HI. The program may recover from an initial step-size which is much too large or much too small because it continuously adjusts the step-size in search of an optimum value, but a bad choice loses time. A frequently reasonable choice for HI is between ABS(XF-XI)/100 and ABS(XF-XI)/10. If the equations have right-hand sides which are changing rapidly, a smaller HI must be chosen. In this case, an order-of-magnitude estimate is HI = minimum over I,J of ABS(1/(DFI/DYJ)). The minimum step-size is set at HI/HDIV, where HDIV = 1024.

ERROR TOLERANCE - EPS. Usually EPS should be chosen in the interval @-9 $\leq$ EPS $\leq$ @-4. However, a value in the interval @-10 $\leq$ EPS $\leq$ @-2 is accepted by the program. Very small values of EPS cause round-off errors to accumulate as well as increase the running time.

CONVERGE CRITERION - SE, RE, or AE. The convergence test chosen is applied successively to the latest two extrapolated values of each component, Y[I], I = 1,2,...,N, of the dependent variable vector Y. Assuming the difference between these for the Jth component is R[J], the three convergence tests are as follows:

a. Standard Error - SE.
   ABS (R[J]) < EPSxM[J] J = 1,2,...,N
   where M[J] is the largest absolute value attained by Y[J] thus far. If M[J] < EPS, it is replaced by EPS.

b. Relative Error - RE.
   ABS(R[J]) < EPSxABS(Y[J]) J = 1,2,...,N
   where Y[J] is the current approximation to the Jth component. If ABS(Y[J]) < EPS, it is replaced by EPS.

c. Absolute Error - AE.
   ABS(R[J]) < EPS J = 1,2,...,N

ERROR CONDITIONS AND THE RETURN LABEL - ERRLBL. The input parameters are checked to see if they satisfy the conditions set forth under the description of parameters. If a parameter is incorrectly specified, an appropriate error message is printed and a return is made to the statement in the calling program labeled ERRLBL (i.e., to the statement whose label is passed as the actual parameter corresponding to the format parameter ERRLBL). These same two actions occur if the extrapolation fails to converge at some step.

OUTPUT PROCEDURE AND STOPPING CONDITION. A separate procedure, XOUTX, to print the solution is declared and called inside DIFEQNS. XOUTX is the best place to insert a stopping condition if a particular problem calls for one. (A stopping condition is a decision to terminate the procedure as soon as the solution satisfies a prescribed condition, e.g., that each component of the solution be less than @-5 in magnitude. In standard usage the procedure is terminated when the independent variable reaches a prescribed value of XF.) If a stopping condition is inserted, it should be followed by the statement

```
GO TO TMNLBL;
```

which results in an exit via the end of DIFEQNS. Other exits are possible if the label L4 (line 47000) and the label EXIT (line 48300) in the calls to XOUTX are replaced by a global label in the calling program.

REDUCTION OF HIGHER ORDER SYSTEMS OF EQUATIONS TO FIRST ORDER. To be acceptable to DIFEQNS the equations must be first order. However, any system of higher order equations may be reduced to a larger system of first order equations. Following is an example involving a single, second order equation.

```
GIVEN D2U/DX2 = F(X,U,DU/DX)
SET Y1 = U, Y2 = DY1/DX to get the system
DY1/DX = Y2
DY2/DX = F(X,Y1,DY1/DX)
```

PROGRAM RUNNING TIME. As an indication only, the average elapsed processor time for the two calls given under the EXAMPLES paragraph is 1.49 seconds and 9.13 seconds, respectively. It must be noted that this system is small and its solution is very well behaved.

SIMPLEDIFEQNS PROCEDURE.
The syntax is as follows:

```
PROCEDURE SIMPLEDIFEQNS(XFX,N,XI,XF,YSTART,HI,EPS,ERR);
VALUE N,XI,XF,HI,EPS,ERR;
PROCEDURE XFX;
INTEGER N;
REAL XI,XF,HI,EPS,ERR;
ARRAY YSTART[0];
```

PURPOSE.
SIMPLEDIFEQNS solves a system of first order ordinary differential equations and prints the solution at increments on the system line printer.

INPUT.
Input to SIMPLEDIFEQNS consists of:

XFX - procedure to generate the right-hand side of the differential equation; it must have a heading of the form:

```
PROCEDURE NAME(X,Y,DY);
VALUE X; REAL X;
ARRAY Y,DY[0];
```

N - the number of differential equations in the system

XI - initial value of the independent variable

XF - final value of the independent variable; XF may be less than XI in which case HI must be negative

YSTART - array containing the initial values which must be stored in YSTART[1:N]

HI - initial step-size; HI may be negative

EPS - error tolerance (1.0@-10≤EPS≤1.0@-2)

ERR - convergence criterion parameter

SE - standard error
RE - relative error
AE - absolute error

OUTPUT.
Output for SIMPLEDIFEQNS consists of:

YSTART - contains the most recently calculated solution values

METHOD.
SIMPLEDIFEQNS uses rational extrapolation.

‹EFERENCE.
SIMPLEDIFEQNS is an adaption of the FORTRAN package DESUB which is Vol. 2, Issue 1, of the Numerical Mathematics Program Library Project, Bell Telephone Laboratories, Murray Hill, New Jersey.

EXAMPLES.
The call SIMPLEDIFEQNS(XFX,2,0,10,YSTART,.1,@-9,"RE") could be inserted into the sample program given under DIFEQNS. A somewhat different output format with the same solution would result. The label FAIL would be superfluous.

DISCUSSION.
SIMPLEDIFEQNS is a simplified version of DIFEQNS. It employs an identical numerical algorithm but omits input parameter checking and the specified points logic, reducing program size by almost one-half but only marginally increasing execution speed. The omission of checking is a serious disadvantage: Should an input parameter lie out of range, a program failure may occur without a message to the user; or if convergence is not achieved, a standard return rather than a branch is made (however, in this case an error message is printed on the output printer). Because of this and the availability of binding on the system, DIFEQNS is the clear choice over SIMPLEDIFEQNS except in those applications where object program size is a critical factor.

Besides the above, several minor differences exist between the two programs. DIFEQNS uses the specified point capability to terminate the solution exactly at XF, whereas SIMPLEDIFEQNS uses a procedure ENDTEST to stop at the first solution point past XF. ENDTEST is declared near the end of SIMPLEDIFEQNS and may easily be reprogramed by the user to provide a different stopping condition.

SECTION 4

NUMERICAL INTEGRATION

SINGLE INTEGRATION.

ROMBERGINT PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE ROMBERGINT(F,A,B,EPS);
VALUE A,B,EPS; REAL A,B,EPS;
REAL PROCEDURE F;
```

PURPOSE. ROMBERGINT integrates a function of one variable over a finite interval.

INPUT. Input to ROMBERGINT consists of:

F - the function to be integrated must be supplied as a typed ALGOL procedure with the heading:

```
REAL PROCEDURE NAME(X); VALUE X; REAL X;
```

A - lower limit of integration

B - upper limit of integration

EPS - relative error tolerance

USAGE. ROMBERGINT is for use with moderately smooth integrands and medium accuracy, say EPS = 1.0@-6.

METHOD. The basic idea is to combine two successive estimates obtained from the trapezoid rule to get a more accurate estimate. Let T(K,0) denote the trapezoid sums using 2**K subdivisions of the interval of integration. Then T(K,1) = (4xT(K+1,0)-T(K,0))/(4-1) is, generally speaking, a more accurate approximation to the integral. Higher order approximations, T(K,M), are computed recursively from the formula T(K,M) = (4**MxT(K+1,M-1)-T(K,M-1))/(4**M-1). The "diagonal" sequence T(0,M), M=0,1,...,K, is assumed to converge most rapidly to the integral. The procedure is exited when ABS(T(0,K)-T(0,K-1))≤EPS x the trapezoid approximation (with 2**K subdivisions) to the integral of ABS(F(X)) or when K reaches 15. The sequence of operations used to compute T(K,M) is that given by Rutishauser (below).

REFERENCES. Bauer, F. L.; Rutishauser, H.; and Stiefel, E. 1963. New Aspects in Numerical Quadrature. In Experimental Arithmetic, High Speed Computing and Machinery. Proc. of Symposia in Applied Mathematics 15:199-218. Providence, R.I.:Amer. Math. Society.

Rutishauser, H. 1967. Description of ALGOL 60. In Handbook for Automatic Computation, Vol. 1. ed. C. Reinsch and J. H. Wilkinson, pp. 105-106. New York:Springer-Verlag.

SIMPSONINT PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE SIMPSONINT(F,A,B,EPS);
VALUE A,B,EPS; REAL A,B,EPS;
REAL PROCEDURE F;
```

PURPOSE. SIMPSONINT integrates a function of one variable over a finite interval.

INPUT. Input to SIMPSONINT consists of:

F - the function to be integrated must be supplied as a typed ALGOL procedure with the heading:

```
REAL PROCEDURE NAME(X); VALUE X; REAL X;
```

A - lower limit of integration

B - upper limit of integration

EPS - relative error tolerance

USAGE. SIMPSONINT is adaptive and is recommended for moderate accuracies, e.g., EPS $\geq$ 1.0@-6, with integrands which are complicated or integrands whose derivative may be infinite or large.

METHOD. The procedure is based on the adaptive Simpson method of McKeeman, but employs interval bisection in place of trisection. The working procedure, RECURSIMPS, is given a subinterval S, an estimate for the integral over this subinterval computed using Simpson's rule for two subdivisions, an estimate for the integral, ABSAREA, of ABS(F(X)) over the whole interval based on all the subintervals created up to this point, and an error tolerance adjusted for the length of the current subinterval. It then computes a new estimate for S using Simpson's rule for four subdivisions, revises ABSAREA to account for the new subdivision, reduces EPS by EPSFACTOR (currently set at .75 which is conservative), and performs the test: ABS (old estimate - new estimate)$\leq$EPSxABSAREA. If the test is passed, the next subinterval to the right is evaluated. If not, S is divided in half and RECURSIMPS is called recursively to evaluate each half, unless the level exceeds 24 when convergence is assigned regardless of whether it occurs. The spacing at level 24 is approximately (B-A)x6.2@-7.

REFERENCES. McKeeman, W. M. 1962. Algorithm 145, Adaptive Numerical Integration by Simpson's Rule. Comm. ACM 5:604.

Lyness, J. N. 1969. Notes on the Adoptive Quadrature Routine. JACM 16:483-495.

BULIRSCHINT PROCEDURE.
The syntax is as follows:

```
PROCEDURE BULIRSCHINT(F,A,B,EPS,INT,ERR);
VALUE A,B,EPS; REAL A,B,EPS,INT,ERR;
REAL PROCEDURE F;
```

PURPOSE. BULIRSCHINT integrates a function of one variable over a finite interval.

INPUT. Input to BULIRSCHINT consists of:

F - the function to be integrated must be supplied as a typed ALGOL procedure with the heading:

```
REAL PROCEDURE NAME(X); VALUE X; REAL X;
```

A - lower limit of integration

B - upper limit of integration

EPS - relative error tolerance

OUTPUT. Output for BULIRSCHINT consists of:

INT - value of integral
ERR - an upper bound for ABS (INT - TRUE VALUE OF INTEGRAL)

USAGE. BULIRSCHINT is recommended for high accuracies (EPS≥1.0@-9) and very smooth integrands. It is not to be used to integrate polynomials of a degree less than four or periodic functions over a periodicity interval.

METHOD. This is a variation of Romberg integration, recently proposed by Bulirsch and Stoer, called rational extrapolation. Let T(J) denote the trapezoid rule approximation to the integral, INT, for successively smaller step-sizes H(J). The basic idea is to fit a rational function of H**2 to a sequence of the trapezoid sums. The value of the rational function at H=0 (the rational extrapolation) is generally a more accurate approximation to INT than any of the trapezoid sums. The extrapolated values are calculated for successive sequences of trapezoid sums and tested to see if they are approaching a limit. Let T(I,K) be the extrapolated value of the rational function which is fitted to the values T(J), J=I,I+1,...,I+K. The T(I,K) is calculated recursively in the program from T(J) with step-sizes H(J) = L/2,L/3,L/4,L/6,L/8,L12,..., where L=B-A, in the order T(1,1),T(2,1),T(1,2),T(3,1),T(2,2),T(1,3), T(4,1),T(3,2),..., subject to the limitation K≤6, I+K≤15. These values fill out a trapezoidal tableau whose right-hand border, T(1,1),T(1,2), ...,T(1,6),T(2,6),T(3,6),...,T(9,6), is assumed to converge to INT (this can be proved if the integrand is smooth and no round-off error is committed). Only the current extrapolated value (denoted by T in the program) and the preceding one (denoted by TA) are used in the convergence test so that none of the interior values of the tableau need to be explicitly calculated or saved.

A complicated 2-fold convergence criterion is used. The closeness of two successive extrapolated values is tested by ERR=ABS(T-TA) < EPSxWxTAB. EPS is the input tolerance and TAB is the current trapezoid rule approximation to the integral of ABS(F(X)). The factor W is determined by the program in such a way that the actual error is generally the same order of magnitude as EPSxTAB. This means that the returned parameter ERR is conservative, frequently overstating the actual error by the order of magnitude of W, where $4 \leq W \leq 144$. The second part of the test is used to prevent premature exiting from the procedure in cases where the text on T-TA is passed early because of an unusual combination of circumstances. This test does not allow an exit in the early stages unless certain combinations of the T(I,J) have become monotonic (their limiting behavior).

REFERENCE. Bulirsch, R. and Stoer, J. 1967. Numerical Quadrature by Extrapolation. Numerische Mathematik 9:271-278.

MULTIPLE INTEGRATION.

SIMPS PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE SIMPS(X,X1,X2,EPS,F);
VALUE X1,X2,EPS; REAL X,X1,X2,EPS,F;
```

PURPOSE. SIMPS evaluates numerically a 2- or 3-dimensional iterated integral.

INPUT. Input to SIMPS consists of:

X - variable of integration

X1 - lower limit of integration

X2 - upper limit of integration

EPS - relative error tolerance

F - function to be integrated; it can be declared as a real procedure or passed as a real expression

USAGE. SIMPS is specifically designed for multidimensional integrals with variable limits of integration. The integration should be smooth. EPS should be greater than or equal to 1.0@-6.

EXAMPLES. SIMPS(X,0,1,@-5,SIMPS(Y,0,SQRT(1-X**2),.5@-5,X**2+Y**2)) computes the integral of X**2+Y**2 over the first quadrant of the unit circle. An example of a 3-dimensional integral is SIMPS(X,-1,2,@-4, SIMPS(Y,0,X,.5@-4,SIMPS(Z,0,X**2+Y**2,.25@-4,G(X,Y,Z)))), where G is declared as a real procedure of the three variables X,Y,Z.

METHOD. A single call to SIMPS computes an estimate of the integral of F(X) between the limits X1 and X2 using a series of approximations generated by the Simpson rule. Let S(K) be the Simpson rule approximation using 2**K subintervals, S(K) = Hx(F(A)+4xF(A+H)+2xF(A+2xH)+ (B-A)/2**K. The convergence test ABS(S(K)-S(K-1))$\leq$ EPSxW is applied for K $\geq$ 3 so that at least eight points are used in the final estimate. If this test is not passed, S(15) is returned as the answer. The procedure is written in such a way that it may call itself. Also, the limits X1 and X2 may be expressions. Therefore, SIMPS is specifically designed to evaluate integrated integrals.

REFERENCE. Olynyk, Frank. 1964. ALGORITHM 233, Simpson's Rule for Multiple Integration. Comm. ACM 7:348.

DOUBLEINTEGRAL PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE DOUBLEINTEGRAL(F,A1,B1,A2,B2,EPS);
VALUE A1,B1,A2,B2,EPS; REAL A1,B1,A2,B2,EPS;
```

PURPOSE. DOUBLEINTEGRAL integrates numerically a function of two variables over a rectangle.

INPUT. Input to DOUBLEINTEGRAL consists of:

F - function to be integrated; it must have a heading of the form:

```
REAL PROCEDURE F(X,Y);
VALUE X,Y; REAL X,Y;
```

A1 - lower limit of integration of X axis

B1 - upper limit of integration of X axis

A2 - lower limit of integration of Y axis

B2 - upper limit of integration of Y axis

EPS - relative error tolerance

USAGE. DOUBLEINTEGRAL may only be used for double integrals with constant limits. The procedure is adaptive to handle general integrands. EPS should be greater than or equal to 1.0@-6.

EXAMPLE. DOUBLEINTEGRAL(F,0,1,0,2,@-5) evaluates the integral of a function F of two variables over the rectangle with vertices at (0,0), (0,2), (1,2), (0,2).

METHOD. The double integral is treated as a 1-dimensional integral of a function which is itself an integral. The adaptive procedure SIMPSONINT is used for the 1-dimensional integration.

REFERENCE. This is a modification of: McKeeman, W. M. 1962. ALGORITHM 146, Multiple Integration. Comm. ACM 5:604.

TRIPLEINTEGRAL PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE TRIPLEINTEGRAL(F,A1,B1,A2,B2,A3,B3,EPS);
VALUE A1,B1,A2,B2,A3,B3,EPS; REAL A1,B1,A2,B2,A3,B3,EPS;
REAL PROCEDURE F;
```

PURPOSE. TRIPLEINTEGRAL integrates numerically a function of three variables over a 3-dimensional rectangle.

INPUT. Input to TRIPLEINTEGRAL consists of:

F - function to be integrated; it must have a heading of the form:

```
REAL PROCEDURE F(X,Y,Z);
VALUE X,Y,Z; REAL X,Y,Z;
```

A1 - lower limit of integration on X axis

B1 - upper limit of integration on X axis

A2 - lower limit of integration on Y axis

B2 - upper limit of integration on Y axis

A3 - lower limit of integration on Z axis

B3 - upper limit of integration on Z axis

EPS - relative error tolerance

USAGE. TRIPLEINTEGRAL may only be used for triple integrals with constant limits. The procedure is adaptive to handle general integrands. EPS should be greater than or equal to 1.0@-6.

EXAMPLE. TRIPLEINTEGRAL(F,0,1,-1,1,-2,0,@-4) integrates the function F of three variables over the hyperrectangle with vertices at (0,1,0), (1,1,0),(1,-1,0),(0,-1,0),(0,1,-2),(1,1,-2),(1,-1,-2),(0,-1,-2).

METHOD. TRIPLEINTEGRAL is an extension of the method used for DOUBLEINTEGRAL.

REFERENCE. This is a modification of: McKeeman, W. M. 1962. ALGORITHM 146, Multiple Integration. Comm. ACM 5:604.

SECTION 5

SPECIAL FUNCTIONS

JBESS PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE JBESS(NMAX,X,J);
VALUE NMAX,X; INTEGER NMAX; REAL X;
ARRAY J[0];
```

PURPOSE.
JBESS evaluates the Bessel functions J0,...,JNMAX at the fixed value X.

INPUT.
Input to JBESS consists of:

NMAX - an integer which can be positive, negative, or zero
X - an unrestricted real value

OUTPUT.
Output for JBESS consists of:

J - an array containing the values J0,...,JNMAX
dimensioned [0:ABS(NMAX)]

JBESS - returned the value J(NMAX,X)

METHOD.
This procedure is a special case of one given by Gautschi. An iterative backward recurrence algorithm is used. The iteration is continued until approximately D significant digits of accuracy are obtained. D is set at 11. The algorithm is most efficient when X is small or moderately large in absolute value. If NMAX or X is negative, the relations J(-K,X) = (-1)**K*J(K,X) and J(K,-X) = (-1)**K*J(K,X) are used.

REFERENCE.
Gautschi, Walter. 1964. ALGORITHM 236, Bessel Functions of the First Kind. Comm. ACM 7:479-480.

IBESS PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE IBESS(NMAX,X,I);
VALUE NMAX,X; INTEGER NMAX; REAL X;
ARRAY I[0];
```

PURPOSE.
IBESS evaluates the modified Bessel functions I0,...,INMAX at the fixed value of X.

INPUT.
Input to IBESS consists of:

NMAX - an integer which can be positive, negative, or zero
X - an unrestricted real value

OUTPUT.
Output for IBESS consists of:

I - an array containing the value I0,...,INMAX
dimensioned [0:ABS(NMAX)]

IBESS - returns the value I(NMAX,X)

METHOD.
This procedure is a special case of one given by Gautschi. An iterative backward recurrence algorithm is used. The iteration is continued until approximately D significant digits of accuracy are obtained. D is set at 11. The algorithm is most efficient when X is small or moderately large in absolute value. If NMAX or X is negative, the relations I(K,X) = I(-K,X) and I(K,-X) = (-1)**K*I(K,X) are used.

REFERENCE.
Gautschi, Walter. 1964. ALGORITHM 236, Bessel Functions of the First Kind. Comm. ACM 7:479-480.

CXJBES PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE CXJBES(NMAX,X,Y,U,V);
VALUE NMAX,X,Y; INTEGER NMAX; REAL X,Y;
ARRAY U,V[0];
```

PURPOSE.
JBSSCX evaluates the Bessel Functions J0,...,JNMAX at the fixed, complex value Z = X + I * Y.

INPUT.
Input to JBSSCX consists of:

NMAX - an integer which can be positive, negative, or zero
X - the unrestricted real part of the argument
Y - the unrestricted imaginary part of the argument

OUTPUT.
Output for JBSSCX consists of:

U - an array containing the real parts of the values J0,...,JNMAX dimensioned [0:ABS(NMAX)]

V - an array containing the imaginary parts of the values J0,...,JNMAX dimensioned [0:ABS(NMAX)]

METHOD.
This procedure is a special case of one given by Gautschi. An iterative backward recurrence algorithm is used. The iteration is continued until approximately D significant digits of accuracy are obtained. D is set at 11. The algorithm is most efficient when modulus(z) is small or moderately large. If NMAX is negative the formula J(-K,Z) = (-1)**K*J(K,Z) is used.

REFERENCE.
Gautschi, Walter. 1964. ALGORITHM 236, Bessel Functions of the First Kind. Comm. ACM 7:479-480.

EL1 PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE EL1(X,KC);
VALUE X,KC; REAL X,KC;
```

PURPOSE.
EL1 evaluates the incomplete elliptic integral of the first kind defined here as the integral from 0 to X of the function of T:

```
DT/((1+T**2)*(1+KC**2*T**2)).
```

The parameters X and KC may be any real numbers. An alternate definition is EL1(X,KC) = Integral from 0 to ARCTAN (X) of:

```
DPHI/SQRT(1-K**2*SIN(PHI)**2), where K**2 + KC**2 = 1.
```

METHOD.
The algorithm is based on the Landen transformation. Use of the constants CA = 10**(-D/2) and CB = 10 **(-D-2) yields amount D significant figures. D is set to 11. For KC = 0, EL1 reduces to ARSINH(X) which is evaluated in double precision from its logarithmic representation.

REFERENCE.
Bulirsch, R. 1965. Numerical Calculations of Elliptic Integrals and Elliptic Functions. Numerische Mathematik 7:78-90.

CEL1 PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE CEL1 (KC);
VALUE KC; REAL KC;
```

PURPOSE.
CEL1 evaluates the complete elliptic integral of the first kind defined here as the integral from 0 to infinity of the function of T:

```
DT/((1+T**2)*(1+KC**2*T**2)).
```

The parameter KC may be any real number. An alternate definition is CEL1(X,KC) = Integral from 0 to PI/2 of:

```
DPHI/SQRT(1-K**2*SIN(PHI)**2),  where K**2 + KC**2 = 1.
```

METHOD.
The algorithm is based on the Landen transformation. Use of the constant CA = 10**(-D/2) yields about D significant figures. D is set to 11. For KC = 0 the integral is not finite and an exponent overflow is caused.

REFERENCE.
Bulirsch, R. 1965. Numerical Calculations of Elliptic Integrals and Elliptic Functions. Numerische Mathematik 7:78-90.

EL2 PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE EL2(X,KC,A,B);
VALUE X,KC,A,B; REAL X,KC,A,B;
```

PURPOSE.
EL2 evaluates the general elliptic integral of the second kind defined as the integral from 0 to X of the function of T:

```
DT*(A+B*T**2)/((1+T**2)*SQRT((1+T**2)*(1+KC**2*T**2))).
```

The parameters X,KC,A, and B may be any real numbers. An alternate definition is EL2(X,KC,A,B) = Integral from 0 to ARCTAN(X) of:

```
DPHI*(A+(B-A)*SIN(PHI)**2/(SQRT(1-KC**2*SIN(PHI)**2)),
```

where KC**2 + K**2 = 1.

METHOD.
The algorithm is based on the Landen transformation. Use of the constants CA = 10**(-D/2) and CB = 10**(-D-2) yields when A and B have the same sign about D valid figures, which are significant if ABS(A) GTR ABS(B). D is set to 11. The C*Z part of the value EL2 gives the value of the Jacobian Zeta function for Z = K**2. For KC=0 EL2 is represented by double precision elementary functions.

REFERENCE.
Bulirsch, R. 1965. Numerical Calculations of Elliptic Integrals and Elliptic Functions. Numerische Mathematik 7:78-90.

CEL2 PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE CEL2(KC,A,B);
VALUE KC,A,B; REAL KC,A,B;
```

PURPOSE.
CEL2 evaluates the general complete elliptic integral of the second kind defined as the integral from 0 to infinity of:

```
DT*(A+B*T**2)/((1+T**2)*SQRT((1+T**2)*(1+KC**2*T**2))).
```

The parameters X,KC,A, and B may be any real numbers. An alternative definition is EL2(X,KC,A,B) = Integral from 0 to ARCTAN(X) of:

```
DPHI*(A+(B-A)*SIN(PHI)**2/(SQRT(1-KC**2*SIN(PHI)**2)),
```

where KC**2 + K**2 = 1.

METHOD.
The algorithm is based on the Landen transformation. Use of the constants PI/4 and CA = 10**(-D/2) yields about D significant figures when A and B have the same sign. D is set to 11. For KC=0 and B=0 the integral reduces to the constant A. For KC=0 and B NEQ 0 the integral is not finite. In this case an exponent overflow is caused.

REFERENCE.
Bulirsch, R. 1965. Numerical Calculations of Elliptic Integrals and Elliptic Functions. Numerische Mathematik 7:78-90.

EL3 PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE EL3(X,KC,P);
VALUE X,KC,P; REAL X,KC,P;
```

PURPOSE.
EL3 evaluates the incomplete elliptic integral of the third kind which is defined as the integral from 0 to X of the function of T:

```
(1+T**2)/((1+P*T**2)*SQRT((1+T**2)*(1+KC**2*T**2)) * DT.
```

The parameters X,P and KC are permitted to be any real numbers with the restriction that 1+P*X**2 NEQ 0. Standard notation for EL3 is PI(PHI,N,K) = Integral from 0 to PHI of:

```
DPHI/((1+N*SIN(PHI)**2)*SQRT(1-K**2*SIN(PHI)**2)).
```

The relation PI(PHI,N,K) = EL3(X,KC,P) holds, where PHI = ARCTAN(X), K**2+KC**2 = 1, and P = N+1.

METHOD.
The algorithm is based on the Bartky transformation. Use of constants CA = 10**(-D/2), CB = 10**(-D-2) and ND = D-2 yields D significant figures. One digit may be lost by cancellation. D is set to 11. An exponent overflow is caused if 1+P*X**2=0.

REFERENCE.
Bulirsch, R. 1965. Numerical Calculations of Elliptic Integrals and Elliptic Functions. Numerische Mathematik 7:78-90.

CEL3 PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE CEL3(KC,P);
VALUE KC,P; REAL KC,P;
```

PURPOSE.
CEL3 evaluates the complete elliptic integral of the third kind defined as the integral from 0 to infinity of the function of T:

```
DT*(1+T**2)/((1+P*T**2)*SQRT((1+T**2)*(1+KC**2*T**2))).
```

The trigonometric form of CEL3 is the integral from 0 to PI/2 of:

```
DPHI/((1+(P-1)*SIN(PHI)**2*SQRT(1-K**2*SIN(PHI)**2)).
```

METHOD.
The algorithm is based on the Landen transformation. Use of the constants PI/2 and CA = 10**(-D/2) yields about D significant figures. D is set to 11. No restrictions are placed on the input parameters KC and P. However, if either KC or P = 0 the integral is not finite and an exponent overflow is caused.

REFERENCE.
Bulirsch, R. 1965. Numerical Calculations of Elliptic Integrals and Elliptic Functions. Numerische Mathematik 7:78-90.

CEL_PROCEDURE.
The syntax is as follows:

```
REAL PROCEDURE CEL(KC,P,A,B);
VALUE KC,P,A,B; REAL KC,P,A,B;
```

PURPOSE.
CEL evaluates the general complete elliptic integral defined as the integral from 0 to infinity of the function of T:

```
DT * (A+B*T**2)/((1+P*T**2)*SQRT((1+T**2)*(1+KC**2*T**2))).
```

There are no restrictions on the parameters KC,P,A, and B. If KC=0 and B NEQ 0 the integral is not finite and an exponent overflow is caused.

The trigonometric form of CEL is the integral from 0 to PI/2 of:

```
DPHI*(A*C**2 + B*S**2)/((C**2 + P*S**2)*SQRT(C**2 + KC**2*S**2))
```

where C = COS(PHI) and S = SIN(PHI).

METHOD.
CEL is evaluated by an algorithm given by Bartky. The use of the constant CA = 10**(-D/2) yields D significant figures. D is set to 11. If KC=0 and B=0, the approximation CEL(0,P,A,0) =/ CEL(@-11,P,A,0) (with a relative error of the order of @-11) is used.

REFERENCE.
Bulirsch, R. 1965. Numerical Calculations of Elliptic Integrals and Elliptic Functions. Numerische Mathematik 7:78-90.

SECTION 6

MISCELLANEOUS

PLOTTER PROCEDURE.

The syntax is as follows:

```
PROCEDURE PLOTTER(OUTFILE,FIRSTPT,NPOINTS,NGRAPHS,X,Y);
VALUE FIRSTPT,NPOINTS,NGRAPHS; INTEGER FIRSTPT,NPOINTS,NGRAPHS;
ARRAY X[0], Y[0,0];
FILE OUTFILE;
```

PURPOSE.

PLOTTER plots up to four graphs on the file OUTFILE. The abscissae (X values) must be equally spaced and may be increasing or decreasing. The abscissae may be calendar data: days or months abbreviated to the first three characters, e.g., MON,WED,FRI, or AUG,SEP,OCT.

INPUT.

Input to PLOTTER consists of:

FIRSTPT - index of first X value to be plotted

NPOINTS - index of last X value to be plotted

NGRAPHS - number of graphs to be plotted

X - X[0:NPOINTS] contains the abscissae to be plotted; they must be equally spaced, may be increasing or decreasing, and may be the first three characters of days or of months (but not mixed)

Y - Y[0:NPOINTS,0:NGRAPHS-1] contains the ordinates to be plotted; the ordinates associated with the abscissae X[K] are Y[K,0],Y[K,1],...,Y[K,NGRAPHS-1]

OUTPUT.

Output for PLOTTER consists of:

OUTFILE - the defines specifying the size of the graph have been preset for a 120-column, 6-lines-per-inch printer

METHOD.

The X values are plotted down the page, and the Y values are plotted across the page. The largest graph is scaled relative to the number of print positions available and the remaining graphs (if any) relative to it. In other words, the same scale factor is applied to all of the graphs. The X values are scaled if it is necessary to remove leading or trailing zeroes, and the formats under which abscissae and ordinates are printed are determined by the size and precision of the data.

A number of defines are set in the beginning of the procedure to determine various characteristics of the graph. XPRINTINCREMENT controls the printing frequency of the abscissae; a value of K causes every Kth abscissae to be printed. Flattening of the graphs occur if PRINT-

COLUMNS is set to less than 120, its default value. Indenting the left margin can be accomplished by giving a nonzero value to INDENT.

A maximum of four graphs is allowed. This limit can be increased simply by assigning more symbols to the array GRAPHSYMBOLS near the end of the procedure. If two graphs cross at a point, only the symbol corresponding to the higher numbered graph is printed.

REFERENCE.

Lloyd, P. 1966. ALGORITHM 278, GRAPH PLOTTER. Comm. ACM 9:88.

READMAIRIX_PROCEDURE.
The syntax is as follows:

```
PROCEDURE READMATRIX(ROWS,COLS,A,READIT,PRINTIT,EXIT);
VALUE ROWS,COLS; INTEGER ROWS,COLS;
ARRAY READIT,PRINTIT;
LABEL EXIT;
```

PURPOSE.
Reads the input matrix A in free format from the file READIT. Blanks may be used as delimiters in place of commas and the FORTRAN exponent symbol E may be used in place of @. The first element of each row must begin a new card (record). The number of elements read per row is counted and compared with N. If agreement is not obtained, an error message is written. If less than N elements are entered, control is transferred to the label exit. The matrix is written on the file PRINTIT.

READSYMMAIRIX_PROCEDURE.
The syntax is as follows:

```
PROCEDURE READSYMMATRIX(N,A,READIT,PRINTIT,EXIT);
VALUE N; INTEGER N;
ARRAY A[0,0];
FILE READIT,PRINTIT;
LABEL EXIT;
```

PURPOSE.
Reads the symmetric input matrix A in free format from the file READIT. Only the lower triangle of the matrix (including the diagonal) is read. Blanks may be used as delimiters in place of commas and the FORTRAN exponent symbol E may be used in place of @. The first element of each row must begin a new card (the file READIT is assumed to consist of 14-word unblocked EBCDIC records). If an end-of-file is encountered before the entire matrix is read, control is transferred to the label exit. The matrix is written on the file PRINTIT.

SECTION 7

FORTRAN COMPATIBLE PROCEDURES

FORTRAN compatible versions of the most important procedures, excluding those requiring 2-dimensional arrays, are included in the library. The selected procedures are modified to allow binding to FORTRAN host programs, and renamed with 6-character names to conform to FORTRAN naming conventions. Thus, no USE card is required with the binder. For easy identification, the modified procedure names begin with F.

It should be noted that most of the arrays used in the ALGOL procedures are declared with lower bounds of zero, but indexed with a lower bound of one (i.e., the zero location is unused). Consequently, FORTRAN input arrays passed to such procedures must be indexed from 2, and output arrays passed back to FORTRAN hosts contain results beginning at location 2.

Also, the roots of a polynomial calculated by FCXPLY or FRLPLY are returned in locations 2 - N+1 of the arrays ZEROR and ZEROI.

FRLPLY_PROCEDURE.
This version of REALPOLYZEROFINDER is suitable for binding to a FORTRAN host.

NOTE

The roots are returned starting in location 2 of the FORTRAN arrays passed for the formal arrays ZEROR and ZEROI.

The syntax for FRLPLY is as follows:

```
PROCEDURE FRLPLY(DEGREE,P,ZEROR,ZEROI);
VALUE DEGREE; INTEGER DEGREE;
ARRAY P,ZEROR,ZEROI[*];
```

PURPOSE.
FRLPLY finds all the zeroes of a real polynomial.

FCXPLY PROCEDURE.
This version of COMPLEXPOLYZEROFINDER is suitable for binding to a FORTRAN host.

NOTE
The roots are returned starting in location 2 of the FORTRAN arrays passed for the formal arrays ZEROR and ZEROI.

The syntax for FCXPLY is as follows:

```
PROCEDURE FCXPLY(DEGREE,PR,PI,ZEROR,ZEROI);
VALUE DEGREE; INTEGER DEGREE;
ARRAY PR,PI,ZEROR,ZEROI[*];
```

PURPOSE.
FCXPLY finds all the zeroes of a complex polynomial.

FRMBRG PROCEDURE.
This version of ROMBERGINT is suitable for binding to a FORTRAN host.

The syntax for FRMBRG is as follows:

```
[PROCEDURE F(X);
VALUE X; REAL X;
EXTERNAL;]

REAL PROCEDURE FRMBRG(A,B,EPS);
VALUE A,B,EPS;
REAL A,B,EPS;
```

PURPOSE.
FRMBRG integrates a function of one variable over a finite interval.

FSMPSN_PROCEDURE.
This version of SIMPSONINT is suitable for binding to a FORTRAN host. The function sub-program named F must appear in the host.

The syntax for FSMPSN is as follows:

```
[PROCEDURE F(X);
VALUE X; REAL X;
EXTERNAL;]

REAL PROCEDURE FSMPSN(A,B,EPS);
VALUE A,B,EPS;
REAL A,B,EPS;
```

PURPOSE.
FSMPSN integrates a function of one variable over a finite interval.

FBULIR_PROCEDURE.
This version of BULIRSCHINT is suitable for binding to a FORTRAN host.
The function sub-program named F must appear in the host.

The syntax for FBULIR is as follows:

```
[PROCEDURE F(X);
VALUE X; REAL X;
EXTERNAL;]

PROCEDURE FBULIR(A,B,EPS,INT,ERR);
VALUE A,B,EPS;
REAL A,B,EPS,INT,ERR;
```

PURPOSE.
FBULIR integrates a function of one variable over a finite interval.

FDBINT_PROCEDURE.
This version of DOUBLEINTEGRAL is suitable for binding to a FORTRAN host. A function of two variables named F must be declared as a function sub-program in the FORTRAN host.

The syntax for FDBINT is as follows:

```
[REAL PROCEDURE F(X,Y);
VALUE X,Y; REAL X,Y;
EXTERNAL;]

REAL PROCEDURE FDBINT(A1,B1,A2,B2,EPS);
VALUE A1,B1,A2,B2,EPS;
REAL A1,B1,A2,B2,EPS;
```

PURPOSE.
FDBINT integrates a function of two variables over a rectangle.

FTRINT PROCEDURE.
This version of TRIPLEINTEGRAL is suitable for binding to a FORTRAN host. A function of three variables named F must be declared as a function sub-program in the FORTRAN host.

The syntax for FTRINT is as follows:

```
[REAL PROCEDURE F(X,Y,Z);
VALUE X,Y,Z; REAL X,Y,Z;
EXTERNAL;]

REAL PROCEDURE FTRINT(A1,B1,A2,B2,A3,B3,EPS);
VALUE A1,B1,A2,B2,A3,B3,EPS;
REAL A1,B1,A2,B2,A3,B3,EPS;
```

PURPOSE.
FTRINT integrates a function of three variables over a 3-dimensional rectangle.

SECTION 8

FORTRAN MATRICES

LINEAR EQUATIONS, MATRIX INVERSION, AND DETERMINANTS.

LINEQN SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE LINEQN(NM,N,A,B,DETERM)
INTEGER NM,N
INTEGER IP(100)
REAL A(NM,N),B(N)
DOUBLE PRECISION DETERM
```

PURPOSE. LINEQN solves the linear system of equations A*X = B and computes a determinant of the coefficient matrix.

INPUT. Input to LINEQN consists of:

NM - row dimension of the 2-dimensional array A as declared in the calling program dimension statement

N - number of equations and unknowns in the system

A - square coefficient matrix

B - right-hand-side vector

OUTPUT. Output from LINEQN includes:

A - input matrix is overwritten by its LU decomposition
B - solution X is overwritten on B
DETERM - double precision determinant of the coefficient matrix

SUBROUTINES REQUIRED. The necessary subroutines are DECOMP and SOLVE.

RESTRICTIONS. The order N of the system must not exceed 100. For larger systems the dimension of the pivot index array IP must be increased.

METHOD. Subroutine DECOMP uses Gaussian elimination with partial pivoting to triangularize A. The upper triangular factor U overwrites the upper triangle (including the diagonal) of A. The negative of the lower triangular factor L (omitting its unit diagonal) is overwritten on the strict lower triangle of A. Actual row interchanges are carried out in the upper triangle. The determinant, product of the diagonal elements of U, is computed in double precision to minimize the possibility of exponent overflow or underflow. If the determinant is 0, an error message is printed and a divide by zero occurs in SOLVE. Otherwise SOLVE overwrites the solution on B by the usual back substitution method.

REFERENCE. Moler, Cleve B. 1972. Matrix Computation with FORTRAN and Paging. Comm. ACM 15:268-270 and 274.

LNQPRV SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE LNQPRV(NM,N,A,LU,B,X,DIGITS,DETERM)
INTEGER I,J,NM,N
INTEGER IP(100)
REAL DIGITS
REAL A(NM,N),LU(NM,N),B(N),X(N)
DOUBLE PRECISION DETERM
```

PURPOSE. LNQPRV solves the linear system A*X = B to machine accuracy and computes the determinant of coefficient matrix A.

INPUT. Input to LNQPRV consists of:

NM - row dimension of the 2-dimensional array A as declared in the calling program dimension statement

N - number of equations and unknowns in the system

A - square coefficient matrix. A is preserved.

B - right-hand-side vector. B is preserved.

OUTPUT. Output from LNQPRV includes:

LU - a 2-dimensional array containing the triangular decomposition of A

X - solution

DIGITS - number of the initially correct digits in solution X

DETERM - double precision determinant of the coefficient matrix

SUBROUTINES REQUIRED. The necessary subroutines are DECOMP, SOLVE, and IMPRUV.

RESTRICTIONS. The order N of the system must not exceed 100. For larger systems the dimension of the pivot index array IP must be increased.

METHOD. Subroutine DECOMP uses Gaussian elimination with partial pivoting to triangularize A. The upper triangular factor U overwrites the upper triangle (including the diagonal) of A. The negative of the lower triangular factor L (omitting its unit diagonal) is overwritten on the strict lower triangle of A. Actual row interchanges are carried out in the upper triangle. The determinant, product of the diagonal elements of U, is computed in double precision to minimize the possibility of exponent overflow or underflow. If the determinant is 0, an error message is printed and a divide by zero occurs in SOLVE. Otherwise SOLVE computes the solution X by the usual back substitution method. Lastly, subroutine IMPRUV refines the solution X by iterative

improvement. DIGITS is based on the solution computed in SOLVE.

REFERENCE. Moler, Cleve B. 1972. Matrix Computation with FORTRAN and Paging. Comm. ACM 15:268-270 and 274.

LNEGNS SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE LNEGNS(NM,N,A,K,C,DETERM)
INTEGER J,K,N,NM
INTEGER IP(100)
REAL A(NM,N),C(NM,K)
DOUBLE PRECISION DETERM
```

PURPOSE. LNEQNS solves the K systems of equations A*Y = C and computes the determirant of coefficient matrix A.

INPUT. Input tc LNEQNS ccnsists cf:

NM - rcw dimensicn of the 2-dimersional arrays A and C as declared in the calling program dimension statement

N - crder cf A

K - number of columns of C (number of right-hand sides)

A - square ccefficient matrix

C - a 2-dimensional array contairing the K right-hand sides as cclumns

OUTPUT. Cutput from LNEQNS includes:

A - input matrix is cverwritten by its LU decomposition
C - the K solutions are overwritten on the columns of C
DETERM - double precision determinant of the coefficient matrix

SUBROUTINES REQUIREC. The necessary subroutines are DECOMP and SCLVE.

RESTRICTICNS. The crder N of the system must not exceed 100. For larger systems the dimension of the pivot index array IP must be increased. Arrays A and C must have the same actual row dimension.

METHOD. Subroutine DECOMP uses Gaussian elimination with partial pivoting to triangularize A. The upper triangular factor U overwrites the upper triangle (including the diagonal) of A. The negative of the lower triangular factor L (omitting its unit diagonal) is overwritten on the strict lower triangle cf A. Actual row interchanges are carried out in the upper triangle. The determinant, product of the diagonal elements of U, is computed in double precision to minimize the possibility of exponent overflow cr underflow. If the determinant is 0, an error message is printed and a divide by zero occurs in SOLVE. Otherwise SCLVE is called successively for each column of C and overwrites the corresponding columns of C with the solutions.

REFERENCE. Moler, Cleve B. 1972. Matrix Computation with FORTRAN and Paging. Comm. ACM 15:268-270 and 274.

SYMBLQ SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE SYMBLQ(NRA,N,M,A,B,X,FIRST,*)
DIMENSION A(NRA,1),B(N),X(N)
LOGICAL FIRST
```

PURPOSE. SYMBLQ solves the symmetric linear system A*X = B where A is a positive-definite band matrix.

INPUT. Input to SYMBLQ consists of:

NRA - row dimension of the 2-dimensional array A as declared in the calling program dimension statement

N - order of system

M - number of lines on either side of the main diagonal; must be a positive integer less than N

A - A[1:N,1:M+1] contains the lower half of the symmetric, positive-definite band coefficient matrix. The subdiagonals of the matrix are stored in the columns of A, and the main diagonal is stored in A(I,M+1), I = 1 (1) N. The elements of the stored array occupy the positions illustrated below for N = 5, M = 2 of a convertional array:

Stored Array			Lower Triangle of Conventional Array				
X	X	A(1,3)	A(1,3)				
X	A(2,2)	A(2,3)	A(2,2)	A(2,3)			
A(3,1)	A(3,2)	A(3,3)	A(3,1)	A(3,2)	A(3,3)		
A(4,1)	A(4,2)	A(4,3)		A(4,1)	A(4,2)	A(4,3)	
A(5,1)	A(5,2)	A(5,3)			A(5,1)	A(5,2)	A(5,3)

The elements indicated by X's in the upper-left corner of the stored array are not used.

B - B[1:N] contains the right-hand side of the system.

FIRST - a TRUE/FALSE switch. The call SYMBLQ(NRA,N,M,A,B,X,.TRUE.,*) is used for the first set of equations. Additional sets with the same coefficient matrix and different right-hand sides are solved by the call SYMBLQ(NRA,N,M,A,B,X,.FALSE.,*).

* - statement label in the calling program to which control is transferred if matrix A is not positive-definite

OUTPUT. Cutput from SYMBLQ includes:

A - lower triangular part of the Cholesky decomposition LU overwrites A.

X - solution vector

METHOD. The Cholesky decomposition A = LU is performed in-place. L is a lower triangular matrix, and U is its transpose. Matrix L is stored in the same form as matrix A. The reciprocals of the main diagonal elements are stored in place of the diagonal elements. If matrix A is not positive-definite, the procedure fails and control is transferred to the statement labeled * in the calling program.

REMARKS. The call SYMBLQ(NRA,N,M,A,B,B,FIRST,*) overwrites B with the solution vector X.

REFERENCES. Martin, R. S. and Wilkinson, J. H. 1965. Symmetric Decomposition of Positive Definite Bard Matrices. Numerische Mathematik 7:355-361.

Reinsch, C. and Wilkinson, J. H. eds. 1971. Linear Algebra. Handbook for Automatic Computation. Vol. 2. New York:Springer-Verlag.

EXAMPLE. The sample calling program given below can be used to solve the sample data that follows.

Sample Calling Program.
A sample calling program follows:

```
      DIMENSION A(10,5),B(10),X(10)
      LOGICAL FIRST
      FIRST=.TRUE.
      NRA=10
   11 FORMAT(2I5)
   13 FORMAT(10F8.0)
      READ(5,11) N,M
      MP1=M+1
      DO   20 I=1,N
   20 READ(5,13) (A(I,J),J=1,MP1)
      READ(5,13) (B(I),I=1,N)
      CALL SYMBLQ(NRA,N,M,A,B,X,FIRST,&101)
      STOP
  101 WRITE(6,103)
  103 FORMAT(//" ** ERROR EXIT **")
      STOP
      END
```

Sample Data.
The followirg is sample data for the preceding program:

```
 0.0     0.0     5.0
 0.0    -4.0     6.0
 1.0    -4.0     6.0
 1.0    -4.0     6.0
 1.0    -4.0     5.0
 2.0    -1.0     0.0      -1.0      2.0
```

Remarks.
The value 0.0 is assigned to the unused, upper-left locations marked X.

SYMLNQ SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE SYMLNQ(NRA,N,A,LDU,B,X,*)
REAL A(NRA,N),LDU(NRA,N),B(N),X(N)
```

PURPOSE. SYMLNQ solves the symmetric linear system A*X = B where matrix A is positive-definite.

INPUT. Input to SYMLNQ consists of:

NRA - row dimension of the 2-dimensional arrays A and LDU as declared in the calling program dimension statement

N - order of system

A - symmetric coefficient matrix. Only the lower triangle of A needs to be stored.

B - right-hand-side vector

* - statement label in the callirg program to which control is transferred if Cholesky decomposition of A does not exist

OUTPUT. Output from SYMLNQ includes:

X - solution vector

LDU - strict lower triangle of LDU contains L-I, and diagonal contains D-inverse.

SUBROUTINES REQUIRED. The necessary subroutines are SYMDCP and SYMSLV.

METHOD. A copy of matrix A (assumed to be positive-definite) is decomposed without pivoting into the triangular form A = LDU. The factors are defined as follows:

L - unit lower triangular matrix
D - diagonal
U - transpose of L

The decomposition is stable for positive-definite matrices. If matrix A is not positive-definite, however, serious loss of accuracy may occur or the decomposition may not exist. If the decomposition does not exist, control is transferred to the statement labeled * in the calling program. The triangular form is then solved in two stages by SYMSLV.

REMARKS. The call SYMLNQ(NRA,N,A,A,B,X,*) overwrites matrix A with its triangular factors L and D. The call SYMLNQ(NRA,N,A,LDU,B,B,*) overwrites vector B with the solution vector X.

REFERENCES. Martin, R. S.; Peters, G.; and Wilkinson, J. H. 1965. Symmetric Decomposition of a Positive Definite Matrix. Numerische Mathematik 7:362-383.

Reinsch, C. and Wilkinson, J. H. eds. 1971. Linear Algebra. Handbook for Automatic Computation. Vol. 2. New York:Springer-Verlag.

SYMLQP SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE SYMLQP(NRA,N,A,LDU,B,X,DIGITS,*,R)
REAL A(NRA,N),LDU(NRA,N),B(N),X(N),R(N)
```

PURPOSE. SYMLQP solves the symmetric linear system A*X = B, where matrix A is positive-definite, and improves the solution X to machine accuracy.

INPUT. Input to SYMLQP consists of:

NRA - row dimension of the 2-dimensional arrays A and LDU as declared in the calling program dimension statement

N - order of system

A - symmetric coefficient matrix. Only the lower triangle of the matrix needs to be stored. If the full matrix is stored, the call to SYMMET is superfluous and should be removed.

B - right-hand-side vector

* - statement label in the callirg program to which control is transferred if Cholesky decomposition of matrix A does not exist or if iterative improvement does not converge

OUTPUT. Output from SYMLQP includes:

X - solution vector

DIGITS - number of correct digits in initial solution

LDU - strict lower triangle contains L-I, and diagonal contains D-inverse.

R - last residual vector computed

SUBROUTINES REQUIRED. The necessary subroutines are SYMDCP, SYMSLV, SYMMET, and SYMIMP.

METHOD. A copy of matrix A (assumed to be positive-definite) is decomposed without pivoting into the triangular form A = LDU. The factors are defined as follows:

L - unit lower triangular matrix
D - diagonal
U - transpose of L

The decomposition is stable for positive-definite matrices. If A is not positive-definite, however, serious loss of accuracy may occur or the decomposition may not exist. The system is solved by SYMSLV, and if possible, the solution is improved to machine accuracy. If the triangular decomposition fails to exist or if the iterative improvement

fails tc corverge, an error message is printed anc control is transferred to the statement labeled * in the calling program.

REMARKS. Vectors B and X must occupy separate storage. Matrices A and LDU must occupy separate storage.

REFERENCES. Martin, R. S.; Peters, G.; and Wilkinson, J. H. 1966. Iterative Refinement of the Solution cf a Positive Definite System of Equatiors. Numerische Mathematik 8:203-216.

Martin, R. S.; Peters, G.; and Wilkinson, J. H. 1965. Symmetric Decomposition of a Positive Definite Matrix. Numerische Mathematik 7:362-383.

Reinsch, C. and Wilkinson, J. H. eds. 1971. Linear Algebra. Handbook for Automatic Computation. Vol. 2. New York:Springer-Verlag.

LLSQSE SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE LLSQSE(NRA,M1,M,N,A,QR,B,X,RES,FIRST,*,DIAG,PIVOT,
    BETAS,G)
DIMENSION A(NRA,N),QR(NRA,N),B(M),X(N),RES(M),DIAG(N),PIVOT(N),
    BETAS(N),G(N)
INTEGER PIVOT
LOGICAL FIRST
```

PURPOSE. LLSQSE solves an overdetermined system of M equations in N (N .LE. M) unknowns. A specified number of equations can be solved exactly.

INPUT. Input to LLSQSE consists of:

NRA - row dimension of the 2-dimensional arrays A and QR as declared in the calling program dimension statement

M1 - first M1 equations are solved exactly; must satisfy 0 .LE. M1 .LE. N

M - number of equations (rows of the matrix)

N - number of unknowns (columns of the matrix)

A - coefficient matrix

B - right-hand-side vector

FIRST - Boolean switch must be TRUE on the first call and FALSE on subsequent calls having the same coefficient matrix and different right-hand sides.

* - statement label in the callirg program to which control is transferred if the QR-decomposition cannot be completed

OUTPUT. Cutput from LLSQSE includes:

X - solution vector

RES - residuals are stored in locations M1+1 to M.

QR - QR-decompcsition of A

DIAG - contains the diagonal elements of the upper triangular factor R

OTHER PARAMETERS. Additional parameters of LLSQSE consist of:

PIVOT - records the row exchanges in partial pivoting done during the QR-decomposition

BETAS - stores the BETA's

G - vector G (work array)

METHOD. If FIRST is TRUE, matrix A is decomposed by a sequence of elementary Householder transformations into the product of an orthogonal matrix Q and an upper triangular matrix R. Pivoting is accomplished by successively reducing the remaining column having the largest sum of squares. If FIRST is FALSE, the array QR is assumed to already contain the QR-decomposition of matrix A. Control proceeds directly to LLSSLV, which computes the solution X and the residual vectors RES.

REMARKS. If the residuals RES are not needed, the call LLSQSE(NRA,M1,M,N,A,A,B,X,RES,FIRST,*,DIAG,PIVOT,G) can overwrite matrix A with its QR-decomposition. Overwriting A with its QR-decomposition saves MxN words of storage, but the residuals will not be correct.

REFERENCE. Bjorck, A. and Golub, G. 1967. Iterative Refinement of Linear Least Square Solution by Householder Transformation. BIT 7: 322-337.

EXAMPLE. The sample calling program given below can be used to solve the sample data that follows.

Sample Calling Program.
A sample calling program follows:

```
      DIMENSION A(20,10),QR(20,11),B(20),X(10),R(20),PIVOT(10),
     1   BETAS(10)
      DIMENSION DIAG(10),G(10)
      INTEGER PIVOT
      NRA=20
   11 FORMAT(3I5)
   13 FORMAT(8F10.0)
      READ(5,11) M1,M,N
      DO  20 I=1,M
   20 READ(5,13) (A(I,J),J=1,N),B(I)
      CALL LLSQSE(NRA,M1,M,N,A,QR,B,X,R,.TRUE.,&101,DIAG,PIVOT,
     1   BETAS,G)
C     SOLVE FOR A SECOND RIGHT-HAND SIDE VECTOR  B.
      READ(5,13) (B(I),I=1,M)
      CALL LLSQSE(NRA,M1,M,N,A,QR,B,X,R,.FALSE.,&101,DIAG,PIVOT,
     1   BETAS,G)
      STOP
  101 WRITE(6,103)
  103 FORMAT(/" ** ERROR EXIT **"/)
      STOP
      END
```

Sample Data.
The following is data for the preceding program:

```
    36      -630      3360      -7560      7560       463
  -630     14700     -88200     211680   -220500    -13860
  3360     -88200     564480   -1411200   1512000     97020
 -7560     211680   -1411200    3628800  -3969000   -258720
  7560    -220500    1512000   -3969000   4410000    291060
 -2772      83160    -582120    1552320  -1746360   -116424
 -4157     -17820      93555    -261800    288288   -118944
```

LLSQPR SUBRCUTINE.
The syntax is as follows:

```
SUBROUTINE LLSQPR(NRA,M1,M,N,A,QR,B,X,RES,FIRST,*,FAIL,PIVOT,WK)
DIMENSION A(NRA,N),QR(NRA,1),B(M),X(N),RES(M),PIVOT(N),WK(N)
INTEGER PIVOT
LOGICAL FIRST,FAIL
```

PURPOSE. LLSQPR solves an overdetermined system of M equations in N (N .LE. M) unknowns and improves the solution to machine accuracy. A specified number of the equations can be satisfied exactly.

INPUT. Input to LLSQPR consists of:

NRA - row dimension of the 2-dimensional arrays A and QR as declared in the calling program dimension statement; must satisfy NRA .GE. M+2

M1 - first M1 equations are solved exactly; must satisfy 0 .LE. M1 .LE. N

M - number of equations (rows of the matrix)

N - number of unknowns (columns of the matrix)

A - coefficient matrix; declared column dimension must be greater than or equal to N.

B - right-hand-side vector

FIRST - Boolean switch must be TRUE on the first call and FALSE on subsequent calls having the same coefficient matrix and different right-hand sides.

* - statement label in the calling program to which control is transferred if the QR-decomposition cannot be completed

FAIL - set TRUE if the iterative refinement fails to sufficiently improve the solution; otherwise, set to FALSE

OUTPUT. Output from LLSQPR includes:

X - solution vector

RES - residuals are stored in locations M1+1 to M.

QR - QR-decomposition of matrix A, with the diagonal elements of the upper triangular factor R stored in the (M+1)th row. The (M+2)th row and the (N+1)th column are used for working storage. Declared dimensions QR(NRA,NCQ) must satisfy (NRA,NCQ) .GE. (M+2,N+1).

OTHER PARAMETERS. Additicnal parameters of LLSQPR consist of:

PIVOT - records the row exchanges in partial pivoting done during the QR-decomposition; dimension is greater than or equal to N.

WK - working array; dimension is greater than or equal to N.

METHOD. If FIRST is TRUE, matrix A is decomposed by a sequence of elementary Householder transformations into the product of an orthoganal matrix Q and an upper triangular matrix R. Pivoting is accomplished by successively reducing the remaining column having the largest sum of squares. If FIRST is FALSE, the array QR is assumed to already contain the QR-decomposition of matrix A, and control proceeds directly to ACCSLV. In ACCSLV the first set of solutions is taken to be null vectors, and the first two iteration steps are always executed. With DX and DR defined as the changes in X and RES respectively, the iteration for the current right-hand side is terminated when the two conditions below are simultaneously satisfied:

```
E(DX(I)) .GE. .125*E(DX(I-1) .OR. E(DX(I)) .LE. ETA*E(X(1))
E(DR(I)) .GE. .125*E(DR(I-1) .OR. E(DR(I)) .LE. ETA*E(X(1))
```

The character E represents the Euclidean norm, and DX(I) is the value of DX after the Ith iteration. If the iteration has been terminated and the condition

```
E(DX(I)) .GT. 2*ETA*E(X(1)) .AND. E(DR(I)) .GT. 2*ETA*E(R(1))
```

occurs concurrently, the parameter FAIL is set to TRUE.

REFERENCE. Bjorck, A. and Golub, G. 1967. Iterative Refinement of Linear Least Square Solution by Householder Transformation. BII 7:322-337.

EXAMPLE. The sample calling program given below can be used to solve the sample data that follows.

Sample Calling Program.
A sample calling program follows:

```
      LOGICAL FAIL
      INTEGER PIV
      DIMENSION A(22,10),QR(22,11),B(20),X(10),R(20),PIV(10),WK(10)
      NRA=22
   11 FORMAT(3I5)
   13 FORMAT(8F10.0)
      READ(5,11) M1,M,N
      DO  20 I=1,M
   20 READ(5,13) (A(I,J),J=1,N)
      READ(5,13) (B(I),I=1,M)
      CALL LLSQPR(NRA,M1,M,N,A,QR,B,X,R,.TRUE.,&101,FAIL,PIV,WK)
COMMENT  SOLVE FOR A SECOND RIGHT-HAND SIDE VECTOR  B.
      READ(5,13) (B(I),I=1,M)
```

```
      CALL LLSQPR(NRA,M1,M,N,A,QR,B,X,R,.FALSE.,&101,FAIL,PIV,WK)
      STOP
  101 WRITE(6,103)
  103 FORMAT(/" ** ERROR EXIT **"/)
      STOP
      END
```

Sample Data.

The following is data for the preceding program:

```
    36      -630      3360      -7560      7560
  -630     14700     -88200     211680   -220500
  3360    -88200     564480   -1411200   1512000
 -7560    211860   -1411200    3628800  -3969000
  7560   -220500    1512000   -3969000   4410000
 -2772     83160    -582120    1552320  -1746360
   463    -13860      97020    -258720    291060   -116424
 -4157    -17820      93555    -261800    288288   -118944
```

DTLQPR SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE DTLQPR(N,A,LU,B,X,PS,SCALES,DIGITS,DX,DETBAS,DETEXP)
DOUBLE PRECISION A(N,N),LU(N,N),B(N),X(N),SCALES(N),DX(N),DIGITS,
   DETBAS
INTEGER PS(N),DETEXP
```

PURPOSE. DTLQPR solves the linear system A*X = B, improves the solution to machine accuracy, and computes the determinant of the coefficient matrix A.

INPUT. Input to DTLQPR consists of:

N - dimension of coefficient matrix A
A - coefficient matrix
B - right-hand-side vector

OUTPUT. Output from DTLQPR includes:

X - solution vector

PS - contains permuted row indices

DIGITS - number of correct digits in initial solution; must be declared in double precision in the calling program

DETBAS - contains the significant digits of DET(A); must be declared in double precision in the calling program

DETEXP - contains the appropriate power of 10 for DETBAS; must be declared in the calling program as an integer

SCALES, LU, DX - working-storage matrices

METHOD. Gaussian elimination with partial pivoting is used for the initial solution. Accumulated inner products are used to calculate the residuals for the iterative improvement.

REMARKS. The parameter DIGITS contains the number of digits of the solution that do not change during the improvement. The parameters A, LU, B, X, SCALES, and PS must be dimensioned in the calling program as follows:

```
DOUBLE PRECISION A(N,N),LU(N,N),B(N),X(N),SCALES(N)
INTEGER PS(N)
```

SYMLQS SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE SYMLQS(N,A,K,C,Y,P,KAIG)
DOUBLE PRECISION A(N,N),C(K,N),Y(K,N),P(N)
```

PURPOSE. SYMLQS solves the K symmetric systems of equations A*Y = C with the K right-hand sides stored in rows 1 through K of array C and the corresponding K solutions placed in rows 1 through K of Y.

INPUT. Input to SYMLQS consists of:

N - dimension of coefficient matrix

A - array containing the symmetric coefficient matrix. Only the lower triangle of the matrix needs to be stored (strict upper triangle is not used).

K - number of sets of equations to be solved

C - array of right-hand sides

OUTPUT. Output from SYMLQS includes:

A - triangular decomposition of matrix A overwrites the strict lower triargle of matrix A.

Y - solutions of the K systems are stored in rows 1 through K of Y.

KAIG - if this flag equals 1, Cholesky decomposition does not exist.

P - working storage matrix

METHOD. Matrix A (assumed to be positive-definite) is decomposed without pivoting into the triangular form A = LDU. The factors are defined as follows:

L - unit lower triangular matrix
D - diagonal
U - transpose of L

The decomposition is stable for positive-definite matrices. If matrix A is not positive-definite, however, serious loss of accuracy may occur or the decomposition may not exist. If the decomposition does not exist, an error message is written and control is transferred to the end of the subroutine. Subroutine LINEQNS can be used for greater accuracy if the system is not positive-definite. After SYMDECOMPOSE, SYMSOLVE is used successively for each row of C. The rows of C and Y are treated as 1-dimensional arrays (array rows) so that no efficiency is lost by the double subscripting.

REMARKS. Parameters A, C, Y, and P must be dimensioned in the calling program as follows:

```
DOUBLE PRECISION A(N,N),C(K,N),Y(K,N),P(N)
```

The call SYMLQS(N,A,K,C,C,P) overwrites the right-hand-side matrix C with the solution matrix Y.

SYINPR SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE SYINPR(N,A,AINVER,LU,T,DX,R,P,B,KAIG)
DOUBLE PRECISION A(N,N),AINVER(N,N),LU(N,N),T(N),DX(N),R(N),P(N),B(N)
```

PURPOSE. SYINPR computes the complete inverse of the symmetric matrix A, improves it to machine accuracy, and stores the array in AINVER.

INPUT. Input to SYINPR consists of:

N - order of A

A - array containing the matrix to be inverted; only the lower triangle is required.

OUTPUT. Output from SYINPR includes:

AINVER - array containing the full inverse of A
KAIG - if this flag equals 1, Cholesky decomposition does not exist.
LU, T, DX, R, P, B - working storage matrices

METHOD. A copy of matrix A (assumed to be positive-definite) is decomposed without pivoting into the triangular form A = LDU. The factors are defined as follows:

L - unit lower triangular matrix
D - diagonal
U - transpose of L

If matrix A is not positive-definite, the triangular decomposition may not exist. The inverse is calculated and improved column by column, taking into consideration the special form of the columns of the identity matrix. If the triangular decomposition fails to exist or if the iterative improvement of a column fails to converge, an error message is written and control is transferred to the end of the subroutine.

REMARKS. Parameters A, AINVER, LU, T, DX, R, P, and B must be dimensioned in the calling program as follows:

```
DOUBLE PRECISION A(N,N),AINVER(N,N),LU(N,N),T(N),DX(N),R(N),P(N),B(N)
```

SYINPR is much slower than SINVRA and should be used only if a fully accurate inverse is desired.

LINQNS SUBRCUTINE.
The syntax is as follows:

```
SUBRCUTINE LINQNS(N,A,K,C,Y,PS)
DOUBLE PRECISION A(N,N),C(K,N),Y(K,N)
INTEGER PS(N)
```

PURPOSE. LINQNS solves the K systems of equations A*Y = C. The K right-hand sides are stored in rows 1 through K of the KxN matrix C. The corresponding K solutions are placed in rows 1 through K of the KxN matrix Y.

INPUT. Input to LINQNS consists of:

N - dimension of coefficient matrix A

A - coefficient matrix

K - row dimension of matrices C and Y

C - matrix used to store the set of right-hand sides as rows; dimensioned C(K,N)

OUTPUT. Output from LINQNS includes:

Y - matrix used to store the set of solutions as rows; dimensioned Y(K,N)

PS - contains permuted row indices

METHOD. Gaussian elimination with partial pivoting decomposes A into triangular form. SOLVE is then used successively for each row of C. Solutions for any number of right-hand sides can be calculated rapidly. No iterative improvement is used. Each row of matrices Y and C is treated as a 1-dimensional array.

REMARKS. Parameters A, C, Y, and PS must be dimensioned in the calling program as follows:

```
DOUBLE PRECISION A(N,N),C(K,N),Y(K,N)
INTEGER PS(N)
```

Matrix A is overwritten by its LU decomposition during the subroutine.

CXLINQ SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE CXLINQ(N,RA,IA,RLU,ILU,RB,IB,RX,IX,RDX,IDX,SCALES,PS,
   FIRST,IMPROV,IR)
DOUBLE PRECISION RA(N,N),IA(N,N),RLU(N,N),ILU(N,N),RB(N),IB(N),
   RX(N),IX(N),RDX(N),IDX(N),SCALES(N),IR(N)
INTEGER PS(N)
LOGICAL FIRST,IMPROV
```

PURPOSE. CXLINQ solves a system of complex linear equations:

$$(RA+I*IA)(RX+I*IX) = RB+I*IB$$

The solution can be optionally improved to machine accuracy.

INPUT. Input to CXLINQ consists of:

N - complex dimension of coefficient matrix A

RA - real part of coefficient matrix A

IA - imaginary part of coefficient matrix A

RB - real part of right-hand-side vector

IB - imaginary part of right-hand-side vector

FIRST - Boolean value that is TRUE the first time CXLINQ is called with a given coefficient matrix and FALSE for additional calls

IMPROV - Boolean value that specifies whether iterative improvement is desired or not

OUTPUT. Output from CXLINQ includes:

RLU - real part of the triangular decomposition of matrix A
ILU - imaginary part of the triangular decomposition of matrix A
RX - real part of solution vector
IX - imaginary part of solution vector
PS - contains row-permuted indices
RDX, IDX, SCALES, IR - working storage matrices

METHOD. If FIRST is TRUE, CXDECOMPOSE uses Gaussian elimination with partial pivoting to reduce a copy of A to triangular form. The pivoting is accomplished by interchanging the row indices stored in the array PS, rather than interchanging the physical rows. If FIRST is FALSE, the triangular form of A is assumed to be already stored in arrays RLU and ILU. CXSOLVE computes the solution, and if IMPROV is TRUE, the solution is refined to machine accuracy in routine CXIMPROVE. These operations are extensions to the complex arithmetic of the method used by LINPAC. If a singular matrix is detected, an error message is printed and CXLINQ exits. If the IMPROV option is requested and the

solution fails to converge after 23 iterations, an error message is printed and the last iteration is taken as a solution.

REMARKS. The call CXLINQ(N,RA,IA,RA,IA,RB,IB,RX,IX,RDX,IDX,SCALES,PS, FIRST,FALSE,IR) overwrites matrices RA and IA with the triangular decomposition of A. This call saves 2xNxN words of storage.

REFERENCE. Forsythe, G. E. and Moler, C. 1967. Computer Solution of Linear Algebraic Systems. Englewood Cliffs, N.J.:Prentice-Hall.

INVERT SUBRCUTINE.
The syntax is as follcws:

```
SUBRCUTINE INVERT(NM,N,A,AINV,DETERM)
INTEGER NM,N
INTEGER IP(100)
REAL A(NM,N),AINV(NM,N)
REAL X(100)
DOUBLE PRECISION DETERM
```

PURPOSE. INVERT inverts a square matrix.

INPUT. Input to INVERT ccnsists of:

NM - row dimensicn of the 2-dimersional arrays A and AINV as declared in the calling program dimension statement

N - orcer of A

OUTPUT. Cutput from INVERT includes:

A - input matrix is cverwritten by its LU decomposition
AINV - inverse matrix is stored in AINV
DETERM - dcuble precision determinant of the coefficient matrix

SUBROUTINES REQUIRED. The necessary subroutines are INVDCP and DOT.

RESTRICTICNS. The order N or A must not exceed 100. For larger systems the dimensicn of working arrays IP and X must be increased. The arrays A and AINV must have the same actual row dimension and must occupy separate storage.

METHOD. Subrcutine INVDCP uses Gaussian elimination with partial pivoting to triangularize A. The upper triangular factor U overwrites the upper triangle (including the diagonal) of A. The negative of the lower triangular factor L (omitting its unit diagonal) is overwritten cn the strict lower triangle of A. Actual row interchanges are carried out fcr the full matrix. The determinant, product of the diagonal elements cf U, is computed in dcuble precision to minimize the possibility cf exponent overflow or underflow. If the determinant is zero, an error message is printed and a divide by zero occurs in INVERT. Otherwise, INVERT computes the inverse by solving the N systems of equations A*AINV = I, taking full advantage of the zeroes in the columns of the identity matrix.

REFERENCE. Moler, Cleve B. 1972. Matrix Computation with FORTRAN and Paging. Comm. ACM 15:268-270 and 274.

INVERA SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE INVERA(N,A,PS,V)
DOUBLE PRECISION A(N,N),V(N)
INTEGER PS(N)
```

PURPOSE. INVERA replaces matrix A by its inverse.

INPUT. Input to INVERA consists of:

N - dimension of A
A - matrix to be inverted

OUTPUT. Output from INVERA includes:

A - original matrix is overwritten by its inverse.
PS - contains permuted row indices
V - working storage matrix

METHOD. INVERA inverts matrix A in-place: As elements of the inverse are computed, they replace the corresponding elements of A. Inversion in-place saves NxN words of storage over procedure INVERS. INVERA, however, is slower. Procedure TRIANGLDECOMP first decomposes matrix A into triangular form. (TRIANGLDECOMP is set the same as procedure DECOMPOSE in LINPAC except that actual row interchanges are used.) Procedure INVERT then inverts the triangular form by calculating the inverse X of the product LU (the triargular form) from the following conditions:

XL - an upper triangular matrix
UX - a unit lower triangular matrix

To obtain the inverse of A, the columns of X are interchanged in the reverse order as the row interchanges in TRIANGLDECOMP.

REMARKS. The parameters A, V, and PS must be dimensioned in the calling program as follows:

```
DOUBLE PRECISION A(N,N),V(N)
INTEGER PS(N)
```

REFERENCE. Fox, L. 1951. Practical Solution of Linear Equations and Inversion of Matrices. In Nat. Bur. Stand. AMS 39 ed. O. Taussky. pp. 1-54.

INVEPR SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE INVEPR(N,A,LU,B,X,PS,SCALES,DX,DETBAS,DETEXP,AINVER)
DOUBLE PRECISION A(N,N),LU(N,N),B(N),X(N),SCALES(N),DX(N),DETBAS,
   AINVER(N,N)
INTEGER PS(N),DETEXP
```

PURPOSE. INVEPR calculates the inverse of the matrix stored in A(N,N) and stores it in AINVER(N,N). The determinant of A is also calculated.

INPUT. Input to INVEPR consists of:

N - dimension of A
A - matrix to be inverted

OUTPUT. Output from INVEPR includes:

AINVER - array containing the inverse of A

DETBAS - contains the significant digits of DET(A); must be declared in double precision in the calling program

DETEXP - contains the appropriate power of 10 for DETBAS; must be declared as an integer in the calling program

PS - contains permuted row indices

SCALES, X, DX, LU, B - working storage matrices

METHOD. INVEPR calculates the inverse of A by successively solving the N systems of equations represented by A*INVER = I, where I is the NxN identity matrix. Each column of AINVER is calculated to machine precision using iterative refinement.

REMARKS. Parameters A, AINVER, SCALES, X, DX, LU, B, and PS must be dimensioned in the calling program as follows:

```
DOUBLE PRECISION A(N,N),AINVER(N,N),SCALES(N),X(N),DX(N),LU(N,N),B(N)
INTEGER PS(N)
```

Matrices A and AINVER must occupy separate storage.

DETOFA SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE DETOFA(N,A,PS,SCALES,DETBAS,DETEXP)
DOUBLE PRECISION A(N,N),SCALES(N),DETBAS
INTEGER PS(N),DETEXP
```

PURPOSE. DETOFA calculates the determinant of a matrix that has been brought intc LU form by procedure DECCMPOSE.

INPUT. Input to DETOFA consists of:

N - dimension of A
A - matrix whose determinant is to be computed

OUTPUT. Output from DETOFA includes:

PS - contains permuted row indices

DETBAS - contains the significant digits of DET(A); must be declared in double precision in the calling program

DETEXP - contains the appropriate power of 10 for DETBAS; must be declared as an integer in the calling program

SCALES - working storage matrix

METHOD. The call DECOMPOSE(N,A,A,PS,DETSIGN) computes the triangular matrices L and U and the permutation matrix P so that LU = PA. L-I and U are stored in matrix A. Array PS contains the permuted row indices. If matrix A is singular, an exit occurs.

REMARKS. The parameters A, SCALES, and PS must be dimensioned in the calling program as follows:

```
DOUBLE PRECISION A(N,N),SCALES(N)
INTEGER PS(N)
```

SINVRA SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE SINVRA(NRA,N,A,*)
DIMENSION A(NRA,N)
```

PURPOSE. SINVRA inverts the symmetric, positive-definite matrix A.

INPUT. Input tc SINVRA ccnsists of:

NRA - row dimersion of the 2-dimensional arrav A as declared in the calling program dimension statement

N - crder cf A; must be positive

A - symmetric matrix to be inverted. Only the lower triangle of the matrix needs to be stored (strict upper triangle is not used).

* - statement label in the callirg program to which control is transferred if the Cholesky decomposition of A does not exist

OUTPUT. Outpot from SINVRA includes:

A - entire inverse matrix (also symmetric) overwrites A.

METHOD. Matrix A (assumed to be positive-definite) is decomposed without pivoting into the triangular form A = LDU. The factors are defined as follcws:

L - unit lower triangular matrix
D - diagonal
U - transpose cf L

The deccmposition is stable for pcsitive-definite matrices. If A is not positive-definite, however, serioLs loss of accuracy may occur or the deccmposition may nct exist. If the decomposition does not exist, an error message is printed and ccntrcl is transferred to the statement labeled * ir the calling program. Subroutine INVERT can be used for greater accuracy if the system is not positive-definite.

Matrix L with its unit diagonal omitted is overwritten on the strict lower triangle of A. The reciprocals of the nonzero elements of D are stored cn the main diagonal. The inverse of L (unit diagonal omitted) is computed in-place. The L-inverse and D-inverse are then replaced by the lower triangle of the inverse matrix. SYMMET is used to complete the remainder of the inverse matrix.

REFERENCES. Martin, R. S.; Peters, G.; and Wilkinson, J. H. 1965. Symmetric Decomposition of a Positive Definite Matrix. Numerische Mathematik 7:362-383.

Reinsch, C. and Wilkinson, J. H. eds. 1971. Linear Algebra. Handbook for Automatic Computation. Vol. 2. New York:Springer-Verlag.

EXAMPLE. The sample calling program given below can be used to solve the sample data that follows.

Sample Calling Program.
A sample calling program follows:

```
      DIMENSION A(10,10)
      NRA=10
   11 FORMAT(I5)
   13 FORMAT(10F8.0)
      READ(5,11) N
      DO   20 I=1,N
   20 READ(5,13) (A(I,J),J=1,I)
      READ(5,13) (B(I),I=1,N)
      CALL SINVRA(NRA,N,A,&101)
      STOP
  101 WRITE(6,103)
  103 FORMAT(//" ** ERROR EXIT **")
      STOP
      END
```

Sample Data.
The following is sample data for the preceding program:

```
  420.
  210.     140.
  140.     105.      84.
  105.      84.      70.      60.
  875.     539.     399.     319.
```

EIGENVALUES AND EIGENVECTORS.

This section includes FORTRAN subroutines for computing the full set of eigenvalues and eigenvectors of symmetric, real, complex and complex Hermitian matrices. In addition, the subroutines MXEGVL and MNEGVL are included. They compute the K largest or K smallest eigenvalues, respectively, of a symmetric matrix.

For each type of matrix, the basic method used for computing the eigenvalues is preliminary reduction to a simpler form, e.g., tridiagonalization followed by QR or LR (closely related to QR) iterations. Each subroutine documented here consists of two or more calls to a set of working subroutines included in the library under the directory EISPACK. These working subroutines are translations of ALGOL procedures having the same names. The majority of the ALGOL procedures have been developed by J. H. Wilkinson and his collegues. Recognized for several years as the best general-purpose eigenproblem programs, many of these procedures are included in the ALGOL section of this library. Translation to FORTRAN and subsequent extensive testing have been carried out at Argonne National Laboratories and several leading universities under the auspicies of the NSF-sponsored National Activity to Test Software (NATS) project. No written documentation of the EISPACK subroutines is included here because of extensive internal documentation. A separate, machine-readable, EISPACK documentation file is included in the library.

The ALGOL procedures are published together in Handbook for Automatic Computation, Vol. II, "Linear Algebra," J. H. Wilkinson and C. Reinsch, eds., (Springer-Verlag, 1971). Each is also prepublished in various issues of the journal Numerische Mathematik.

The EISPACK series of subroutines have been developed and made available by the NATS project of the Applied Mathematics Division, Argonne National Laboratories, Argonne, Illinois.

SYMVAL SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE SYMVAL(NM,N,A,VALUES)
INTEGER NM,N,IERR
REAL A(NM,N), VALUES(N)
REAL E(100)
```

PURPOSE. SYMVAL finds all N eigenvalues of a real symmetric matrix.

INPUT. Input to SYMVAL consists of:

NM - row dimension of the 2-dimensional array A as declared in the calling program dimension statement

N - order of matrix

A - contains the real symmetric input matrix. Only the lower triangle of the matrix is needed.

OUTPUT. Output from SYMVAL includes:

A - the lower triangle of A is destroyed. The strict upper triangle of A is unaltered.

VALUES - array containing the eigenvalues of the matrix in ascending order

SUBROUTINES REQUIRED. The necessary subroutines are TRED1 and TQL1.

RESTRICTIONS. The order N of the matrix must not exceed 100. For larger matrices, the dimension of the working storage array E must be increased.

METHOD. The matrix is first reduced to tridiagonal form using subroutine TRED1 which employs Householder transformations. The eigenvalues of the tridiagonal matrix are calculated by subroutine TQL1 using QL iterations. If the Kth eigenvalue is not determined after 30 iterations, the value of K is printed and the subroutine exited normally with the first K-1 eigenvalues stored correctly. Failures are exceedingly rare.

REFERENCE. Bowdler, H.; Martin, R. S.; Reinsch, C.; and Wilkinson, J. H. 1968. The QR and QL Algorithms for Symmetric Matrices. Numerische Mathematik 11:293-306.

ACKNOWLEDGEMENT. These FORTRAN translations of TRED1 and TQL1 are part of the EISPACK series of subroutines developed and made available by the NATS project of the Applied Mathematics Division, Argonne National Laboratories, Argonne, Illinois.

SYMVEC SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE SYMVEC(NM,N,A,VALUES,VECTRS)
INTEGER NM,N,IERR
REAL A(NM,N),VECTRS(NM,N), VALUES(N)
REAL E(100)
```

PURPOSE. SYMVEC finds all N eigenvalues and orthonormal eigenvectors of a real symmetric matrix.

INPUT. Input to SYMVEC consists of:

NM - row dimension of the 2-dimensional arrays A and VECTRS as declared in the calling program dimension statement

N - order of matrix

A - contains the real symmetric input matrix. Only the lower triangle of the matrix is needed.

OUTPUT. Output from SYMVEC includes:

A - the lower triangle of A is destroyed. The strict upper triangle of A is unaltered.

VALUES - array containing the eigenvalues of the matrix in ascending order

VECTRS - a 2-dimensional array containing the normalized eigenvectors of A in its first N columns

SUBROUTINES REQUIRED. The necessary subroutines are TRED2 and TQL2.

RESTRICTIONS. The order N of the matrix must not exceed 100. For larger matrices, the dimension of the working storage array E must be increased. The arrays A and VECTRS must have the same actual row dimension and must occupy separate storage.

METHOD. The matrix is first reduced to tridiagonal form using subroutine TRED2 which employs Householder transformations. TQL2 computes the eigenvalues using QL iterations and the orthonormal eigenvectors of the original matrix using information supplied by TRED2. Should the eigenvectors of the tridiagonal matrix be desired instead, the matrix VECTRS must be set to the identity matrix before TQL2 is entered. If the Kth eigenvalue is not determined after 30 iterations, the value of K is printed and the subroutine exited normally with the first K-1 eigenvalues stored correctly. Failures are exceedingly rare.

REFERENCE. Bowdler, H.; Martin, R. S.; Reinsch, C.; and Wilkinson, J. H. 1968. The QR and QL Algorithms for Symmetric Matrices. Numerische Mathematik 11:293-306.

ACKNOWLEDGEMENT. These FORTRAN translations of TRED2 and TQL2 are part of the EISPACK series of subroutines developed and made available by the NATS project of the Applied Mathematics Division, Argonne National Laboratories, Argonne, Illinois.

EIGVAL SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE EIGVAL(NM,N,A,WR,WI)
INTEGER NM,N,LOW,IGH,IERR
REAL A(NM,N), WR(N),WI(N)
```

PURPOSE. EIGVAL finds all N eigenvalues of the general real matrix A.

INPUT. Input to EIGVAL consists of:

NM - row dimension of the 2-dimensional array A as declared in the calling program dimension statement

N - order of matrix

A - contains the general real input matrix

OUTPUT. Output from EIGVAL includes:

A - the input matrix is destroyed by the computation

WR - the real parts of the eigenvalues are stored in the array WR

WI - the imaginary parts of the eigenvalues are stored in the array WI. Complex conjugate eigenvalues are stored consecutively.

SUBROUTINES REQUIRED. The necessary subroutines are BALANC, ORTHES, and HQR.

METHOD. A is first prepared by the subroutine BALANC, which detects any isolated eigenvalues and reduces the norm of A without introducing any roundoff errors. ORTHES is then called to reduce A to an upper Hessenberg form using elementary orthogonal (Householder) transformations. Subroutine HQR, which utilizes the Francis Double Step QR algorithm, is employed to find the eigenvalues of A. If the Kth eigenvalue is not determined after 30 iterations, the value of K is printed and the subroutine exited normally with the eigenvalues of indices K+1, K+2, ..., N stored correctly. Failures are exceedingly rare.

REMARKS. Balancing is strongly recommended because it generally improves accuracy at a small additional cost in time. However, it is not necessary to balance orthogonal or normal matrices. If BALANC is not used, LOW must be set to 1 and UPP to N.

If the original matrix is already of upper Hessenberg form (zeroes below the first sub-diagonal), the call to ORTHES may be removed (subroutine BALANC preserves the Hessenberg form).

REFERENCE. Martin, R. S.; Peters, G.; and Wilkinson, J. H. 1970. The QR Algorithm for Real Hessenberg Matrices. Numerische Mathematik 14:219-231.

ACKNOWLEDGEMENT. These FORTRAN translations of BALANC, ORTHES and HQR are part of the EISPACK series of subroutines developed and made available by the NATS project of the Applied Mathematics Division, Argonne National Laboratories, Argonne, Illinois.

EIGVEC SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE EIGVEC(NM,N,A,WR,WI,Z)
INTEGER NM,N,LOW,IGH,IERR
REAL A(NM,N),Z(NM,N), WR(N),WI(N)
REAL SCALE(100)
```

PURPOSE. EIGVEC finds all N eigenvalues and normalized eigenvectors of the general real matrix A.

INPUT. Input to EIGVEC consists of:

NM - row dimension of the 2-dimensional arrays A and Z as declared in the calling program dimension statement

N - order of matrix

A - contains the general real input matrix

OUTPUT. Output from EIGVEC includes:

A - the input matrix is destroyed by the computation

WR - the real parts of the eigenvalues are stored in the array WR

WI - the imaginary parts of the eigenvalues are stored in the array WI. Complex conjugate eigenvalues are stored consecutively.

Z - a 2-dimensional array containing the normalized eigenvectors of A in its first N columns. If the Kth eigenvalue is real, column K contains its eigenvector. If the Kth eigenvalue is complex with positive imaginary part, column K contains the real part and column K+1 the imaginary part of the Kth eigenvector. The complex conjugate eigenvector is not stored.

SUBROUTINE REQUIRED. The necessary subroutines are BALANC, ORTHES, ORTRAN, HQR2, BALBAK, and NRMLZ.

RESTRICTIONS. The order N of the matrix must not exceed 100. For larger matrices, the dimension of the working storage array SCALE must be increased. The arrays A and Z must have the same actual row dimension and must occupy separate storage.

METHOD. A is first prepared by subroutine BALANC, which detects any isolated eigenvalues and reduces the norm of A without introducing any roundoff errors. ORTHES is then called to reduce A to upper Hessenberg form using elementary orthogonal (Householder) transformations. The matrix defining the similarity reduction of A to upper Hessenberg form is then formed and stored in the array Z. Subroutine ORTRANS, using the information left below

the sub-diagonal of A by ORTHES, completes the preliminary reductions and transformations. Subroutine HQR2 is employed to obtain the eigenvalues of A and the eigenvectors of the balanced matrix. HQR2 utilizes the Francis Double-Step QR algorithm to compute the eigenvalues and back substitution for the eigenvectors. The eigenvectors of the balanced matrix are next transformed into eigenvectors of the original matrix by subroutine BALBAK. Finally, the eigenvectors of A are normalized so that a real eigenvector has its component of maximum magnitude equal to 1.0. A complex eigenvector has its maximum modulus component equal to 1.0 + 0*I.

REMARKS. Balancing is strongly recommended because it generally improves accuracy at a small additional cost in time. However, it is not necessary to balance orthogonal or normal matrices. If BALANC is not used, LOW must be set to 1 and UPP to N. If the original matrix is already of upper Hessenberg form (zeroes below the first sub-diagonal), the calls to ORTHES and ORTRANS may be removed (subroutine BALANC preserves the Hessenberg form). In this case Z must be set to the identity matrix before HQR2 is entered. If un-normalined eigenvectors are sufficient, the call to subroutine NRMLZ may be removed. If the Kth eigenvalue is not determined after 30 iterations, the value of K is printed and the subroutine exited normally with the eigenvalues of indices K+1, K+2, ..., N stored correctly. Failures are exceedingly rare.

REFERENCE. Peters, G. and Wilkinson, J. H. 1970. Eigenvectors of Real and Complex Matrices by LR and QR Triangularizations. Numerische Mathematik 16:181-204.

ACKNOWLEDGEMENT. These FORTRAN translations of BALANC, ORTHES, ORTRAN, HQR2 and BALBAK are part of the EISPACK series of subroutines developed and made available by NATS project of the Applied Mathematics Division, Argonne National Laboratories, Argonne, Illinois.

CEIGVL SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE CEIGVL(NM,N,AR,AI,WR,WI)
INTEGER NM,N,LOW,IGH,IERR
REAL AR(NM,N),AI(NM,N), WR(N),WI(N)
```

PURPOSE. CEIGVL finds all N eigenvalues of the general complex matrix A.

INPUT. Input to CEIGVL consists of:

NM = row dimension of the 2-dimensional arrays AR and AI as declared in calling program dimension statement

N - complex order of the complex matrix A

AR,AI - the real and imaginary parts of A are input in the arrays AR and AI, respectively

OUTPUT. Output from CEIGVL includes:

AR,AI - the input matrix is destroyed in the computation

WR,WI - the real and imaginary parts of the eigenvalues are placed in the arrays WR and WI, respectively

SUBROUTINES REQUIRED. The necessary subroutines are CBAL, COMHES, and COMLR.

METHOD. A is first prepared by subroutine CBAL, which detects any isolated eigenvalues, reduces the norm of the matrix, and equalizes the L1 norm of corresponding rows and columns without introducing roundoff error. The balanced matrix is next reduced to the upper Hessenberg form by subroutine COMHES utilizing stabilized elementary similarity transformations. The eigenvalues of the upper Hessenberg matrix (which are also the eigenvalues of A) are computed by subroutine COMLR which employs the LR algorithm with partial pivoting. If the Kth eigenvalue is not determined after 30 iterations, the value of K is printed and the subroutine exited normally with the eigenvalues of indices K+1, K+2, ..., N stored correctly. Failures are exceedingly rare.

REMARKS. Balancing is strongly recommended because it generally improves accuracy at a small additional cost in time. However, it is not necessary to balance normal matrices. If balancing is not used, LOW must be set to 1 and UPP to N. If the original matrix is already of upper Hessenberg form (zeroes below the first sub-diagonal), the call to COMHES may be removed (subroutine CBAL preserves the Hessenberg form).

REFERENCE. Martin, R. S. and Wilkinson, J. H. 1968. The Modified LR Algorithm for Complex Hessenberg Matrices. Numerische Mathematik 12:369-376.

ACKNOWLEDGEMENT. These FORTRAN translations of CBAL, COMHES and COMLR are part of the EISPACK of subroutines, developed and made available by the NATS project of the Applied Mathematics Division, Argonne National Laboratories, Argonne, Illinois.

CEIGVC SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE CEIGVC(NM,N,AR,AI,WR,WI,ZR,ZI)
INTEGER NM,N,LOW,IGH,IERR
INTEGER INT(100)
REAL AR(NM,N),AI(NM,N), ZR(NM,N),ZI(NM,N), WR(N),WI(N)
REAL SCALE(100)
```

PURPOSE. CEIGVC finds all N eigenvalues and normalized column eigenvectors of the general complex matrix A.

INPUT. Input to CEIGVC consists of:

NM - row dimension of the 2-dimensional arrays AR, AI, ZR, and ZI as declared in the calling program dimension statement

N - complex order of the complex matrix A

AR,AI - the real and imaginary parts of A are input in the arrays AR and AI, respectively

OUTPUT. Output from CEIGVC consists of:

AR,AI - the strict upper triangles of AR and AI are overwritten with the eigenvectors or the triangular matrix produced by the LR algorithm

WR,WI - the real and imaginary parts of the eigenvalues are placed in the arrays WR and WI, respectively

ZR,ZI - the real and imaginary parts of the normalized eigenvectors are stored in the columns of ZR and ZI

SUBROUTINES REQUIRED. The necessary subroutines are CBAL, COMHES, COMLR2, CBABK2, and CNRMLZ.

RESTRICTIONS. The order N of the matrix must not exceed 100. For larger matrices, the dimension of the working storage arrays INT and SCALE must be increased. The arrays AR,AI and ZR,ZI must have the same actual row dimension and must occupy separate storage.

METHOD. A is first prepared by subroutine CBAL, which detects any isolated eigenvalues, reduces the norm of the matrix, and equalizes the L1 norms of corresponding rows and columns without introducing roundoff error. The balanced matrix is next reduced to the upper Hessenberg form by subroutine COMHES utilizing stabilized elementary similarity transformations.

After completing the preliminary reductions, LR iterations with partial pivoting are employed by COMLR2 to compute the eigenvalues of A. COMLR2 also computes the eigenvectors of the balanced matrix by a back substitution process. These are transformed into eigenvectors of

the original matrix A by subroutine CBABK2. Finally, each eigenvector is normalized so that its component of maximum modulus is equal to 1.0 + 0*I.

REMARKS. Balancing is strongly recommended because it generally improves accuracy at a small additional cost in time. However, it is not necessary to balance normal matrices. If balancing is not used, LOW must be set to 1 and UPP to N.

If the original matrix is already of upper Hessenberg form (zeroes below the first sub-diagonal), the call to COMHES may be removed (subroutine CBAL preserves the Hessenberg form). But if un-normalized eigenvectors are sufficient, subroutine CNRMLZ may be omitted. If the Kth eigenvalue is not determined after 30 iterations, the value of K is printed and the subroutine exited normally with the eigenvalues of indices K+1, K+2, ..., N stored correctly. Failures are exceedingly rare.

REFERENCE. Peters, G. and Wilkinson, J. H. 1970. Eigenvectors of Real and Complex Matrices by LR and QR Triangularizations. Numerische Mathematik 16:181-204.

ACKNOWLEDGEMENT. These FORTRAN translations of CBAL, COMHES, COMLR2 and CBABK2 are part of the EISPACK series of subroutines developed and made available by the NATS project of the Applied Mathematics Division, Argonne National Laboratories, Argonne, Illinois.

HREGVL SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE HREGVL(NM,N,AR,AI,VALUES)
INTEGER NM,N,IERR
REAL AR(NM,N)AI(NM,N), VALUES(N)
REAL TAU(2,100), E(100)
```

PURPOSE. HREGVL finds all N eigenvalues of a complex Hermitian matrix.

INPUT. Input to HREGVL consists of:

NM - row dimension of the 2-dimensional arrays AR and AI as declared in calling program dimension statement

N - order of matrix

AR,AI - contains the real and imaginary parts of the complex Hermitian input matrix. Only the lower triangles are needed.

OUTPUT. Output from HREGVL includes:

AR,AI - the lower triangles of AR and AI are destroyed. The strict upper triangles are unaltered.

VALUES - array containing the eigenvalues of the matrix in ascending order

SUBROUTINES REQUIRED. The necessary subroutines are HTRIDI and TQL1.

RESTRICTIONS. The order N of the matrix must not exceed 100. For larger matrices, the dimension of the working storage arrays E and TAU must be increased.

METHOD. The matrix is first reduced to tridiagonal form, using subroutine HTRIDI which employs unitary similarity transformations. The eigenvalues of the tridiagonal matrix are then calculated by TQL1, using QL transformations. If the Kth eigenvalue is not determined after 30 iterations, the value of K is printed and the subroutine exited normally with the first K-1 eigenvalues stored correctly. Failures are exceedingly rare.

REFERENCE. Martin, R. S.; Reinsch, C.; and Wilkinson, J. H. 1968. Householder's Tridiagonalization of a Symmetric Matrix. Numerische Mathematik 11:181-195.

ACKNOWLEDGEMENT. These FORTRAN translations of TQL1 and the complex analogue of TRED1 are part of the EISPACK series of subroutines developed and made available by the NATS project of the Applied Mathematics Division, Argonne National Laboratories, Argonne, Illinois.

HREGVC SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE HREGVC(NM,N,AR,AI,VALUES,ZR,ZI)
INTEGER NM,N,IERR
REAL AR(NM,N),AI(NM,N), ZR(NM,N),ZI(NM,N), VALUES(N)
REAL TAU(2,100), E(100)
```

PURPOSE. HREGVC finds all N eigenvalues and orthonormal column eigenvectors of the complex Hermitian matrix A.

INPUT. Input to HREGVC consists of:

NM - row dimension of the 2-dimensional arrays AR, AI, ZR, and ZI as declared in calling program dimension statement

N - order of matrix

AR,AI - contains the real and imaginary parts of the complex Hermitian input matrix. Only the lower triangles are needed.

OUTPUT. Output from HREGVC includes:

AR,AI - the lower triangles of AR and AI are destroyed. The strict upper triangles are unaltered.

VALUES - array containing the eigenvalues of the matrix in ascending order

ZR,ZI - the real and imaginary parts of the orthonormal eigenvectors are stored in the columns of ZR and ZI.

SUBROUTINES REQUIRED. The necessary subroutines are HTRIDI, TQL1, and HTRIBK.

RESTRICTIONS. The order N of the matrix must not exceed 100. For larger matrices, the dimension of the working storage arrays E and TAU must be increased. The arrays AR, AI, ZR, and ZI must have the same actual row dimension and must occupy separate storage.

METHOD. The matrix is first reduced to tridiagonal form using subroutine HTRIDI which employs unitary similarity transformations. Next, the array ZR is initialized to the identity matrix, so that TQL2 computes the eigenvectors, as well as the eigenvalues, of the tridiagonal matrix supplied to it by HTRIDI. Finally, HTRIBK transforms the orthonormal eigenvectors of the tridiagonal matrix into orthonormal eigenvectors of the original matrix. By similarity, the eigenvalues of the tridiagonal matrix and the original matrix are the same. If the Kth eigenvalue is not determined after 30 iterations, the value of K is printed and the subroutine exited normally with the first K-1 eigenvalues stored correctly. Failures are exceedingly rare.

REFERENCE. Martin, R. S.; Reinsch, C.; and Wilkinson, J. H. 1968. Householder's Tridiagonalization of a Symmetric Matrix. Numerische Mathematik 11:181-195.

ACKNOWLEDGEMENT. These FORTRAN translations of TQL1 and the complex analogues of TRED1 and TRBAK1 are part of the EISPACK series of subroutines developed and made available by the NATS project of the Applied Mathematics Division, Argonne National Laboratories, Argonne, Illinois.

MXEGVL SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE MXEGVL(NM,N,A,EPS1,K,VALUES,BOUNDS)
INTEGER NM,N,K,IDEF,IERR
INTEGER IND(25)
REAL EPSI
REAL A(NM,N), VALUES(N),BOUNDS(N)
REAL E(100)
LOGICAL TYPE
```

PURPOSE. MXEGVL finds the K algebraically largest eigenvalues of a real symmetric matrix.

INPUT. Input to MXEGVL consists of:

NM - row dimension of the 2-dimensional array A as declared in calling program dimension statement

N - order of matrix

A - contains the real symmetric input matrix. Only the lower triangle of the matrix is needed.

EPSI - absolute error tolerance. The theoretical absolute error in the Rth eigenvalue is usually not greater than R times EPSI.

K - the number of eigenvalues to be computed. K should not exceed N/4.

OUTPUT. Output from MXEGVL includes:

A - the lower triangle of A is destroyed. The strict upper triangle of A is unaltered.

VALUES - array containing the K algebraically largest eigenvalues in descending order

BOUNDS - array containing computed error bounds for the eigenvalues in the array VALUES

SUBROUTINES REQUIRED. The necessary subroutines are TRED1 and RATQR.

RESTRICTIONS. The order N of the matrix must not exceed 100. The number K of eigenvalues to be computed must not exceed 25. For larger problems, the dimension of the working storage arrays E and IND must be increased.

METHOD. The matrix is first reduced to tridiagonal form using subroutine TRED1 which employs Householder transformations. The K largest eigenvalues of the tridiagonal matrix are calculated by subroutine RATQR which uses rational QR iterations with Newton corrections. No failure exit is taken. If the iteration for the Rth eigenvalue ceases

to converge, the last value is taken as an approximation to the Rth eigenvalue and the computation proceeds to the next eigenvalue.

REMARKS. If more than N/4 eigenvalues are wanted, SYMVAL should be used. If the matrix is known to be definite, the statement IDEF = 0 may be replaced by IDEF = 1 if positive-definite or IDEF = -1 if negative-definite, reducing execution time.

REFERENCE. Reinsch, C. and Bauer, F. L. 1968. Rational QR Transformation with Newton Shift for Symmetric Matrices. Numerische Mathematik 11:264-272.

ACKNOWLEDGEMENT. These FORTRAN translations of TRED1 and RATQR are part of the EISPACK series of subroutines developed and made available by the NATS project of the Applied Mathematics Division, Argonne National Laboratories, Argonne, Illinois.

MNEGVL SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE MNEGVL(NM,N,A,EPS1,K,VALUES,BOUNDS)
INTEGER NM,N,K,IDEF,IERR
INTEGER IND(25)
REAL EPS1
REAL A(NM,N), VALUES(N),BOUNDS(N)
REAL E(100)
LOGICAL TYPE
```

PURPOSE. MNEGVL finds the K algebraically smallest eigenvalues of a real symmetic matrix.

INPUT. Input to MNEGVL consists of:

NM - row dimension of the 2-dimensional array A as declared in the calling program dimension statement

N - order of matrix

A - contains the real symmetric input matrix. Only the lower triangle of the matrix is needed.

EPS1 - absolute error tolerance. The theoretical absolute error in the Rth eigenvalue is usually not greater than R times EPS1.

K - the number of eigenvalues to be computed. K should not exceed N/4.

OUTPUT. Output from MNEGVL includes:

A - the lower triangle of A is destroyed. The strict upper triangle of A is unaltered.

VALUES - array containing the K algebraically smallest eigenvalues in ascending order

BOUNDS - array containing computed error bounds for the eigenvalues in the array values

SUBROUTINES REQUIRED. The necessary subroutines are TRED1 and RATQR.

RESTRICTIONS. The order N of the matrix must not exceed 100. The number K of eigenvalues to be computed must not exceed 25. For larger problems the dimension of the working storage array E and the array IND must be increased.

METHOD. The matrix is first reduced to tridiagonal form using subroutine TRED1 which employs Householder transformations. The K smallest eigenvalues of the tridiagonal matrix are calculated by subroutine RATQR which uses rational QR iterations with Newton corrections. No failure exit is taken. If the iteration for the

Rth eigenvalue ceases to converge, the last value is taken as an approximation tc the Rth eigenvalue and the computation proceeds to the rext eigenvalue.

REMARKS. If more than N/4 eigenvalues are wanted, SYMVAL should be used. If the matrix is known to be definite, the statement IDEF = 0 may be replaced by IDEF = 1 if pcsitive-definite or IDEF = -1 if negative-definite, reducing execution time.

REFERENCE. Reirsch, C. and Bauer, F. L. 1968. Rational QR Transformation with Newton Shift for Symmetric Matrices. Numerische Mathematik 11:264-272.

ACKNOWLEDGEMENT. These FORTRAN translations of TRED1 and RATQR are part of the EISPACK series of subroutines developed and made available by the NATS project of the Applied Mathematics Division, Argonne National Laboratories, Argonne, Illinois.

MATRIX ARITHMETIC.

The following subroutines perform common arithmetic operations on square matrices and addition, subtraction, and multiplication on conforming rectangular matrices.

MMULTM SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE MMULTM(N,A,B,C)
DIMENSION A(N,N),B(N,N),C(N,N)
```

PURPOSE. MMULTM computes the matrix product C = A*B with matrices A and B stored in A[1:N,1:N] and B[1:N,1:N], respectively. The product is stored in C[1:N,1:N].

INPUT. Input to MMULTM consists of:

N - dimension of A, B, and C
A - first input matrix
B - second input matrix

OUTPUT. Output from MMULTM includes:

C - output matrix A*B

METHOD. C(I,J) is defined to be the inner product of the Ith row of A and the Jth column of B.

REMARKS. Matrix C may not be overwritten on either A or B.

TRANSP SUBRCUTINE.
The syntax is as follows:

```
SUBROUTINE TRANSP(N,A)
DIMENSION A(N,N)
```

PURPOSE. TRANSP replaces the matrix stored in A[1:N,1:N] by its transpose.

INPUT. Input tc TRANSP consists of:

N - dimension cf A
A - matrix tc be transposed

OUTPUT. Output from TRANSP includes:

A - transpcsition of matrix A

METHOD. The (I,J)th element of matrix A is replaced by the (J,I)th element of A.

AMULTR SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE AMULTR(N,A,C)
DIMENSION A(N,N),C(N,N)
```

PURPOSE. AMULTR computes the product C = A*A-transpose of the matrix stored in A[1:N,1:N] and its transpose. The result is stored in C[1:N,1:N].

INPUT. Input to AMULTR consists of:

N - dimension of A and C
A - input matrix

OUTPUT. Output from AMULTR includes:

C - output matrix A*A-transpose

METHOD. C(I,J) is defined to be the inner product of the Ith row of matrix A and the Jth column of matrix A.

REMARKS. Matrix C may not be overwritten on matrix A.

AMULBT SUBROUTINE.
The syntax is as follows:

```
SUBRCUTINE AMULBT(N,A,B,C)
DIMENSION A(N,N),B(N,N),C(N,N)
```

PURPOSE. AMULBT computes the product C = A*B-transpose of the matrix stored in A[1:N,1:N] and the transpose of the matrix stored in B[1:N,1:N]. The result is stored in C[1:N,1:N].

INPUT. Input to AMULBT consists of:

N - dimension of A, B, and C
A - first input matrix
B - second input matrix

OUTPUT. Output from AMULBT includes:

C - output matrix A*B-transpose

METHOD. Rather than actually calculating the transpose of matrix B, AMULBT takes the elements of B by rows instead of by columns for premultiplication by matrix A.

REMARKS. Matrix C may not be overwritten on matrix A or B.

MATVEC SUBRCUTINE.
The syntax is as follows:

```
SUBRCUTINE MATVEC(N,A,X,Y)
DIMENSION A(N,N),X(N),Y(N)
```

PURPOSE. MATVEC computes the product Y = A*X of the NxN matrix stored in A[1:N,1:N] and the Nx1 column vector stored in X[1:N]. The result is stored in Y[1:N].

INPUT. Input to MATVEC consists of:

N - square dimension of A and linear dimension of X and Y
A - input matrix
X - input vector

OUTPUT. Output from MATVEC includes:

Y - output vector A*X

METHOD. Y(I) is defined to be the inrer product of the Ith row of matrix A and vector X.

REMARKS. Vector Y may not be overwritten on vector X.

VECMAT SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE VECMAT(N,A,X,Y)
DIMENSION A(N,N),X(N),Y(N)
```

PURPOSE. VECMAT computes the product Y = X*A of the 1xN row vector stored in X[1:N] and the matrix stored in A[1:N,1:N]. The result is stored in Y[1:N].

INPUT. Input to VECMAT consists of:

N - square dimension of A and linear dimension of X and Y
A - input matrix
X - input vector

OUTPUT. Output from VECMAT includes:

Y - output vector X*A

METHOD. The Ith component of vector Y is defined to be the inner product of vector X and the Ith column of matrix A.

REMARKS. Vectors X and Y must occupy separate storage.

VPLURV SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE VPLURV(N,AI,R,X,BI)
DIMENSION AI(N),X(N),BI(N)
```

PURPOSE. VPLURV adds the scalar multiple R*X of the vector stored in X[1:N] and the real number R to the vector stored in AI[1:N]. The result is stored in BI[1:N].

INPUT. Input to VPLURV consists of:

N - dimension of AI, X, and BI
AI - first input vector
R - real multiplier
X - second input vector

OUTPUT. Output from VPLURV includes:

BI - output vector

METHOD. Vector BI is defined by BI(J) = AI(J)+R*X(J).

MMINDG SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE MMINDG(N,D,A,C)
DIMENSION A(N,N),C(N,N),D(N)
```

PURPOSE. MMINDG subtracts the diagonal matrix that is stored as a 1-dimensional array in D[1:N] from the matrix stored in A[1:N,1:N]. The result C = A-D is stored in C[1:N,1:N].

INPUT. Input to MMINDG consists of:

N - square dimension of A and C and linear dimension of D
D - input diagonal matrix stored as a vector
A - input matrix

OUTPUT. Output from MMINDG includes:

C - output matrix A-D

METHOD. The elements of array D are subtracted from the corresponding diagonal elements of matrix A.

REMARKS. Matrix C can be overwritten on matrix A by making the call MMINDG(N,D,A,A).

MMULTD SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE MMULTD(N,D,A,C)
DIMENSION A(N,N),C(N,N),D(N)
```

PURPOSE. MMULTD postmultiplies the matrix stored in A[1:N,1:N] by the diagonal matrix that is stored as a 1-dimensional array in D[1:N]. The result C = A*D is stored in C[1:N,1:N].

INPUT. Input tc MMULTD consists of:

N - square dimension of A and C and linear dimension of D
D - input diagonal matrix stored as a vector
A - input matrix

OUTPUT. Output from MMULTD includes:

C - output matrix A*D

METHOD. The elements of the Jth column of matrix A are multiplied by D(J) to obtain the Jth column of matrix C.

REMARKS. Matrix C can be overwritten on matrix A by making the call MMULTD(N,D,A,A).

DMULTM SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE DMULTM(N,D,A,C)
DIMENSION A(N,N),C(N,N),D(N)
```

PURPOSE. DMULTM premultiplies the matrix stored in A[1:N,1:N] by the diagonal matrix that is stored as a 1-dimensional array in D[1:N]. The result, C = D*A is stored in C[1:N,1:N].

INPUT. Input to DMULTM consists of:

N - square dimension of A and C and linear dimension of D
D - input diagonal matrix stored as a vector
A - input matrix

OUTPUT. Output from DMULTM includes:

C - output matrix D*A

METHOD. The elements of the Ith row of matrix A are multiplied by D(I) to obtain the Ith row of matrix C.

REMARKS. Matrix C can be overwritten on matrix A by making the call DMULTM(N,D,A,A).

RPLUSM SUBRCUTINE.
The syntax is as follcws:

```
SUBRCUTINE RPLLSM(N,R,A,B)
DIMENSION A(N,N),B(N,N)
```

PURPOSE. RPLUSM adds the real number R to each component of the matrix A[1:N,1:N]. The result B = R+A is stored in B[1:N,1:N].

INPUT. Input to RPLUSM consists of:

N - dimension of A and B
R - input real number
A - input matrix

OUTPUT. Output from RPLUSM includes:

B - cutput matrix R+A

REMARKS. Matrix B can be overwritten on matrix A by making the call RPLUSM(N,R,A,A).

RMINM SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE RMINM(N,R,A,B)
DIMENSION A(N,N),B(N,N)
```

PURPOSE. RMINM subtracts each component of the matrix stored in A[1:N,1:N] from the real number R. The result B = R-A is stored in B[1:N,1:N].

INPUT. Input to RMINM consists of:

N - dimension of A and B
R - input real number
A - input matrix

OUTPUT. Output from RMINM includes:

B - output matrix R-A

REMARKS. Matrix B can be overwritten on matrix A by making the call RMINM(N,R,A,A).

RMULTM SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE RMULTM(N,R,A,B)
DIMENSION A(N,N),B(N,N)
```

PURPOSE. RMULTM multiplies each component of the matrix stored in A[1:N,1:N] by the real number R. The result B = R*A is stored in B[1:N,1:N].

INPUT. Input to RMULTM consists of:

N - dimension of A and B
R - input real number
A - input matrix

OUTPUT. Output from RMULTM includes:

B - output matrix R*A

REMARKS. Matrix B can be overwritten on matrix A by making the call RMULTM(N,R,A,A).

MCOPY SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE MCOPY(N,A,B)
DIMENSION A(N,N),B(N,N)
```

PURPOSE. MCOPY copies the matrix stored in A[1:N,1:N] into the matrix stored in B[1:N,1:N].

INPUT. Input to MCOPY consists of:

N - dimension of A and B
A - matrix to be copied

OUTPUT. Output from MCOPY includes:

B - output matrix (copy of A)

METHOD. Matrix A is copied into matrix B element by element.

RECMPM SUBROUTINE.
The syntax is as follows:

```
SUBRCUTINE RECMPM(N,M,A,B,C)
DIMENSION A(N,M),B(N,M),C(N,M)
```

PURPOSE. RECMPM computes the sum A+B of the rectangular matrices A[1:N,1:M] and B[1:N,1:M]. The result is stored in C[1:N,1:M].

INPUT. Input tc RECMPM ccnsists of:

N - row dimension cf A, B, and C
M - column dimension of A, B, and C
A - first input matrix
B - second input matrix

OUTPUT. Cutput from RECMPM includes:

C - output matrix A+B

METHOD. The matrices are added elemert by element.

REMARKS. Either matrix A or B can be overwritten with matrix C by making the call RECMPM(N,M,A,B,A) or RECMPM(N,M,A,B,B), respectively.

RECMMM SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE RECMMM(N,M,A,B,C)
DIMENSION A(N,M),B(N,M),C(N,M)
```

PURPOSE. RECMMM computes the differerce C = A-B between the rectangular matrices A[1:N,1:M] and B[1:N,1:M]. The result is stored in C[1:N,1:M].

INPUT. Input tc RECMMM consists of:

N - row dimension of A, B, and C
M - column dimension of A, B, and C
A - first input matrix
B - second input matrix

OUTPUT. Cutput from RECMMM includes:

C - output matrix A-B

METHOD. The matrices are subtracted element by element.

REMARKS. Either matrix A or B can be overwritten with matrix C by making the call RECMMM(N,M,A,B,A) or RECMMM(N,M,A,B,B), respectively.

RECMMU SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE RECMMU(N,K,M,A,B,C)
DIMENSION A(N,K),B(K,M),C(N,M)
```

PURPOSE. RECMMU computes the product C = A*B of the N-rowed, K-columned matrix A[1:N,1:K] and the K-rowed, M-columned matrix B[1:K,1:M]. The result is stored in C[1:N,1:M].

INPUT. Input to RECMMU consists of:

N - row dimension of A and C
K - column dimension of A and row dimension of B
M - column dimension of B and C
A - first input matrix
B - second input matrix

OUTPUT. Output from RECMMU includes:

C - output matrix A*B

METHOD. The (I,J)th element of matrix C is defined to be the inner product of the Ith row of matrix A and the Jth column of matrix B.

REMARKS. Matrices A, B, and C must occupy separate storage.

SECTION 9

INTERPOLATION

GENERAL.

The subroutines in this section all concern interpolation; that is, they give approximate function values at points between known function values. In the 1-dimensional case, the function F(X) is known at certain points, X(1),..., X(N), called mesh points; and approximations to its values at other points in the interval (X(1), X(N)) are desired. In the 2-dimensional case, the function F(X,Y) is known at the vertices of the rectangular grid (X(I), Y(J)), I=1,...,NX, J=1,..., NY. The approximate values are desired at other points in the rectangle (X(1), X(NX)) x (Y(1), Y(NY)). Cubic spline interpolation (in the 1-dimensional case) and bicubic spline interpolation (in the 2-dimensional case) are used to approximate the function values at the non-mesh points.

A cubic spline of interpolation to the function F(X) on the mesh X(1),...,X(N) is a function S(X), satisfying the following conditions:

a. S(X(I)) = F(X(I)), I=1,...,N.

b. S(X) is a cubic polynomial on each mesh interval (X(I), X(I+1)), I=1,..., N-1 (but not necessarily the same one on each interval).

c. S,S',S" are continuous on (X(1), X(N)).

In order to specify S uniquely, two additional conditions are required. There are several possibilities of which the two seemingly most natural are allowed:

a. The user may supply the (approximate) first derivatives of the function F at X(1) and X(N).

b. In the absence of any data about these, the natural cubic spline which has S" (X(1)) = S" (X(N)) = 0 may be requested. (The user may desire the natural cubic spline even in some cases in which data about the endpoint derivatives of F is available.)

The cubic spline is distinguished by the following property: Of all twice continuously differentiable functions interpolating to the function F on the mesh and satisfying the end conditions mentioned above, it has minimum curvature in the sense that the integral over (X(1),X(N)) of (S"(X))**2 is a minimum. (For additional optimal properties and convergence results, see the reference.)

For 2-dimensional interpolation, only natural bicubic splines are given, being similar to the cubic spline; but, being a function of two variables, they are bicubic functions (a cubic in X times a cubic in Y) on each mesh rectangle (X(I),X(I+1)) X (Y(J),Y(J+1)). The end conditions assumed are that the second partial derivatives of S(X,Y) with respect to X are zero at (X(1),Y(J)) and (X(NX),Y(J)), J=1,...,NY. The second

partial derivatives of S(X,Y) with respect to Y are zero at (X(I),Y(1)) and (X(I),Y(NY)), I=1,...,NX.

All of the routines may be classified by the following criteria:

a. Dimension of 1- or 2-dimensional data.
b. Spacing of equally or unequally spaced input.
c. Procedure of data densification or interpolation desired.

In addition, the 1-dimensional routines each have two entry points depending on whether the natural spline is desired or the user wishes to specify the endpoint slopes. The difference between a data densifier and an interpolator is merely a question of where the user requires the evaluation of the spline approximation to the function. If the user requires a large number of approximate function values between two adjacent mesh points, a data densifier should be used. (For 1-dimensional, equally spaced data, results may be obtained for several consecutive mesh intervals at once.) Alternatively, if the user requires the approximate function values at a number of specific points in several mesh intervals, an interpolator routine is applicable. In any case, whatever results can be obtained with one, can be obtained with the other although with a possible loss of efficiency.

NAMING CONVENTION.
All spline interpolation routines in this section begin with the letters CS for cubic spline. Following them is a DD for data densifier or an I for interpolator; and lastly an E for equally spaced data or a U for unequally spaced data. If the next character is a 2, the routine is for 2-dimensional data. In the 1-dimensional cases, no additional character indicates the natural cubic spline; and a final D indicates a spline with specified endpoint derivatives. (A name ending with a D is an entry to the similarly named routine without the D.)

For example, on the mesh interval (1,2) with M=9, either CSDDE(D) or CSDDU(D) then returns 11 values, S(1), S(1.1),..., S(1.9), S(2) where S(1) = F(1), S(2) = F(2) and S(1.1),..., S(1.9) are approximate. On (1,2) X (3,4) with IMX=11, IMY=6, either CSDDE2 or CSDDU2 then returns 66 values:

S(1,3), S(1.1,3), . . . S(1.9,3), S(2,3),
S(1,3.2), S(1.1,3.2). . S(1.9,3.2), S(2,3.2),
.
.
.
S(1,3.8), S(1.1,3.8), . . ., S(1.9,3.8), S(2,3.8),
S(1,4), S(1.1,4), ..., S(1.9,4), S(2,4)

Of these, S(1,3), S(2,3), S(1,4), and S(2,4) are exact.

The routines with equally spaced data utilize the method of cardinal splines. A cardinal spline on a given mesh is one which interpolates to one at a mesh point and to zero at all other mesh points. The spline of interpolation to a given set of N data can be written as a linear combination of N+2 cardinal splines. (The two additional ones

are to handle the auxiliary end conditions.) For equally spaced data, all cardinal splines are translates of each other and this fact saves considerably on storage requirements.

For unequally spaced data, the method of cardinal spline interpolation is no longer very efficient because the translation property is lost. The following method is therefore employed. A simple calculation shows that the cubic spline of interpolation can be represented on any mesh interval (X(I), X(I+1)) in terms of the two known quantities F(X(I)), (F(X(I+1)) and the two unknown quantities S"(X(I)), S"(X(I+1)). A tridiagonal system of N linear equations in the N unknowns S"(X(I)), I=1,...,N can be found expressing the continuity of S' at the mesh points. Solution of this system by the efficient Thomas algorithm gives the necessary information for evaluation of the spline.

REFERENCE. Ahlberg, J. H.; Nilson, E. N.; and Walsh, J. L. 1967. The Theory of Splines and Their Applications. New York:Academic Press.

CSDDE SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE CSDDE (A,B,N,F,J,L,M,G,*)
AND ENTRY CSDDED (A,B,N,F,J,L,M,G,*,M1,MN)
REAL A,B,F,G,M1,MN
INTEGER N,J,L,M
DIMENSION F(N),G(1)
DIMENSION G((L-J)*(M+1)+1)
```

PURPOSE. Subroutine CSDDE computes the natural cubic spline interpolate to a set of N equally spaced data and returns a vector of (L-J)*(M+1)+1 equally spaced interpolated function values.

Entry CSDDED computes the cubic spline interpolate to a set of N equally spaced data with specified endpoint first derivatives and returns a vector of (L-J)*(M+1)+1 equally spaced interpolated function values.

INPUT. Input to CSDDE consists of:

A - left-hand endpoint of the interval (A,B) over which data is given

B - right-hand endpoint of the interval (A,B)

N - number of data points

F - vector of N equally spaced function values F(1),F(2), ...,F(N), where F(I) is a function value at A+(I-1)*H and H=(B-A)/(N-1)

J - number corresponding to the Jth abscissa, the left-hand endpoint of the interval (A+(J-1)*H,A+(L-1)*H) over which data densification is desired

L - number corresponding to the Lth abscissa, the right-hand endpoint of the densification interval

M - number of interpolated values sought in the interior of each mesh interval (A+(I-1)*H,A+I*H)

* - corresponds to &STNO where STNO is a statement number in the calling program to which control returns in the event that an input error is detected

M1 - first derivative of a function at A (CSDDED only)

MN - first derivative of a function at B (CSDDED only)

OUTPUT. Output from CSDDE includes:

G - vector of (L-J)*(M+1)+1 equally spaced interpolated, function values in which G(I) is the value at A+(J-1)*H+(I-1)*(H/(M+1))

RESTRICTIONS. A .LT. B, N .GE. 2, 1 .LE. J .LT. L .LE. N, M .GE. 1. Detection of one of these errors results in an error message output on FILE 6 and a return to the statement number designated in the calling sequence. To change the output from FILE 6, replace the statement DATA PRINTR/6/. It is assumed that the dimension of F is at least N, and G is at least (L-J)*(M+1)+1.

METHOD. The method of cardinal spline interpolation is used for both the natural cubic spline (i.e., one having zero second derivatives at A and B) and the cubic spline with specified endpoint derivatives. Truncation of the sums appearing in the calculations is set to allow for 22 terms (adequate for almost all applications) but may be changed if desired by replacing the statement 1 IP22 = 22.

REFERENCES. Ahlberg, J. H. 1971. Spline Approximation as an Engineering Tool. In Computer-Aided Engineering:Proceedings of the Symposium Held at the University of Waterloo, May 11-13, 1971. SM Study No. 5. ed. G. M. L. Gladwell. Ontario, Canada:Solid Mechanics Division, University of Waterloo.

Nilson, E. N. 1970. Cubic Splines on Uniform Meshes. Numerical Mathematics 13:255-258.

CSIE SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE CSIE (A,B,N,F,M,G,*)
AND ENTRY CSIED (A,B,N,F,M,G,*,M1,MN)
REAL A,B,F,G,M1,MN
INTEGER N,M
DIMENSION F(N),G(M)
```

PURPOSE. Subroutine CSIE computes the natural cubic spline interpolate to a set of N equally spaced data and returns a vector of M interpolated function values. Entry CSIED computes the cubic spline interpolate to a set of N equally spaced data with specified endpoint first derivatives and returns a vector of M interpolated function values.

INPUT. Input to CSIED consists of:

A - left-hand endpoint of the interval (A,B) over which data is given

B - right-hand endpoint of the interval (A,B)

N - number of data points

F - vector of N equally spaced function values F(1),F(2), ...,F(N), where F(I) is the function value at A+(I-1)*H and H=(B-A)/(N-1)

M - number of interpolated values sought

G - vector of the M abscissae, in increasing order, at which interpolated function values are sought

* - corresponds to &STNO where STNO is a statement number in a calling program to which control returns in the event that an input error is detected

M1 - first derivative of function at A (CSIED only)

MN - first derivative of function at B (CSIED only)

OUTPUT. Output from CSIED includes:

G - vector of M interpolated function values corresponding to input abscissae

RESTRICTIONS. A .LT. B, N .GE. 2, M .GE. 1, A .LE. G(1), G(M) .LE. B. Detection of one of these errors results in an error message output on FILE 6 and a return to the statement number designated in the calling sequence. To change the output from FILE 6, replace the statement DATA PRINTR/6/. It is assumed that G(I) .LT. G(I+1) for I=1,...,M-1. The dimension of F is at least N and G is at least M.

METHOD. The method of cardinal spline interpolation is used for both the natural cubic spline (i.e., one having zero second derivatives at A and B) and the cubic spline with specified endpoint derivatives. Truncation of the sums appearing in the calculations is set to allow for 22 terms (adequate for almost all applications) but may be changed if desired by replacing the statement 1 IP22 = 22.

REFERENCES. Ahlberg, J. H. 1971. Spline Approximation as an Engineering Tool. In Computer-Aided Engineering:Proceedings of the Symposium Held at the University of Waterloo, May 11-13, 1971. SM Study No. 5. ed. G. M. L. Gladwell. Ontario, Canada:Solid Mechanics Division, University of Waterloo.

Nilson, E. N. 1970. Cubic Splines on Uniform Meshes. Numerical Mathematics 13:255-258.

CSDDU SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE CSDDU (N,X,F,J,M,G,WK,*)
AND ENTRY CSDDUD (N,X,F,J,M,G,WK,*,M1,MN)
REAL X,F,G,WK,M1,MN
INTEGER N,J,M
DIMENSION X(N),F(N),G(1),WK(N,3)
DIMENSION G(M+2)
```

PURPOSE. Subroutine CSDDU computes natural cubic spline interpolates to a set of N unequally spaced data and returns vectors of M+2 equally spaced interpolated function values. Entry CSDDUD computes cubic spline interpolates to a set of N unequally spaced data with specified endpoint first derivatives and returns a vector of M+2 equally spaced interpolated function values.

INPUT. Input to CSDDU consists of:

N - number of data points

X - vector of the N abscissae X(1),X(2),...,X(N) in increasing order

F - vector of N function values F(1),F(2),...,F(N) corresponding to X. F(I) is the function value at X(I).

J - number corresponding to the Jth abscissa, the left-hand endpoint of the interval over which data densification is desired

M - number of interpolated values sought in the interior of the interval (X(J),X(J+1))

WK - work area of a size at least 3*N

* - corresponds to &STNO where STNO is a statement number in the calling program to which control returns in the event that an input error is detected

M1 - first derivative of the function at A (CSDDUD only)

MN - first derivative of the function at B (CSDDUD only)

OUTPUT. Output from CSDDU includes:

G - vector of M+2 equally spaced interpolated function values where G(I) is the value at X(J)+(I-1)*((X(J+1)-X(J))/(M+1))

RESTRICTIONS. N .GE. 2, M .GE. 1, 1 .LE. J .LT. N. Detection of one of these errors results in an error message output on FILE 6 and a return to the statement number designated in the calling sequence. To change the output from FILE 6, replace the statement DATA PRINTR/6/. It is assumed that X(I) .LT. X(I+1) for I=1,...,N-1. The dimension of X is at least N, F at least N, G at least M+2, and WK at least 3*N.

METHOD. A natural cubic spline (i.e., one having zero second derivatives at X(1) and X(N)) or a cubic spline with specified endpoint derivates is computed by solving a tridiagonal system of N linear equations using the Thomas algorithm.

REFERENCE. Ahlberg, J. H.; Nilson, E. N.; and Walsh, J. L. 1967. The Theory of Splines and Their Applications. New York:Academic Press.

CSIU SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE CSIU (N,X,F,M,G,WK,*)
AND ENTRY CSIUD (N,X,F,M,G,WK,*,M1,MN)
REAL X,F,G,WK,M1,MN
INTERGER N,M
DIMENSION X(N),F(N),G(M),WK(N,3)
```

PURPOSE. Subroutine CSIU computes the natural cubic spline interpolate to the set of N unequally spaced data and returns a vector of M interpolated function values. Entry CSIUD computes the cubic spline interpolate to the set of N unequally spaced data with the specified endpoint first derivatives. It returns a vector of M interpolated function values.

INPUT. Input to CSIU consists of:

N - number of data points

X - vector of N abscissae X(1),X(2),...,X(N) in increasing order

F - vector of N function values F(1),F(2),...,F(N) corresponding to X. F(I) is a function value at X(I).

M - number of interpolated values sought

G - vector of M abscissae, in increasing order, at which interpolated function values are sought

WK - work area of a size at least 3*N

* - corresponds to &STNO where STNO is a statement number in the calling program to which control returns in the event that an input error is detected

M1 - first derivative of the function at A (CSIUD only)

MN - first derivative of the function at B (CSIUD only)

OUTPUT. Output from CSIU includes:

G - vector of M interpolated function values corresponding to the input abscissae

RESTRICTIONS. N .GE. 2, M .GE. 1, G(1) .GE. X(1), G(M) .LE. X(N). Detection of one of these errors results in an error message output on FILE 6 and a return to the statement number designated in the calling sequence. To change the output from FILE 6, replace the statement DATA PRINTR/6/. It is assumed that G(I) .LT. G(I+1) for I=1,...,M-1 and X(I) .LT. X(I+1) for I=1,...,N-1. The dimension of X is at least N, F at least N, G at least M, and WK at least 3*N.

METHOD. A natural cubic spline (i.e., one having zero second derivatives at X(1) and X(N)) or a cubic spline with specified end-point derivates is computed by solving a tridiagonal system of N linear equations using the Thomas algorithm.

REFERENCE. Ahlberg, J. H.; Nilson, E. N.; and Walsh, J. L. 1967. The Theory of Splines and Their Applications. New York:Academic Press.

CSDDE2 SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE CSDDE2 (A,B,C,D,NX,NY,F,JX,JY,IMX,IMY,G,WK,IRWF,IRWG,*)
REAL A,B,C,D,F,G,WK
INTEGER NX,NY,JX,JY,IMX,IMY,IRWF,IRWG
DIMENSION F(IRWF,NY),G(IRWG,IMY),WK(1)
DIMENSION WK(NY*(IMX+1)+IMY)
```

PURPOSE. CSDDE2 computes the natural bicubic spline interpolate (2-dimensional) to the set of NX*NY data points on a rectangular grid with equal (but not necessarily the same) spacing in both X and Y directions. It returns an IMX*IMY array of interpolated function values.

INPUT. Input to CSDDE2 consists of:

A - left-hand endpoint of the interval (A,B) on the X axis. The data is given on a rectangle (A,B) x (C,D).

B - right-hand endpoint of the interval (A,B)

C - lower limit of the interval (C,D) on the Y axis

D - upper limit of the interval (C,D)

NX - number of mesh points in the X direction

NY - number of mesh points in the Y direction

F - array of the function values F(1,1),F(2,1),...,F(NX,1), F(1,2),...,F(NX,NY) where F(I,J) is a function value at (A+(I-1)*HX,C+(J-1)*HY) and HX=(B-A)/(NX-1), HY=(D-C)/(NY-1)

JX - number corresponding to the JXth abscissa in the X direction. Data densification occurs over the rectangle (A+(JX-1)*HX,A+JX*HX) x (C+(JY-1)*HY,C+JY*HY)

JY - number corresponding to the JYth ordinate in the Y direction

IMX - number of interpolated values sought in the interior of the interval in the X direction for each value of Y + 2 (n.b.)

IMY - number of interpolated values sought in the interior of the interval in the Y direction for each value of X + 2 (n.b.)

WK - work area of a size at least NY*(IMX+1)+IMY

IRWF - actual number of rows in F in the calling program

IRWG - actual number of rows in G in the calling program

* - corresponds to &STNO where STNO is a statement number in the

calling program to which control returns in the event that an input error is detected

OUTPUT. Output from CSDDE2 includes:

G - array of IMX*IMY interpolated function values
G(1,1),G(2,1),...,G(IMX,1),G(1,2),...,G(IMX,IMY)
where G(I,J) is the function value at
(A+(JX-1)*HX+(I-1)*(HX/(IMX-1)),
C+(JY-1)*HY+(J-1)*(HY/(IMY-1)))

SUBROUTINES REQUIRED. The necessary subroutine is CSDDE.

RESTRICTIONS. A .LT. B, C .LT. D, NX .GE. 2, NY .GE. 2, IMX .GE. 3, IMY .GE. 3, 1 .LE. JX .LT. NX, 1 .LE. JY .LT. NY, NX .LE. IRWF, IMX .LE. IRWG. Detection of one of these errors results in an error message output on FILE 6 and a return to the statement number designated in the calling sequence. To change the output from FILE 6, replace the statement DATA PRINTR/6/. It is assumed that the dimension of F(IRWF, is at least NY), G(IRWG, at least IMY), and WK at least NY*(IMX+1)+IMY.

METHOD. The 1-dimensional natural cubic spline interpolates in X are found interpolating to the functions F(I,J), I=1,...,NX for each value of J=1,...,NY. These splines are all evaluated at appropriate X value(s) and the results used as data for a 1-dimensional natural cubic spline interpolation in the Y direction.

REFERENCE. Ahlberg, J. H.; Nilson, E. N.; and Walsh, J. L. 1967. The Theory of Splines and Their Applications. New York:Academic Press.

CSIE2 SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE CSIE2 (A,B,C,D,NX,NY,F,XO,M,Y,WK,IRWF,*)
REAL A,B,C,D,F,XO,Y,WK
INTEGER NX,NY,M,IRWF
DIMENSION F(IRWF,NY),WK(NY),Y(M)
```

PURPOSE. CSIE2 computes the natural bicubic spline interpolate (2-dimensional) to a set of NX*NY data points on a rectangular grid with equal (but not necessarily the same) spacing in both X and Y directions. It returns a vector of M interpolated function values.

INPUT. Input to CSIE2 consists of:

A - left-hand endpoint of the interval (A,B) on the X axis. Data is given on a rectangle (A,B) x (C,D).

B - right-hand endpoint of the interval (A,B)

C - lower limit of the interval (C,D) on the Y axis

D - upper limit of the interval (C,D)

NX - number of mesh points in the X direction

NY - number of mesh points in the Y direction

F - array of the function values F(1,1),F(2,1),...,F(NX,1), F(1,2),...,F(NX,NY) where F(I,J) is a function value at (A+(I-1)*HX,C+(J-1)*HY) and HX=(P-A)/(NX-1), HY=(D-C)/(NY-1)

XO - X direction abscissa for which interpolates are sought

M - number of interpolated values sought

Y - vector of M ordinates in a Y direction, in increasing order, for which interpolates are sought

WK - work area of a size at least NY

IRWF - actual number of rows in F in the calling program

* - corresponds to &STNO where STNO is a statement number in the calling program to which control returns in the event that an input error is detected

OUTPUT. Output from CSIE2 includes:

Y - vector of M interpolated function values where Y(I) is the function value at (XO,YIN(I)). YIN(I) means the value of Y(I) on input.

SUBROUTINES REQUIRED. The necessary subroutine is CSIE.

RESTRICTIONS. A .LT. B, C .LT. D, NX .GE. 2, NY .GE. 2, M .GE. 1, NX .LE. IRWF, A .LE. XO .LE. B, Y(1) .GE. C, Y(M) .LE. D. Detection of one of these errors results in an error message output on FILE 6 and a return to the statement number designated in the calling sequence. To change the output from FILE 6, replace the statement DATA PRINTR/6/. It is assumed that Y(I) .LT. Y(I+1) for I=1,...,M-1. The dimension of F(IRWF, is at least NY), Y at least M, and WK at least NY.

METHOD. The 1-dimensional natural cubic spline interpolates in X are found interpolating to the functions F(I,J), I=1,...,NX for each value of J=1,...,NY. These splines are all evaluated at appropriate X value(s) and the results used as data for a 1-dimensional natural cubic spline interpolation in the Y direction.

REMARKS. If XO is a mesh point (i.e., XO=X(I) for some I), the corresponding 1-dimensional interpolation routine CSIE yields the same results when applied to the function F(I,J),J=1,...,NY where X(I)=XO.

REFERENCE. Ahlberg, J. H.; Nilson, E. N.; and Walsh, J. L. 1967. The Theory of Splines and Their Applications. New York:Academic Press.

CSDDU2 SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE CSDDU2 (NX,X,NY,Y,F,JX,JY,IMX,IMY,G,WK,IRWF,IRWG,*)
REAL X,Y,F,G,WK
INTEGER NX,NY,JX,JY,IMX,IMY,IRWF,IRWG
DIMENSION F(IRWF,NY),G(IRWG,IMY),WK(1),X(NX),Y(NY)
DIMENSION WK(NY*(IMX+1)+IMY+3*MAX(NX,NY))
```

PURPOSE. CSDDU2 computes the natural bicubic spline interpolate (2-dimensional) to a set of NX*NY data points on a rectangular grid with unequal spacing. It returns an IMX*IMY array of interpolated function values.

INPUT. Input to CSDDU2 consists of:

NX - number of mesh points in the X direction

X - vector of NX abscissae in the X direction

NY - number of mesh points in the Y direction

Y - vector of NY ordinates in the Y direction

F - array of the function values F(1,1),F(2,1),...,F(NX,1), F(1,2),...,F(NX,NY) where F(I,J) is a function value at (X(I),Y(J))

JX - number corresponding to the JXth abscissa in the X direction. Data densification occurs over the rectangle (X(JX),X(JX+1)) x (Y(JY),Y(JY+1))

JY - number corresponding to the JYth ordinate in the Y direction

IMX - number of interpolated values sought in the interior of the interval in the X direction for each value of Y + 2 (n.b.)

IMY - number of interpolated values sought in the interior of the interval in the Y direction for each value of X + 2 (n.b.)

WK - work area of the size at least NY*(IMX+1)+IMY+3*MAX(NX,NY)

IRWF - actual number of rows in F in the calling program

IRWG - actual number of rows in G in the calling program

* - corresponds to &STNO where STNO is a statement number in the calling program to which control returns in the event that an input error is detected

OUTPUT. Output from CSDDU2 includes:

G - array of IMX*IMY interpolated function values G(1,1),G(2,1),...,G(IMX,1),G(1,2),...,G(IMX,IMY)

where G(I,J) is the function value at
(X(JX)+(I-1)*((X(JX+1)-X(JX))/(IMX-1)),
Y(JY)+(J-1)*((Y(JY+1)-Y(JY))/(IMY-1)))

SUBROUTINES REQUIRED. The necessary subroutine is CSDDU.

RESTRICTIONS. NX .GE. 2, NY .GE. 2, 1 .LE. JX .LT. NX, 1 .LE. JY .LT. NY, IMX .GE. 3, IMY .GE. 3, NX .LE. IRWF, IMX .LE. IRWG. Detection of one of these errors results in an error message output on FILE 6 and a return to the statement number designated in the calling sequence. To change the output from FILE 6, replace the statement DATA PRINTR/6/. It is assumed that X(I) .LT. X(I+1) for I=1,...,NX-1 and Y(I) .LT. Y(I+1) for I=1,...,NY-1. The dimension of X is at least NX, Y at least NY, F(IRWF, at least NY), G(IRWG, at least IMY), and WK at least NY*(IMX+1)+IMY+3*MAX(NX,NY).

METHOD. The 1-dimensional natural cubic spline interpolates in X are found interpolating to the functions F(I,J), I=1,...,NX for each value of J=1,...,NY. These splines are all evaluated at appropriate X value(s) and the results used as data for a 1-dimensional natural cubic spline interpolation in the Y direction.

REFERENCE. Ahlberg, J. H.; Nilson, E. N.; and Walsh, J. L. 1967. The Theory of Splines and Their Applications. New York:Academic Press.

CSIU2 SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE CSIU2 (NX,X,NY,Y,F,XO,M,Y1,WK,IRWF,*)
REAL X,Y,F,XO,Y1,WK
INTEGER NX,NY,M,IRWF
DIMENSION F(IRWF,NY),WK(1),X(NX),Y(NY),Y1(M)
DIMENSION WK(NY+3*MAX(NX,NY))
```

PURPOSE. CSIU2 computes the natural bicubic spline interpolate (2-dimensional) to a set of NX*NY data points on a rectangular grid with unequal spacing and returns a vector of M interpolated function values.

INPUT. Input to CSIU2 consists of:

NX - number of mesh points in the X direction

X - vector of NX abscissae in the X direction

NY - number of mesh points in the Y direction

Y - vector of NY ordinates in the Y direction

F - array of the function values F(1,1),F(2,1),...,F(NX,1), F(1,2),...,F(NX,NY) where F(I,J) is a function value at (X(I),Y(J))

XO - X direction abscissa for which interpolates are sought

M - number of interpolated values sought

Y1 - vector of M ordinates in the Y direction, in increasing order, for which interpolates are sought

WK - work area of the size at least NY+3*MAX(NX,NY)

IRWF - actual number of rows in F in the calling program

* - corresponds to &STNO where STNO is a statement number in the calling program to which control returns in the event that an input error is detected

OUTPUT. Output from CSIU2 includes:

Y1 - vector of M interpolated function values where Y1(I) is a function value at (XO,Y1IN(I)), and where Y1IN(I) means the value of Y1(I) on input

SUBROUTINES REQUIRED. The necessary subroutine is CSIU.

RESTRICTIONS. NX .GE. 2, NY .GE. 2, X(1) .LE. XO .LE. X(NX), M .GE. 1, NX .LE. IRWF, Y(1) .LE. Y1(1), Y(NY) .GE. Y1(M). Detection of one of these errors results in an error message output on FILE 6

and a return to the statement number designated in the calling sequence. To change the output from FILE 6, replace the statement DATA PRINTR/6/. It is assumed that X(I) .LT. X(I+1) for I=1,...,NX-1, Y(I) .LT. Y(I+1) for I=1,...,NY-1 and Y1(I) .LT. Y1(I+1) for I=1,... ,M-1. The dimension of X is at least NX, Y at least NY, F(IRWF, at least NY), Y1 at least M, and WK at least NY+3*MAX(NX,NY).

METHOD. The 1-dimensional natural cubic spline interpolates in X are found interpolating to the functions F(I,J), I=1,...,NX for each value of J=1,...,NY. These splines are all evaluated at appropriate X values(s) and the results used as data for a 1-dimensional natural cubic spline interpolation in the Y direction.

REMARKS. If XO is a mesh point (i.e., XO=X(I) for some I), the corresponding 1-dimensional interpolation routine CSIU yields the same results when applied to the function F(I,J),J=1,...,NY where X(I)=XO.

REFERENCE. Ahlberg, J. H.; Nilson, E. N.; and Walsh, J. L. 1967. The Theory of Splines and Their Applications. New York:Academic Press.

SECTION 10

DIFFERENTIAL EQUATIONS, NUMERICAL INTEGRATION, AND DIFFERENTIATION

DIFFERENTIAL EQUATIONS.

The DESUB package for solving systems of first order ordinary differential equations (documented below) is the FORTRAN version of the ALGOL procedure DIFEQNS given in section 3. It uses the rational extrapolation method of R. Bulirsch and J. Stoer. The FORTRAN implementation was developed at Bell Telephone Laboratories and made available by Bell. Extensive testing demonstrates that it is fast and accurate. (For more detailed information see section 3.) The numerical integration and differentiation routines NUMINT and NUMDIF use the spline interpolation described in section 9. Both accept unequally spaced data.

DESP SUBROUTINE.

The syntax is as follows:

```
SUBROUTINE DESP (SP,XFX,N,Y,XI,XF,HI,EPS,ERR,XOUTX,*)
AND ENTRY DE(XFX,N,Y,XI,XF,HI,EPS,ERR,XOUTX,*)
DIMENSION Y(N),DY(20),S(20),R(20),YR(20)
COMMON /PARAM/ZOT,POWER2,EMAX,EMIN,HDIV,ZOTUP,ICHOP
COMMON /IPARAM/M,NMAX
COMMON /INFO/ EX,ER,EH,NE,NERR
COMMON /OTPUT/ SPPRT,HIPRT,XIPRT,XFPRT,EPSPRT,NPPRT,TITLE
LOGICAL STYPE,KONVF,TITLE
EXTERNAL XFX,ERR
```

INPUT. Input to DESP consists of:

SP - input for DESP only. A positive value for SP causes the solution to be computed and output at each point of grid with the spacing SP. Use entry DE when the program-selected output is sufficient.

XFX - user supplied subroutine to calculate the right-hand side of the differential equation. It must have the form:

```
SUBROUTINE XFX(Y,X,DY)
REAL       X
DIMENSION  Y(20),DY(20)
```

Where given the values of X and the dependent variables Y(1),...,Y(N) at X, the subroutine returns the values of the derivatives DY(1),...,DY(N) at X.

N - number of equations in the system

Y - a 1-dimensional array of initial values

XI - initial value of the independent variable

XF - final value of the independent variable. XF may be less

than XI, in which case HI must be negative.

HI - initial stepsize. HI may be negative.

EPS - error tolerance

ERR - parameter to specify the convergence criterion. There are three choices for ERR:

ERR = SE - Standard error
ERR = RE - Relative error
ERR = AE - Absolute error

XOUTX - output subroutine. A printer output routine named DEOUT is supplied with the DESUB package. Alternatively, the user may supply his own routine.

* - in call to DESP, an * must be replaced by &STNO, where STNO is a label in the calling program to which control is transferred in the event of an input error or a failure of the iteration to converge

OUTPUT. Output from DESP includes:

Y - contains the solution at XF. Output at intermediate points is under the control of the subroutine XOUTX.

SUBROUTINES REQUIRED. The necessary subroutines are XFX, ERR, and XOUTX. The latter two are supplied with the DESUB package but may be replaced by user-coded routines. XFX must be supplied by the user.

RESTRICTIONS. N .LE. 20. For larger systems, the value of NMAX is in the Block Data section. The dimensions of the local arrays declared in subroutines DESUB and DESP must be increased. However, DESUB may not be as efficient as predictor-corrector methods for large systems. The input parameters HI and EPS must satisfy:
ABS(XF-XI)/2**15 .LT. ABS(HI), 1.0E-10 .LE. EPS .LE. 1.0E-2.

METHOD. DESP uses the rational extrapolation method of R. Burlirsch and J. Stoer. Extensive testing at Bell Telephone Laboratories has shown this program to be superior in both speed and accuracy to the older standard methods, at least for small systems of equations.

REFERENCE. Bulirsch, R. and Stoer, J. 1966. Numerical Treatment of Ordinary Differential Equations by Extrapolation Methods. Numerische Mathematik 8:1-13.

ACKNOWLEDGEMENT. The DESUB package is Vol. 2, Issue 1 of the Numerical Mathematics Program Library Project, Bell Telephone Laboratories, Murray Hill, New Jersey, and is used with their permission.

NUMERICAL INTEGRATION AND DIFFERENTIATION.

The function F is given in tabular form at unequally spaced abscissae, X(1),X(2),...,X(N), where F(I) is the function value at X(I). The routines in this section calculate an approximate integral of F over (X(1),X(N)) and evaluate approximate first derivatives of F at specified points in the interval (X(1),X(N)).

A cubic spline of interpolation to F on the mesh X(1),...,X(N) with endpoint derivatives specified as the first divided differences (F(2)-F(1))/(X(2)-X(1)) and (F(N)-F(N-1))/(X(N)-X(N-1)) is calculated. (See the section on interpolation with cubic splines for a discussion of the background and method.) NUMINT then calculates the integral of the spline interpolant as the approximate integral of F and NUMDIF evaluates the first derivatives of the spline at the desired points as the approximate first derivatives of F. A useful method for calculating approximate second derivatives of F is to calculate approximate first derivatives using NUMDIF at, for example, the mesh points X(1),...,X(N), and then to use NUMDIF again with these approximate first derivatives as the input function values.

REFERENCE. Ahlberg, J. H.; Nilson, E. N.; and Walsh, J. L. 1967. The Theory of Splines and Their Applications. New York:Academic Press.

Greville, T. N. E. 1967. Spline Functions, Interpolation, and Numerical Quadrature. Mathematical Methods for Digital Computers, Vol. 2. New York:John Wiley and Sons.

NUMINT SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE NUMINT (N,X,F,INTEG,WK)
REAL X,F,INTEG,WK
INTEGER N
DIMENSION X(N),F(N),WK(N,3)
```

PURPOSE. The subroutine NUMINT calculates an approximate integral over (X(1),X(N)) for a function F given in tabular form with unequally spaced data.

INPUT. Input to NUMINT consists of:

N - number of data points

X - vector of the N abscissae X(1),X(2),...,X(N) in increasing order

F - vector of the N function values F(1),F(2),...,F(N) corresponding to X. F(I) is the function value at X(I).

WK - work area of the size at least 3*N

OUTPUT. Output from NUMINT includes:

INTEG - approximate value of the integral of F over the interval (X(1),X(N))

RESTRICTIONS. It is assumed that X(I) .LT. X(I+1) for I=1,...,N-1. The dimension of X is at least N, F at least N, and WK at least 3*N.

METHOD. A cubic spline of interpolation to the function F on the mesh X(1),...,X(N) with specified endpoint derivatives is calculated, and its integral over (X(1),X(N)) evaluated. The endpoint derivatives are assumed to be the first divided differences (F(2)-F(1))/(X(2)-X(1)) and (F(N)-F(N-1))/(X(N)-X(N-1)).

REFERENCES. Ahlberg, J. H.; Nilson, E. N.; and Walsh, J. L. 1967. The Theory of Splines and Their Applications. New York:Academic Press.

Greville, T. N. E. 1967. Spline Functions, Interpolation, and Numerical Quadrature. Mathematical Methods for Digital Computers, Vol. 2. New York:John Wiley and Sons.

NUMDIF SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE NUMDIF (N,X,F,M,G,WK)
REAL X,F,G,WK
INTEGER N,M
DIMENSION X(N),F(N),G(M),WK(N,3)
```

PURPOSE. The subroutine NUMDIF calculates approximate first derivatives for a function given in tabular form with unequally spaced data.

INPUT. Input to NUMDIF consists of:

N - number of data points

X - vector of N abscissae X(1),X(2),...,X(N) in increasing order

F - vector of the N function values F(1),F(2),...,F(N) corresponding to X. F(I) is the function value at X(I).

M - number of first derivatives sought

G - vector of M abscissae, in increasing order, at which derivatives are sought

WK - work area of the size at least 3*N

OUTPUT. Output from NUMDIF includes:

G - vector of M first derivatives corresponding to input abscissae

RESTRICTIONS. It is assumed that X(I) .LT. X(I+1) for I=1,...,N-1, G(I) .LT. G(I+1) for I=1,...,M-1, G(1) .GE. X(1), G(M) .LE. X(N). The dimension of X is at least N, F at least N, G at least M, and WK at least 3*N.

METHOD. A cubic spline of interpolation to the function F on the mesh X(1),...,X(N) with specified endpoint derivatives is calculated. Its derivatives are evaluated at the desired points. The endpoint derivatives are assumed to be the first divided differences (F(2)-F(1))/(X(2)-X(1)) and (F(N)-F(N-1))/(X(N)-X(N-1)).

REFERENCE. Ahlberg, J. H.; Nilson, E. N.; and Walsh, J. L. 1967. The Theory of Splines and Their Applications. New York:Academic Press.

SECTION 11

ZEROES AND EXTREMA

SOLUTION OF NONLINEAR EQUATIONS.

NLNEQN SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE NLNEQN(N,X,WK,STOPIT,EPS,ERS,FREQ)
INTEGER N,STOPIT,FREQ
DIMENSION X(N),WK(1)
REAL EPS,ERS
```

PURPOSE. NLNEQN solves a system of N simultaneous nonlinear equations in N unknowns.

REQUIREMENT. A subroutine bearing the name FUNCT must be supplied by the user. It must have the form:

```
SUBROUTINE FUNCT(X,I,F)
INTEGER I
REAL F,X(I)
```

It returns in F the value of the Ith function of X.

INPUT. Input to NLNEQN consists of:

N - number of equations and variables in the system

X - array of the size N containing the starting values of N variables

WK - array of the minimum size N*(N-(N-1)/2)

STOPIT - maximum allowed number of iterations

EPS - convergence criterion on the functions. An acceptable solution is at hand if for I = 1,...,N ABS(F(I)) < EPS.

ERS - second stopping criterion on the relative distance between two successive approximations

FREQ - output option which equals either the interval between iterations for printout, or 0 if no printout is required

OUTPUT. Output from NLNEQN includes:

X - approximation to the solution unless STOPIT is exceeded or the modified Jacobian becomes singular. In these cases X forms the last iterate.

STOPIT - actual number of iterations performed

SUBROUTINES REQUIRED. FUNCT must be provided by the user.

RESTRICTIONS. The number of equations are limited to 10. Larger systems may be accommodated by enlarging the arrays.

METHOD. The algorithm used is a modification of Newton's developed by Brown. Each equation is expanded in a Taylor series and the variable corresponding to the largest derivative is eliminated. Extensive testing shows Brown's algorithm to be the best general purpose equation solver currently available.

REMARKS. For maximum efficiency the functions in FUNCT must be ordered according to increasing nonlinearity.

REFERENCE. Brown, K. M. 1969. Quadratically Convergent Newton-Like Method Based upon Gaussian Elimination. SIAM J. Numer. Anal. 6:560-569.

ZEROES OF FUNCTIONS OF A SINGLE VARIABLE.

ZEROML SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE ZEROML (FUNCT,N,V,FV,ITNO,EPSV,EPSD,FREQ)
INTEGER N,FREQ,ITNO(N)
REAL V(N),FV(N),EPSV,EPSD
```

PURPOSE. ZEROML uses Muller's method to find real zeroes of a real function F(X) of a single variable.

INPUT. Input to ZEROML consists of:

FUNCT - a user-provided external function which returns the value of F at X. It must have the form:

```
FUNCTION FUNCT (X)
REAL X
```

N - an integer variable specifying the number of starting points contained in V

V - a 1-dimensional array of the size N containing the N starting values

ITNO - a 1-dimensional array of the size N specifying the maximum iterations allowed for each of the N starting points

EPSV - a real variable specifying the first stopping criterion. V(I) is an acceptable zero if ABS(F(V(I))) is less than EPSV.

EPSD - a real variable specifying the second stopping criterion. The iterative process is terminated if the relative distance between two successive approximations is less than EPSD.

FREQ - an integer variable specifying the frequency of output in terms of numbers of iterations between printing results. A value of 0 suppresses printing.

OUTPUT. Output from ZEROML includes:

V - array containing the N computed approximations to zeroes

FV - a 1-dimensional array of the size N containing the N values of F at V

ITNO - array containing the number of iterations performed. On an abnormal termination ITNO(I) contains:

0 - if the function F is a constant

-1 - if the iteration is stopped by exceeding the number of iterations allowed.

-2 - if the iteration is stopped by the second stopping criterion

METHOD. Muller's method of parabolic interpolation is used. Deflation is performed to prevent approximations to converge to previously computed zeroes.

REMARKS. The starting points contained in V should be arranged in monotonically increasing order to minimize the error caused by deflation.

REFERENCES. Muller, D. E. 1956. A Method for Solving Algebraic Equations Using an Automatic Computer. Math. Tables Aids Comput. 10:205-208.

Wilkinson, J. H. 1963. Rounding Errors in Algebraic Processes. New York:Prentice-Hall.

ZEROSS SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE ZEROSS (FUNCT,N,V,ITNO,EPS,FREQ)
INTEGER N,FREQ,ITNO(N)
REAL V(N),ERS
```

PURPOSE. ZEROSS uses the successive substitution method to find the real roots of an equation of the form G(X) = X for a real function G.

INPUT. Input to ZEROSS consists of:

FUNCT - a user-provided external function which returns the value of G at X. It must have the form:

```
FUNCTION FUNCT (X)
REAL X
```

N - an integer variable specifying the number of starting points contained in V

V - a 1-dimensional array of the size N containing the N starting values

ITNO - a 1-dimensional array of the size N specifying the maximum permissible iterations for each starting point

EPS - a real value specifying the stopping criterion. V(I) is an acceptable root if the distance between the two successive approximations is less than EPS.

FREQ - an integer number specifying the frequency of output in terms of the number of iterations between printing results. A value of 0 suppresses output.

OUTPUT. Output from ZEROSS includes:

V - array containing the N computed roots

ITNO - array containing the number of iterations performed. If convergence is not achieved within the number of iterations allowed, a value of -1 is returned.

METHOD. Successive iterates in this method are given by:

$$X(K+1) = G(X(K)) \quad K=0, 1, 2, \ldots$$

Aitken's acceleration method is used to assure fast convergence.

REMARKS. In addition to solving equations of the form G(X) = X, ZEROSS provides a flexible skeleton program for various iterative methods used to solve equations of the form F(X) = 0. For example, in Newton's method without deflation, G should be of the form

G = X - F(X)/F'(X).

REFERENCES. Isaacson, I. and Keller, H. B. 1966. Analysis of Numerical Methods. New York:John Wiley and Sons.

Wilkinson, J. H. 1963. Rounding Errors in Algebraic Processes. New York:Prentice-Hall.

ZERONT SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE ZERONT (FUNCT,N,V,FV,ITNO,EPSV,EPSD,FREQ)
INTEGER N,FREQ,ITNO(N)
DOUBLE PRECISION V(N), FV(N), EPSV,EPSD
```

PURPOSE. ZERONT uses Newton's method to find the real zeroes of a real function F(X) of a single variable.

INPUT. Input to ZERONT consists of:

FUNCT - a user-provided, external, double precision function which returns the value of F at X. It has the form:

```
DOUBLE PRECISION FUNCTION FUNCT(X)
DOUBLE PRECISION X
```

N - an integer variable specifying the number of starting points contained in V

V - a 1-dimensional array of the size N containing the N starting values

ITNO - a 1-dimensional array of maximum permissible iterations for each of the N starting points

EPSV - a double precision variable containing the first stopping criterion. V(I) is an acceptable zero if ABS (F(V(I))) is less than EPSV.

EPSD - a double precision variable containing the second stopping criterion. The iterative process is terminated if the relative distance between two successive approximations is less than EPSD.

FREQ - an integer variable containing the frequency of output in terms of the number of iterations between printing results. A value of 0 suppresses printing.

OUTPUT. Output from ZERONT includes:

V - array containing the N approximations of zeroes

FV - a 1-dimensional array of the size N containing the N values of F at V

ITNO - array containing the number of iterations performed to arrive at the approximations in V. On abnormal termination, ITNO(I) contains:

0 - if function F is a constant

-1 - if the iteration is stopped by exceeding the number of iterations allowed

-2 - if the iteration is stopped by the second stopping criterion

METHOD. ZERONT uses Newton's method with acceleration to assure fast convergence. Deflation of the function is used to prevent approximations of convergence to previously computed zeroes.

REMARKS. Newton's method requires a starting point to be close to zero. It might be necessary to try several starting points to ensure convergence to the true zero. Derivatives needed in Newton's method are generated within ZERONT. Initial starting points in V should be arranged in a monotonically increasing order to minimize error due to deflation. Double precision is used to allow higher accuracy.

REFERENCES. Isaacson, I. and Keller, H. B. 1966. Analysis of Numerical Methods. New York:John Wiley and Sons.

Wilkinson, J. H. 1963. Rounding Errors In Algebraic Processes. New York:Prentice-Hall.

EXTREMA OF FUNCTIONS.

OPTHKJ SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE OPTHKJ(FUNCT,N,X,DX,BL,BU,EPS,ITNO,FREQ,Y,FY)
INTEGER N,ITNO,FREQ
REAL X(N),DX(N),BL(N),BU(N),EPS,Y(N),FY
```

PURPOSE. OPTHKJ uses the Hooke-Jeeves direct search method to solve optimization problems of an N-dimensional, nonlinear function F(X). Upper and lower bounds on independent variables can be specified.

INPUT. Input to OPTHKJ consists of:

FUNCT - a user-provided external function which should return the value of F at X. It is of the form:

```
FUNCTION FUNCT(X)
REAL X(1)
```

N - specifies the number of independent variables of F

X - 1-dimensional array of the size N that contains the starting vector

DX - 1-dimensional array of the size N that contains the initial step-size vector

BL - 1-dimensional array of the size N that specifies the lower bound of the solution vector. Sufficiently large negative numbers (e.g., -0.1E+10) should be entered if no lower bound is to be set.

BU - 1-dimensional array of the size N that specifies the upper bound of the solution vector. Sufficiently large positive numbers (e.g., +0.1E+10) should be entered if no upper bound is to be set.

EPS - specifies the stopping criterion. Iteration terminates if the norm of the reduced DX is less than EPS.

ITNO - specifies the maximum number of iterations allowed

FREQ - specifies the frequency of output in terms of the number of iterations between printing results. A value of 0 suppresses output.

OUTPUT. Output from OPTHKJ includes:

ITNO - equals either the number of iterations performed, or -1 if the maximum number of iterations allowed is exceeded

Y - 1-dimensional array of the size N which contains the approximate solution vector found, unless ITNO is -1

FY - the function value of F at Y

METHOD. OPTHKJ uses the Hooke-Jeeves direct search method, which is an algorithm without using derivatives. Exploratory and pattern searches are conducted alternatively to find better approximations. DX, the step size, is reduced when both of the searches fail, and iteration terminates when its norm is less than the EPS specified.

REMARKS. In general, OPTHKJ finds a local optimum. Consequently, to assure optimality, the user should try to start with several different initial vectors.

REFERENCE. Hooke, R. and Jeeves, T. A. 1962. Direct Search Solutions of Numerical and Statistical Problems. Journal of ACM. 8:212-229.

NONLINEAR PROGRAMMING.

PENHKJ SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE PENHKJ(FUNCT,N,L,M,X,BL,BU,EPSP,EPSD,ITNO,FREQ,Y,FY)
INTEGER N,L,M,ITNO,FREQ
REAL EPSP,EPSD,X(N),BL(N),BU(N),Y(N),FY(L+M+1)
```

PURPOSE. PENHKJ finds the optimal solution of a constrained nonlinear programming problem of the form:

$$\underset{(BL \;.LE.\; X \;.LE.\; BU)}{MIN} \quad F(X)$$

$$\text{SUBJECT TO} \quad G_I(X) \;.LE.\; 0,\ I=1,2.....L,$$

$$H_I(X) = 0,\ I=1,2.....M.$$

The penalty function method is used with the intermediate unconstrained optimization problems solved by the Hooke-Jeeves method.

INPUT. Input to PENHKJ consists of:

FUNCT - a user-provided external function of the form:

```
FUNCTION FUNCT(X,I)
REAL X
INTEGER I
```

L - specifies the number of inequality constraints
FUNCT should return the value:

$G_I(X)$, IF 1 .LE. I .LE. L,

$H_{I-L}(X)$, IF (L+1) .LE. I .LE. (L+M)

$F(X)$, IF I=L+M+1

N - specifies the number of independent variables of F

L - specifies the number of inequality constraints

M - specifies the number of equality constraints

X - 1-dimensional array of the size N that contains the starting vector

BL - 1-dimensional array of the size N that specifies the lower bound of the solution vector. Sufficiently large negative numbers (e.g., -0.1E+10) should be entered if no lower bound is set.

BU - 1-dimensional array of the size N that specifies the upper

bound of the solution vector. Sufficiently large positive numbers (e.g., +0.1E+10) should be entered if no upper bound is set.

EPSP - specifies the stopping criterion. Iteration terminates and X is an acceptable approximation if P(X), the penalty at vector X, is less than EPSP.

EPSD - specifies the second stopping criterion. The iterative process is terminated if the relative distance between two successive approximations is less than EPSD.

ITNO - specifies the maximum number of iterations allowed

FREQ - specifies the frequency of output in terms of the number of iterations between printing results. A value of 0 suppresses output.

VY - 1-dimensional array of the size (N+M+L+1), reserved at a working area

OUTPUT. Output from PENHKJ includes:

ITNO - contains the number of iterations performed. On an abnormal termination, it equals either -1 exceeding the number of allowed iterations if the maximum number of iterations is exceeded, or -2 if iteration is stopped by a second stopping criterion.

Y - 1-dimensional array of the size N that contains the approximate solution vector, unless ITNO .LE. 0

VY - 1-dimensional array of the size (N+M+L+1). FY(I) contains:

$G_I(Y)$, IF 1 .LE. I .LE. L,

$H_{I-L}(Y)$, IF (L+1) .LE. I .LE. (L+M)

F(Y), IF I=L+M+1

EPSP - contains P(Y), the penalty at the approximate solution Y

METHOD. PENHKJ uses the well known penalty function method. It transforms a constrained mathematical programming problem of the form used in PURPOSE (above) into a sequence of unconstrained optimization problems of the form:

$$\underset{(BL < X < BU)}{MIN} \quad Z_K(X) = F(X) + W_K * P_K(X), \quad K=0,1,2,\ldots$$

W's (the penalty coefficients) are a monotonically increasing sequence. P(X)'s, (the penalties at X) are measures of the extent to which the constraints are violated. In PENHKJ, a scaling technique is used to

avoid domination of P(X) by any one or group of constraints. The Hooke-Jeeves method is used to solve the sequence of optimization problems of the form used immediately above.

REMARKS. To assure optimality, the user should try to start with several different initial vectors.

REFERENCES. Fiacco, A. V. and McCormick, G. P. 1968. Nonlinear Programming:Sequential Unconstrained Minimization Techniques. New York:John Wiley and Sons.

Hooke, R. and Jeeves, T. A. 1962. Direct Search Solutions of Numerical and Statistical Problems. Journal of ACM 8:212-229.

Keefer, D. L. and Gottfried, B. S. 1970. Differential Constraint Scaling in Penalty Function Optimization. AIIE Transactions 2:281-289.

SECTION 12

FORTRAN POLYNOMIALS

ROOTS OF POLYNOMIALS.

RLPOLY SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE RLPOLY(NDEG,PP,ZEROR,ZEROI,FAIL)
DIMENSION ZEROR(NDEG),ZEROI(NDEG),PP(1)
LOGICAL FAIL
```

PURPOSE. RLPOLY finds all zeroes (rocts) of a real polynomial PP = A(1)*X**NDEG+...+A(NDEG)*X+A(NDEG+1).

INPUT. Input to RLPOLY consists of:

NDEG - degree of the polynomial

PP - array of coefficients of the polynomial ordered from largest to smallest power; remains intact

OUTPUT. Output from RLPOLY consists of:

ZEROR, ZEROI - arrays of the real and imaginary parts of the zeroes of the polynomial

FAIL - Boolean value set TRUE if algorithm fails to find all zeroes; otherwise FALSE

RESTRICTIONS. The algorithm accepts polynomials with a maximum degree 50 (1 .LE. NDEG .LE. 50). To change the size of the polynomials that can be solved, replace the dimensions of the arrays in common area and the NDEG test.

METHOD. RLPOLY uses a 3-stage variable-shift iteration method developed by M. A. Jenkins and J. F. Traub. The zeroes are calculated one or two at a time, generally in increasing order of magnitude, using real arithmetic wher possible. Failure of the iteration to converge for a root causes ar error message to be printed and the subprogram to exit with the zeroes already found stored in arrays ZEROR and ZEROI. Failures are rare.

REFERENCE. Jenkins, M. A. 1969. Three Stage Variable-Shift Iterations for the Solution of Polynomial Equations with a Posteriori Error Bounds for the Zeros. Technical Report No. CS 138. Stanford, Calif.:Computer Science Dept., Stanford University.

EXAMPLE. The sample calling program that follows can be used to solve the sample data following the program.

Sample Calling Program.

A sample calling program follows:

```
      DIMENSION ZR(50),ZI(50),CP(51)
      LOGICAL FAIL
      INTEGER DEGREE
   11 FORMAT(I5)
   13 FORMAT(8F10.0)
      READ(5,11) DEGREE
      NCOEF=DEGREE+1
      READ(5,13) (CP(I),I=1,NCOEF)
      CALL RLPOLY(DEGREE,CP,ZR,ZI,FAIL)
      STOP
      END
```

Sample Data.

The following is data for the preceding program:

```
      1.0      -1.01      -5.01      2.03      4.04      8.04
```

CXPOLY SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE CXPOLY(NDEG,PPR,PPI,ZEROR,ZEROI,FAIL)
DIMENSION ZEROR(NDEG),ZEROI(NDEG),PPR(1),PPI(1)
LOGICAL FAIL
```

PURPOSE. CXPOLY finds all zeroes of a complex polynomial PP = (PPR,PPI).

INPUT. Input to CXPOLY consists of:

NDEG - degree of the polynomial

PPR, PPI - arrays of the real and imaginary parts of the coefficients of the polynomial ordered from largest to smallest power; remain intact

OUTPUT. Output from CXPOLY consists of:

ZEROR, ZEROI - arrays of the real and imaginary parts of the zeroes of the polynomial

FAIL - Boolean value set TRUE if algorithm fails to find all zeroes; otherwise FALSE

RESTRICTIONS. The algorithm accepts polynomials of maximum degree 50 (1 .LE. NDEG .LE. 50). To change the size of polynomials that can be solved, replace the dimensions of the arrays in common area.

METHOD. CXPOLY uses a 3-stage variable-shift iteration method developed by M. A. Jenkins and J. F. Traub. The zeroes are calculated one at a time, generally in increasing order of magnitude, using complex arithmetic. Failure of the iteration to converge for a root causes an error message to be printed and the subprogram to exit with the zeroes already found stored in arrays ZEROR and ZEROI. Failures are rare.

REFERENCES. Jenkins, M. A. 1969. Three Stage Variable-Shift Iterations for the Solution of Polynomial Equations with a Posteriori Error Bounds for the Zeros. Technical Report No. CS 138. Stanford, Calif.:Computer Science Dept., Stanford University.

Jenkins, M. A. and Traub, J. F. 1972. ALGORITHM 419, Zeros of a Complex Polynomial. Comm. ACM 15:97-100.

EXAMPLE. The sample calling program given below can be used to solve the sample data that follows.

Sample Calling Program.
A sample calling program follows:

```
      DIMENSION ZR(50),ZI(50),PRL(51),PIM(51)
      LOGICAL FAIL
      INTEGER DEGREE
   11 FORMAT(I5)
```

```
   13 FORMAT(8F10.0)
      READ(5,11) DEGREE
      NCOEF=DEGREE+1
      READ(5,13) (PRL(I),PIM(I),I=1,NCOEF)
      CALL CXPOLY(DEGREE,PRL,PIM,ZR,ZI,FAIL)
      STOP
      END
```

Sample Data.
The following is data for the precedirg program:

```
  1.0       0.0      -8.1      -3.1       16.3      15.71      -2.89
 6.21     -8.62    -18.19     27.43
```

OPERATIONS ON POLYNOMIALS.

POLADD SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE POLADD(N1DEG,COEF1,N2DEG,COEF2,NRDEG,COEFR)
DIMENSION COEF1(1),COEF2(1),COEFR(1)
```

PURPOSE. POLADD adds two polynomials.

INPUT. Input to POLADD consists of:

N1DEG - degree + 1 of first polynomial

COEF1 - array of coefficients for first polynomial ordered from largest to smallest power; dimension [1:N1DEG]

N2DEG - degree + 1 of second polynomial

COEF2 - array of coefficients for second polynomial ordered from largest to smallest power; dimension [1:N2DEG]

OUTPUT. Output from POLADD consists of:

NRDEG - degree + 1 of resultant polynomial

COEFR - array of coefficients for resultant polynomial ordered from largest to smallest power; dimension [1:NRDEG)

METHOD. The degree NRDEG-1 of the resultant polynomial is calculated as the larger of the two degrees N1DEG-1 and N2DEG-1. Corresponding coefficients are added to form the array COEFR.

POLSUB SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE POLSUB(N1DEG,COEF1,N2DEG,COEF2,NRDEG,COEFR)
DIMENSION COEF1(1),COEF2(1),COEFR(1)
```

PURPOSE. POLSUB subtracts one polynomial from another.

INPUT. Input to POLSUB consists of:

N1DEG - degree + 1 of first polynomial

COEF1 - array of coefficients for first polynomial ordered from largest to smallest power; dimension [1:N1DEG]

N2DEG - degree + 1 of second polynomial

COEF2 - array of coefficients for second polynomial ordered from largest to smallest power; dimension [1:N2DEG]

OUTPUT. Output from POLSUB consists of:

NRDEG - degree + 1 of resultant polynomial

COEFR - array of coefficients for resultant polynomial ordered from largest to smallest power; dimension [1:NRDEG]

METHOD. The degree NRDEG-1 of the resultant polynomial is calculated as the larger of the two degrees N1DEG-1 and N2DEG-1. Coefficients in array COEF2 are subtracted from corresponding coefficients in array COEF1.

POLMUL SUBRCUTINE.
The syntax is as follows:

```
SUBRCUTINE POLMUL(N1DEG,COEF1,N2DEG,COEF2,NRDEG,COEFR)
DIMENSION COEF1(1),COEF2(1),COEFR(1)
```

PURPOSE. POLMUL multiplies two polynomials.

INPUT. Input to POLMUL consists of:

N1DEG - degree + 1 of first polynomial

COEF1 - array of coefficients for first polynomial ordered from largest to smallest power; dimension [1:N1DEG]

N2DEG - degree + 1 of second polynomial

COEF2 - array of coefficients for second polynomial ordered from largest to smallest power; dimension [1:N2DEG]

OUTPUT. Output from POLMUL consists of:

NRDEG - degree + 1 of resultant polynomial

COEFR - array of coefficients for resultant polynomial ordered from largest to smallest power; dimension [1:NRDEG]

METHOD. The degree NRDEG-1 of the resultant polynomial is calculated as N1DEG+N2DEG-2. The coefficients of array COEFR are calculated as the sum of the products of the coefficients of COEF1 and COEF2 having exponents that sum to the corresponding exponent of COEFR.

REMARKS. COEFR cannot occupy the same location as COEF1 or COEF2.

POLDIV SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE POLDIV(N1DEG,COEF1,N2DEG,,COEF2,NQDEG,COEFQ,TOL,ERRCOD)
DIMENSION COEF1(1),COEF2(1),COEFQ(1)
```

PURPOSE. POLDIV divides one polynomial by another.

INPUT. Input to POLDIV consists of:

N1DEG - degree + 1 of dividend polynomial; replaced by degree + 1 of remainder

COEF1 - array of coefficients for dividend polynomial ordered from largest to smallest power; replaced by remainder after division; dimension [1:N1DEG]

N2DEG - degree + 1 of divisor polynomial

COEF2 - array of coefficients for divisor polynomial ordered from largest to smallest power; dimension [1:N2DEG]

TOL - tolerance value below which coefficients are eliminated during normalization

OUTPUT. Output from POLDIV consists of:

N1DEG - degree + 1 of remainder polynomial

COEF1 - array of coefficient for remainder polynomial; dimension [1:N1DEG]

NQDEG - degree + 1 of quotient polynomial

COEFQ - array of coefficients of quotient polynomial ordered from largest to smallest power; dimension [1:NQDEG]

ERRCOD - error code: Zero is normal, and 1 is for zero divisor.

METHOD. Polynomial COEF1 is divided by polynomial COEF2 giving the quotient COEFQ and the remainder such that COEF1 = COEFQ*COEF2+remainder. The divisor array COEF2 and the remainder array get normalized.

REMARKS. The remairder replaces COEF1. If N2DEG equals 1, then N1DEG is set to zero. The divisor COEF2 remains unchanged. If the degree of the civisor polynomial is greater than that of the dividend, array NQDEG is set to zero and the calculation is bypassed.

POLDER SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE POLDER(NDEG,COEF,NDDEG,COEFD)
DIMENSION COEF(1),COEFD(1)
```

PURPOSE. POLDER finds the derivative of a polynomial.

INPUT. Input to POLDER consists of:

NDEG - degree + 1 of polynomial

COEF - array of coefficients for original polynomial ordered from largest to smallest power; dimension [1:NDEG]

OUTPUT. Output from POLDER consists of:

NDDEG - degree + 1 of derivative; equals NDEG-1

COEFD - array of coefficients for derivative ordered from largest to smallest power; dimension [1:NDDEG]

METHOD. The parameter NDDEG is set at NDEG-1. The derivative is then calculated by multiplying the coefficients by their respective exponents.

POLINT SUBRCUTINE.
The syntax is as follcws:

```
SUBRCUTINE POLINT(NDEG,COEF,C,NIDEG,COEFI)
DIMENSION COEF(1),COEFI(1)
```

PURPOSE. PCLINT firds the integral of a polynomial.

INPUT. Input tc POLINT consists of:

C - corstant of integration

NDEG - degree + 1 cf original polynomial

COEF - array of coefficients for original polynomial ordered from largest to smallest pcwer; dimension [1:NDEG]

OUTPUT. Output from POLINT consists cf:

NIDEG - degree + 1 of integral; equals NDEG+1

COEFI - array cf ccefficients for integral polynomial ordered from largest to smallest power; dimension [1:NIDEG]

METHOD. The parameter NIDEG is set at NDEG+1, and the constant term is set at C. The integral is calculated by dividing the coefficients by their respective exponents.

POLHOR SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE POLHOR(NDEG,COEF,BOOL,ARG,R,K)
DIMENSION COEF(1),R(1)
```

PURPOSE. POLHOR finds the values of the first K derivatives of a polynomial for a given value of the variable. POLHOR can also calculate the values of the first K coefficients of the expansion of the polynomial in a power series in the neighborhood of a given point.

INPUT. Input to POLHOR consists of:

NDEG - degree + 1 of polynomial

COEF - array of coefficients ordered from largest to smallest power; dimension [1:NDEG]

BOOL - Boolean variable set TRUE if the values of the first K-1 derivatives are desired; set FALSE if the expansion of the polynomial in a power series is desired

ARG - value of variable if BOOL is TRUE; if BOOL is FALSE, neighborhood at which power series expansion is desired

K - K-1 is the highest order of the derivative if BOOL is TRUE; if BOOL is FALSE, largest degree of the term with which the power series is terminated

OUTPUT. Output from POLHOR consists of:

R - array of values of derivatives if BOOL is TRUE; Jth entry is the value of the (J-1)th derivative. If BOOL is FALSE, array of values of the first K coefficients of the power series. Dimension R[1:K]

METHOD. The Horner scheme is used.

REFERENCE. Pankiewicz, W. 1968. ALGORITHM 337, Calculation of a Polynomial and Its Derivative Values by Horner Scheme. Comm. ACM 11:663.

POLMAD SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE POLMAD(N1DEG,COEF1,C,N2DEG,COEF2,NRDEG,COEFR)
DIMENSION COEF1(1),COEF2(1),COEFR(1)
```

PURPOSE. POLMAD adds the coefficients of one polynomial to the product of a factor and coefficients of another polynomial.

INPUT. Input to POLMAD consists of:

N1DEG - degree + 1 of first polynomial

COEF1 - array of coefficients for first polynomial ordered from largest to smallest power; dimension [1:N1DEG]

N2DEG - degree + 1 of second polynomial

COEF2 - array of coefficients for second polynomial ordered from largest to smallest power; dimension [1:N2DEG]

C - factor to be multiplied by array COEF2

OUTPUT. Output from POLMAD consists of:

NRDEG - degree + 1 of resultant polynomial

COEFR - array of resultant coefficients ordered from largest to smallest power; dimensior [1:NRDEG]

METHOD. The degree NRDEG-1 of the resultant polynomial is calculated as the larger of the two degrees N1DEG-1 and N2DEG-1. To form array COEFR, the coefficients in array COEF1 are added to the corresponding coefficients in array COEF2 multiplied by C.

POLPOP SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE POLPOP(N1DEG,COEF1,N2DEG,COEF2,NRDEG,COEFR,CT1,CT2)
DIMENSION COEF1(1),COEF2(1),COEFR(1),CT1(1),CT2(1)
```

PURPOSE. POLPOP substitutes the variable of a polynomial with another polynomial.

INPUT. Input to POLPOP consists of:

N1DEG - degree + 1 of first polynomial

COEF1 - array of coefficients for first polynomial ordered from largest to smallest power; dimension [1:N1DEG]

N2DEG - degree + 1 of second polynomial

COEF2 - array of coefficient for second polynomial ordered from largest to smallest power; dimension [1:N2DEG]

OUTPUT. Output from POLPOP consists of:

NRDEG - degree + 1 of resultant polynomial

COEFR - array of coefficients for resultant polynomial ordered from largest to smallest power; dimension [1:NRDEG]

OTHER. Additional parameters of POLPOP are:

CT1, CT2 - arrays for internal calculations; dimension [1:(N1DEG-1)*(N2DEG-1)+1]

METHOD. The variable of polynomial COEF1 is substituted by polynomial COEF2 to form polynomial COEFR. The degree of the resultant polynomial is (N1DEG-1)*(N2DEG-1).

REMARKS. If either N1DEG or N2DEG is negative, NRDEG is set at zero and calculation is bypassed.

POLMOV SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE POLMOV(N1DEG,COEF1,N2DEG,COEF2)
DIMENSION COEF1(1),COEF2(1)
```

PURPOSE. POLMOV moves one polynomial to another.

INPUT. Input to POLMOV consists of:

N1DEG - degree + 1 of polynomial to be moved

COEF1 - array of coefficients of polynomial, ordered from largest to smallest power, to be moved; dimension [1:N1DEG]

OUTPUT. Output from POLMOV consists of:

N2DEG - degree + 1 of resultant polynomial

COEF2 - array of coefficients of resultant polynomial ordered from largest to smallest power; dimension [1:N2DEG]

METHOD. Parameter N2DEG is replaced by N1DEG, and array COEF1 is moved to COEF2.

POLNOR SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE POLNOR(NDEG,COEF,TOL)
DIMENSION COEF(1)
```

PURPOSE. POLNOR normalizes the coefficient array of a polynomial.

INPUT. Input to POLNOR consists of:

NDEG - degree + 1 of polynomial; replaced by final degree + 1

COEF - array of original coefficients ordered from largest to smallest power; replaced by final coefficients; dimension [1:NDEG]

TOL - tolerance value below which coefficients are eliminated

OUTPUT. Output from POLNOR consists of:

NDEG - degree + 1 of normalized polynomial
COEF - array of coefficients for normalized polynomial

METHOD. The degree NDEG-1 of the polynomial is reduced by 1 for each leading coefficient having an absolute value less than or equal to TOL. Coefficients are then moved to the left so the leading coefficient is in COEF(1).

REMARKS. If all coefficients are less than TOL, the result is a zero polynomial with NDEG equal to zero.

POLSOR SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE POLSOR(NDEG,COEF,U)
DIMENSION COEF(1)
```

PURPOSE. POLSOR shifts the origin.

INPUT. Input to POLSOR consists of:

NDEG - degree + 1 of polynomial

COEF - array of coefficients ordered from largest to smallest power; replaced by array of transformed coefficients

U - shift parameter

OUTPUT. Output from POLSOR consists of:

COEF - array of transformed coefficients

METHOD. The coefficient array COEF(I) of polynomial P(X) is transformed such that Q(X) = P(X-U), where Q(X) denotes the polynomial having the transformed coefficient array.

POLGCD SUBROUTINE.
The syntax is as follows:

```
SUBROUTINE POLGCD(N1DEG,COEF1,N2DEG,COEF2,TOL,ERRCOD,COEFT)
DIMENSION COEF1(1),COEF2(1),COEFT(1)
```

PURPOSE. POLGCD finds the greatest common divisor of two polynomials.

INPUT. Input to POLGCD consists of:

N1DEG - degree + 1 of first polynomial

COEF1 - array of coefficients of first polynomial ordered from largest to smallest power; dimension [1:N1DEG]

N2DEG - degree + 1 of second polynomial

COEF2 - array of coefficients for second polynomial ordered from largest to smallest power; replaced by greatest common divisor; dimension [1:N2DEG]

TOL - tolerance value below which coefficient is eliminated during normalization

OUTPUT. Output from POLGCD consists of:

N2DEG - degree of greatest common divisor polynomial

COEF2 - coefficient array for greatest common divisor

ERRCOD - resultant error code: Zero indicates no error; 1 indicates COEF1 or COEF2 is a zero polynomial.

OTHER. An additional parameter of POLGCD is:

COEFT - array for internal calculations; dimension [1:N1DEG-N2DEG+1]

METHOD. The greatest common divisor of the polynomials COEF1 and COEF2 is determined by a Euclidean algorithm. The coefficient arrays COEF1 and COEF2 are destroyed. The greatest common divisor is generated in COEF2.

REMARKS. The parameter N1DEG must be greater than N2DEG. If N2DEG equals 1 on return, the arrays COEF1 and COEF2 are prime. The greatest common divisor is a constant. The parameter N1DEG is destroyed during the computation. For division of COEF1 by COEF2 with the result stored in COEFT and the remainder in COEF1, use the call POLDIV(N1DEG,COEF1,N2DEG,COEF2,NTDEG,COEFT,TOL,ERRCOD).

SECTION 13

FORTRAN SPECIAL FUNCTIONS

CXBESJ SUBROUTINE.

The syntax is as follows:

```
SUBRCUTINE CXBESJ(NMAX,X,Y,U,V,UA,VA)
DIMENSION U(1),V(1),UA(1),VA(1)
```

PURPOSE.
CXBESJ evaluates the Bessel functions (of the first kind) JO,...,JNMAX at the fixed complex value Z = X+I*Y.

INPUT.
Input to CXBESJ consists cf:

NMAX - integer which can be positive, negative, or zero
X - unrestricted real part of argument
Y - unrestricted imaginary part cf argument

OUTPUT.
Output for CXBESJ consists of:

U - contains the real parts of the values JO,...,JNMAX in U(I), I=1,...,NM+1 where NM = ABS(NMAX)

V - contains the imaginary parts of the values JO,...,JNMAX in V(I), I=1,...,NM+1 where NM = ABS(NMAX)

OTHER PARAMETERS.
Additional CXBESJ parameters are:

UA - working array used for UAPPROX and RR1; must be at least of size 2*NM+1

VA - workirg array used for VAPPROX and RR2; must be at least of size 2*NM+1

FUNCTION REQUIRED.
A necessary functior is:

TOFY

METHOD.
This subroutine is a special case of cne given by Gautschi. The subroutine uses an iterative backward recurrence algorithm. The iteration is continued until approximately D (set at 11) significant digits of accuracy are obtained. The algorithm is most efficient when MODULUS(Z) is small or moderately large. If NMAX is negative, the following formula is used:

```
J(-K,Z) = (-1)**K*J(K,Z)
```

REMARKS.
Values of X and Y with absolute values less than ETOL are taken to be zeroes.

REFERENCE.
Gautschi, W. 1964. ALGORITHM 236, Bessel Functions of the First Kind. Comm. ACM 7:479-480.

BESSFJ FUNCTION.
The syntax is as follows:

```
FUNCTION BESSFJ(NMAX,X,J,JA)
REAL J(1),JA(1)
```

PURPOSE.
BESSFJ evaluates the Bessel functions (of the first kind) J0,...,JNMAX at the fixed value X.

INPUT.
Input to BESSFJ consists of:

NMAX - integer which can be positive, negative, or zero
X - unrestricted real value

OUTPUT.
Output for BESSFJ consists of:

J - array containing the values J0,...,JNMAX in J(I), I=1,...,NM+1 where NM = ABS(NMAX)

OTHER PARAMETER.
An additional parameter of BESSFJ is:

JA - working array; must be at least of size 2*NM+1

FUNCTION REQUIRED.
A required function is:

TOFY

METHOD.
This function is a special case of one given by Gautschi. The function uses an iterative backward recurrence algorithm. The iteration is continued until approximately D (set at 11) significant digits of accuracy are obtained. The algorithm is most efficient when X is small or moderately large in absolute value. If NMAX or X is negative, the following relations are used:

```
J(-K,X) = (-1)**K*J(K,X)
J(K,-X) = (-1)**K*J(K,X)
```

REMARKS.
For values of X with absolute value less than ETOL, X is taken to be zero.

REFERENCE.
Gautschi, W. 1964. ALGORITHM 236, Bessel Functions of the First Kind. Comm. ACM 7:479-480.

SECTION 14

FORTRAN NUMERICAL INTEGRATION

SINGLE INTEGRATION.

SIMINT FUNCTION.
The syntax is as follows:

```
FUNCTION SIMINT(A,B,EPS)
```

PURPOSE. SIMINT numerically integrates a function of one variable over a finite interval.

INPUT. Input to SIMINT consists of:

A - lower limit of integration
B - upper limit of integration
EPS - relative error tolerance

USAGE. SIMINT is adaptive and is recommended for moderate accuracies (EPS .LE. 1.0E-6) with complicated integrands and integrands having large or infinite derivatives.

EXAMPLE. The following is a sample calling program that can be used with the sample function F(X) = 1.0/SQRT(ABS(XEE)).

```
A=-9.0
B=1000.0
EPS=0.0001
VALUE=SIMINT(A,B,EPS)
STOP
END
```

METHOD. SIMINT uses the adaptive Simpson method with interval bisection.

REMARKS. The function to be integrated must be supplied as a function subprogram named F(X).

REFERENCES. Lyness, J. N. 1969. Notes on the Adaptive Simpson Quadriture Routine. Journal of ACM 16:483-495.

McKeemen, W. M. 1962. ALGORITHM 145, Adaptive Numerical Integration by Simpson's Rule. Comm. ACM 5:604.

BULINT FUNCTION.
The syntax is as follows:

```
FUNCTION BULINT(A,B,EPS,BINT,ERR)
```

PURPOSE. BULINT integrates a functior of one variable over a finite interval.

INPUT. Input to BULINT consists of:

A - lower limit of integration
B - upper limit of integration
EPS - relative error tolerance

OUTPUT. Cutput for BULINT consists of:

BINT - value of integral
ERR - upper bound for ABS(BINT-true value of integral)

RETURNED VALUE. The returned value BULINT is the value of the integral.

USAGE. BULINT is recommended for high accuracies (EPS .LE. 1.0E-8) and smooth integrands. This function is rot to be used to integrate polynomials of degree less than 4 or periodic functions over a periodicity interval.

METHOD. Operation of BULINT is based on the rational extrapolation of trapezoid-rule approximations to the integral. If the error criterion:

ABS(BINT-true value) .LT. EPS*integral of ABS(F(X))

cannot be satisfied, ERR is set to 10 and the subprogram exits with BINT cortairing the final approximation to the integral.

REMARKS. To return the value of F at X, the function to be integrated must be supplied as a function subprogram named F(X).

REFERENCE. Bulirsch, R. and Stoer, J. 1967. Numerical Quadrature by Extrapolation. Numerische Mathematik 9:271-278.

ROMINT FUNCTION.
The syntax is as follows:

```
FUNCTION ROMINT(A,B,EPS)
```

PURPOSE. ROMINT integrates a function of one variable over a finite interval.

INPUT. Input to ROMINT consists of:

A - lower limit of integration
B - upper limit of integration
EPS - relative error tolerance

RETURNED VALUE. The returned value ROMINT is the value of the integral.

USAGE. ROMINT is used with moderately smooth integrands and for medium accuracy (for example, EPS = 1.0E-6).

REMARKS. To return the value of F at X, the function to be integrated must be supplied as a function subprogram named F(X).

REFERENCE. Bauer, F. L.; Rutishauser, H.; and Stiefel, E. 1963. New Aspects in Numerical Quadriture. In Experimental Arithmetic, High Speed Computing, and Mathematics. Proc. of Symposia in Applied Mathematics 15:199-218. Providence, R.I.: Amer. Math. Society.

MULTIPLE INTEGRATION.

DBLINT FUNCTION.
The syntax is as follows:

```
FUNCTION DBLINT(A1,B1,A2,B2,EPS)
```

PURPOSE. DBLINT numerically integrates a function of two variables over a rectargle.

INPUT. Input to DBLINT ccnsists of:

A1 - lower limit of integration cn X-axis
B1 - upper limit of integration cn X-axis
A2 - lower limit of integration cn Y-axis
B2 - upper limit of integration cn Y-axis
EPS - relative error tolerance

USAGE. DBLINT is used only for double integrals with constant limits. A subprogram is adaptive to handle general integrands. EPS should be greater than or equal to 1.0E-6.

EXAMPLE. The fcllowing is a sample calling program for the sample function F(X,Y) = 1.0+X*XY*Y.

```
A1=-1.0
B1=1.0
A2=-1.0
B2=1.0
EPS=0.0001
VALUE=DBLINT(A1,B1,A2,B2,EPS)
STCP
END
```

METHOD. The double integral is treated as a 1-dimensional integral of a function which is itself an integral. The adaptive Simpson method using interval bisection is used for the 1-dimensional integration. DBLINT is a modificatior cf SIMINT for double integrals.

REMARKS. The function to be integrated must be supplied as a function subprogram ramed F(X,Y).

REFERENCES. McKeemen, W. M. 1962. ALGORITHM 145, Adaptive Numerical Integration by Simpson's Rule. Comm. ACM 5:604.

-----.1962. ALGORITHM 146, Multiple Integration. Comm. ACM 5:604.

TPLINT FUNCTION.
The syntax is as follows:

```
FUNCTION TPLINT(A1,B1,A2,B2,A3,B3,EPS)
```

PURPOSE. TPLINT numerically integrates a function of three variables over a 3-dimensional rectangle.

INPUT. Input to TPLINT consists of:

```
A1 - lower limit of integration on X-axis
B1 - upper limit of integration on X-axis
A2 - lower limit of integration on Y-axis
B2 - upper limit of integration on Y-axis
A3 - lower limit of integration on Z-axis
B3 - upper limit of integration on Z-axis
EPS - relative error tolerance
```

USAGE. TPLINT is used only for triple integrals with constant limits. A subprogram is adaptive to handle gereral integrands. EPS should be greater than or equal to 1.0E-6.

EXAMPLE. The following is a sample calling program for the sample function F(X,Y,Z) = 1.0+X*X+Y*Y+Z*Z.

```
A1=-1.0
B1=1.0
A2=-1.0
B2=1.0
A3=-1.0
B3=1.0
EPS=0.0001
VALUE=TPLINT(A1,B1,A2,B2,A3,B3,EPS)
STOP
END
```

METHOD. TPLINT uses an extension of the method used for DBLINT, which is a modification of SIMINT.

REMARKS. The function to be integrated must be supplied as a function subprogram named F(X,Y,Z).

REFERENCES. McKeemen, W. M. 1962. ALGORITHM 145, Adaptive Numerical Integration by Simpson's Rule. Comm. ACM 5:604.

-----.1962. ALGORITHM 146, Multiple Integration. Comm. ACM 5:604.

SECTION 15

FOURIER TRANSFORMS

GENERAL.
This section describes four procedures that compute Fourier transforms by the Cocley-Tukey method (fast Fourier transforms). The four procedures are:

a. FFTF - computes fast Fourier transforms.

b. FFTR - computes inverse Fourier transforms.

c. SINCOS - generates tables of sines and cosines for use by procedures FFTF and FFTR.

d. BITREV2 - arranges arrays into bit-reversed order for use by procedures FFTF and FFTR.

FFTF PROCEDURE.
The syntax is as follows:

```
PROCEDURE FFTF(XR,XI,S,C,M)
```

PURPOSE.
FFTF computes the fast Fourier transform using the Cooley-Tukey method.

INPUT.
Input to FFTF consists of:

XR - real part of the input array
XI - imagirary part of the input array
S - array of sines produced by SINCOS
C - array of cosines produced by SINCOS
M - integer that specifies the size of XR and XI to be 2**M

OUTPUT.
The output from FFTF consists of:

XR - real part of the transformed array (bit-reversed order)
XI - imaginary part of the transformed array (bit-reversed order)

METHOD.
Let Z(0),Z(1),...Z(2**M-1) be 2**M complex numbers. FFTF computes the complex numbers W(0),W(1),...,W(2**M-1) given by:

```
W(L) = sum over J from zero to 2**M-1 of Z(J)EXP(2*P*I*J*K/2**M)
```

where

P = constant Pi
K = value from zero to 2**M-1
L = bit-reversed representation of K in a field of length M
I = square root of -1

REMARKS.
The output is not normalized. A call to FFTF followed immediately by FFTR reproduces the original data multiplied by 2**M.

REFERENCES.
Singleton, R. C. 1967. On Computing the Fast Fourier Transform. Comm. ACM 10:647-654.

Bracewell, R. 1965. The Fourier Transform and Its Applications. New York:McGraw Hill.

FFTR_PROCEDURE.
The syntax is as follows:

```
PROCEDURE FFTR(XR,XI,S,C,M)
```

PURPOSE.
FFTR computes the irverse Fourier trarsform using the Cooley-Tukey method.

INPUT.
Input tc FFTR ccnsists of:

```
XR - real part of the input array (bit-reversed order)
XI - imaginary part cf the input array (bit-reversed order)
S - array of sines produced by SINCOS
C - array cf ccsines produced by SINCOS
M - integer specifying the size cf XR and XI to be 2**M
```

OUTPUT.
Output from FFTR corsists of:

```
XR - real part of transformed array
XI - imaginary part cf transformed array
```

METHOD.
For a ccmplex array W in bit-reversed order, FFTR computes the complex numbers Z(0),Z(1),...,Z(2**M-1) given by:

```
Z(K) = sum over J frcm zero to 2**M-1 of W(L)EXP(-2*P*I*J*K/2**M)
```

where:

```
P = constant Pi
K = values from zero to 2**M-1
L = bit-reversed representation cf K in a field of length M
I = square root of -1
```

REMARKS.
The output is not normalized. A call to FFTF followed immediately by FFTR reproduces the original data multiplied by 2**M.

REFERENCES.
Singleton, R. C. 1967. On Computing the Fast Fourier Transform. Comm. ACM 10:647-654.

Bracewell, R. 1965. The Fourier Transform and Its Applications. New York: McGraw Hill.

SINCOS PROCEDURE.
The syntax is as follows:

```
PROCEDURE SINCOS(S,C,M)
```

PURPOSE.
SINCOS sets up sine and cosine tables for fast Fourier transforms.

INPUT.
Input to SINCOS consists of:

M - integer specifying length of arrays S and C to be 2**(M-1)

OUTPUT.
Output from SINCOS consists of:

S - array of sines needed by FFTF and FFTR
C - array of cosines needed by FFTF and FFTR

REMARKS.
The dimensions of arrays S and C should each be declared ARRAY[0:2**(M-1)].

BITREV2 PROCEDURE.
The syntax is as follows:

```
PROCEDURE BITREV2(XR,XI,N)
```

PURPOSE.
BITREV2 reorders the contents of arrays XR and XI by bit reversal for use with the fast Fourier transform procedures FFTF and FFTR. The parameter N specifies the length of XR and XI to be 2**N. For N equal to 13, 14, 15, or 16, the sequence of exchanges is designed to minimize overlays.

INPUT.
Input to BITREV2 consists of:

XR - input array for bit-reversed arrangement
XI - input array for bit reversal
N - specifies length of XR and XI to be 2**N

OUTPUT.
Output from BITREV2 consists of:

XR - bit-reversed arrangement of input XR
XI - bit-reversed arrangement of input XI

APPENDIX A

PROGRAM EXAMPLES

DETLINEQN

```
BEGIN

%*************************************************************************
 COMMENT      PROGRAM TO SOLVE A SINGLE SYSTEM OF LINEAR EQUATIONS AND
    CALCULATE THE DETERMINANT OF THE COEFFICIENT MATRIX,
    USES PROCEDURES DETLINEQN AND READMATRIX ;
%*************************************************************************

 $ INCLUDE "MATHLIB/SYMBOL/READMATRIX."
 $ INCLUDE "MATHLIB/SYMBOL/DETLINEQN."

 FILE          READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=14),
               PRINTIT(KIND=PRINTER, BUFFERS=2, MAXRECSIZE=22);
 INTEGER       I,N,DETEXP;
 REAL          DETBASE;
 LABEL         NEXTCASE,EOF;
 NEXTCASE:
          READ(READIT,<I3>,N) [EOF];
          BEGIN
 ARRAY         A[0:N,0:N], B[0:N];
             READMATRIX(N,N,A,READIT,PRINTIT,EOF);
 COMMENT       READ RIGHT-HAND-SIDE ;
             READ(READIT,/,FOR I := 1 STEP 1 UNTIL N DO B[I]);
             WRITE(PRINTIT,<//"RIGHT-HAND-SIDE",//(R18,10)>,
                           FOR I := 1 STEP 1 UNTIL N DO B[I]);
             DETLINEQN(N,A,A,B,B,DETBASE,DETEXP);
             WRITE(PRINTIT,<//"SOLUTION",//(R18,11)>,
                           FOR I := 1 STEP 1 UNTIL N DO B[I]);
             IF DETEXP = 0 THEN
             WRITE(PRINTIT,<//"DETERMINANT =",R18,11>,DETBASE) ELSE
             WRITE(PRINTIT,<//"DETBASE =",R18,11,X5,"DETEXP =",I5>,
                           DETBASE,DETEXP);
          END;
          GO TO NEXTCASE;
 EOF:
 END.

  4
 1   1   1   1
 2   1   1   1
 2   3   1   1
 2   3   4   1
 0
 1, 1, 1, 1,
```

LINEQNS

```
BEGIN

%*************************************************************************
 COMMENT      PROGRAM TO SOLVE A SYSTEM OF LINEAR EQUATIONS WITH THE
    SAME COEFFICIENT MATRIX AND SEVERAL RIGHT-HAND-SIDES,
    USES PROCEDURES LINEQNS AND READMATRIX ;
%*************************************************************************

 $ INCLUDE "MATHLIB/SYMBOL/READMATRIX,"
 $ INCLUDE "MATHLIB/SYMBOL/LINEQNS,"

 FILE          READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=14),
               PRINTIT(KIND=PRINTER, BUFFERS=2, MAXRECSIZE=22);
 INTEGER       I,J,K,N;
 LABEL         NEXTCASE,EOF;
 NEXTCASE:
         READ(READIT,<2I3>,N,K) [EOF];
         BEGIN
 ARRAY         A[0:N,0:N], C,Y[0:K,0:N];
            READMATRIX(N,N,A,READIT,PRINTIT,EOF);
            WRITE(PRINTIT,<//"RIGHT-HAND-SIDE MATRIX"//
                          I3," RIGHT-HAND-SIDES STORED AS ROWS"/>,K);
 COMMENT       READ THE RIGHT-HAND-SIDE MATRIX, WITH EACH RIGHT-HAND-
    SIDE ENTERED AS A SEPARATE ROW;
            READMATRIX(K,N,C,READIT,PRINTIT,EOF);
            LINEQNS(N,A,K,C,Y);
            FOR I := 1 STEP 1 UNTIL K DO
            WRITE(PRINTIT,<//"SOLUTION",I3//(R18,11)>,I,
                          FOR J := 1 STEP 1 UNTIL N DO Y[I,J]);
         END;
         GO TO NEXTCASE;
 EOF:
 END.

  6  2
 1, 0, 0, 0, 0, 1,
 1, 1, 0, 0, 0, -1,
 -1, 1, 1, 0, 0, 1,
 1, -1, 1, 1, 0, -1,
 -1, 1, -1, 1, 1, 1,
 1, -1, 1, -1, 1, -1,
 0, 0, 0, 32, 0, 0,
 0, 0, 0, 0, 32, 0,
```

INVERSEOVERA

```
 BEGIN

%*************************************************************************
 COMMENT       SAMPLE PROGRAM WHICH CALCULATES THE INVERSE OF THE N-TH
    ORDER MATRIX GENERATED BY PROCEDURE INVERSEHILBERT,  THE GENERATED
    MATRIX IS THE EXACT INVERSE OF THE N-TH ORDER HILBERT MATRIX.
    PROCEDURE INVERSEOVERA IS USED TO CALCULATE THE INVERSE ;
%*************************************************************************

$ INCLUDE "MATHLIB/SYMBOL/INVERSEOVERA."

 FILE          READIT(KIND = READER, BUFFERS = 2, MAXRECSIZE = 14),
               PRINTIT(KIND = PRINTER, BUFFERS = 2, MAXRECSIZE = 22);
 FORMAT        ROW(/(8(R15.8,X1))),
               ROWOUT(/(6(R18.11,X2))),
               ORDER("ORDER OF MATRIX = ",I3,//"INPUT MATRIX"//),
               INV(//"INVERSE MATRIX"//);
 INTEGER       I,J,N;
         READ(READIT,/,N);
         WRITE(PRINTIT,ORDER,N);
     BEGIN
 ARRAY         A[0:N,0:N];
         INVERSEHILBERT(N,A);
         FOR I := 1 STEP 1 UNTIL N DO
         WRITE(PRINTIT,ROW,FOR J := 1 STEP 1 UNTIL N DO A[I,J]);
         INVERSEOVERA(N,A);
         WRITE(PRINTIT,INV);
         WRITE(PRINTIT,ROWOUT,FOR J := 1 STEP 1 UNTIL N DO A[I,J]);
     END;
 END.

 4,
```

SYMLINEQN

```
BEGIN

%*************************************************************************
 COMMENT      PROGRAM TO SOLVE A SINGLE SYSTEM OF LINEAR EQUATIONS
    WITH A SYMMETRIC COEFFICIENT MATRIX.  ONLY THE LOWER TRIANGULAR
    HALF OF THE MATRIX (INCLUDING THE DIAGONAL) IS NEEDED AS INPUT.
    USES PROCEDURES SYMLINEQN AND READSYMMATRIX ;
%*************************************************************************

 INTEGER  J;
 DEFINE       FORI = FOR I := 1 STEP 1 UNTIL N DO #,
              FORJ = FOR J := 1 STEP 1 UNTIL N DO #;
 FILE         READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=14),
              PRINTIT(KIND=PRINTER, BUFFERS=2, MAXRECSIZE=22);

 $ INCLUDE "MATHLIB/SYMBOL/READSYMMATRIX."
 $ INCLUDE "MATHLIB/SYMBOL/SYMLINEQN."

 INTEGER      I,N;
 LABEL        NEXTCASE,EOF;
 NEXTCASE:
         READ(READIT,<I3>,N) [EOF];
         BEGIN
 ARRAY        A[0:N,0:N], B,X[0:N];
 LABEL        SINGULAR;
           READSYMMATRIX(N,A,READIT,PRINTIT,EOF);
 COMMENT     READ RIGHT-HAND-SIDE ;
           READ(READIT,/,FOR I := 1 STEP 1 UNTIL N DO B[I]);
           WRITE(PRINTIT,<//"RIGHT-HAND-SIDE",//(R18,10)>,
                         FOR I := 1 STEP 1 UNTIL N DO B[I]);
           SYMLINEQN(N,A,B,X,SINGULAR);
           WRITE(PRINTIT,</"SOLUTION"/(R18,11)>, FORJ X[J]);
         IF FALSE THEN
 SINGULAR: WRITE(PRINTIT,<//"EXITED AT SINGULAR">);
         END;
         GO TO NEXTCASE;
 EOF:
END.
```

```
 4
.6
-.5 1
0 -.5 1
.1 0 -.5 .6
1, 0, 0, 0,
```

SYMBANDLINEQN

```
BEGIN

%*************************************************************************
COMMENT      PROGRAM TO SOLVE A SYSTEM OF BANDED, SYMMETRIC, POSITIVE
   DEFINITE LINEAR EQUATIONS WITH ONE OR MORE RIGHT-HAND SIDES.  M IS
   THE NUMBER OF DIAGONALS BELOW (OR ABOVE) THE MAIN DIAGONAL AND N
   IS THE ORDER OF THE MATRIX.
   USES PROCEDURE SYMBANDLINEQN ;
%*************************************************************************

$ INCLUDE "MATHLIB/SYMBOL/SYMBANDLINEQN."

FILE          READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=14),
              PRINTIT(KIND=PRINTER, BUFFERS=2, MAXRECSIZE=22);
FORMAT        ROW(/(6(F18.11,X2)));
INTEGER       I,J,N,M;
BOOLEAN       FIRST;
         FIRST := TRUE;
         READ(READIT,/,N,M);
         WRITE(PRINTIT,</"ORDER =",I3,X5,"BANDWIDTH =",I3/>,N,M);
    BEGIN
ARRAY         A[0:N,0:M], B,X[0:N];
LIST          RHS(FOR J := 1 STEP 1 UNTIL N DO B[J]),
              SOL(FOR J := 1 STEP 1 UNTIL N DO X[J]);
LABEL         BADMAT,GETRHS,EOF;
         FOR I := 1 STEP 1 UNTIL N DO
         READ(READIT,/,FOR J := 0 STEP 1 UNTIL M DO A[I,J]);
         WRITE(PRINTIT,<"INPUT MATRIX"/>);
         FOR I := 1 STEP 1 UNTIL N DO
         WRITE(PRINTIT,ROW,FOR J := 0 STEP 1 UNTIL M DO A[I,J]);
GETRHS:
         READ(READIT,/,RHS) [EOF];
         WRITE(PRINTIT,<//"RIGHT-HAND-SIDE"/>);
         WRITE(PRINTIT,ROW,RHS);
         SYMBANDLINEQN(N,M,A,B,X,FIRST,BADMAT);
         WRITE(PRINTIT,</"SOLUTION"/>);
         WRITE(PRINTIT,ROW,SOL);
         FIRST := FALSE;
         GO TO GETRHS;
         IF FALSE THEN
BADMAT:     WRITE(PRINTIT,<//"MATRIX IS NOT POSITIVE DEFINITE">);
EOF:
    END;
END.

5, 1,
0,1,
-1,2,
-1,2,
-1,2,
-1,2,
1,0,0,0,0,
0,0,0,0,1,
```

CXLINEQN

```
BEGIN

%*************************************************************************
 COMMENT       A SAMPLE PROGRAM WHICH WILL INVERT A COMPLEX MATRIX
    USING CXLINEQN.  THE INVERSE SHOULD NEVER BE CALCULATED, HOWEVER,
    IF ONLY A SYSTEM OF EQUATIONS IS TO BE SOLVED.
    USES PROCEDURES CXLINEQN AND READMATRIX ;
%*************************************************************************

 $ INCLUDE "MATHLIB/SYMBOL/READMATRIX."
 $ INCLUDE "MATHLIB/SYMBOL/CXLINEQN."

 FILE          READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=14),
               PRINTIT(KIND=PRINTER, BUFFERS=2, MAXRECSIZE=22);
 INTEGER       I,J,N;
 DEFINE        FORI = FOR I := 1 STEP 1 UNTIL N DO #,
               FORJ = FOR J := 1 STEP 1 UNTIL N DO #;
          READ(READIT,/,N);
     BEGIN
 ARRAY         RA,IA,RAINVERSE,IAINVERSE[0:N,0:N], RB,IB[0:N];
 LABEL         EOF;
          READMATRIX(N,N,RA,READIT,PRINTIT,EOF);
          READMATRIX(N,N,IA,READIT,PRINTIT,EOF);
 COMMENT       SET UP THE FIRST COLUMN OF THE IDENTITY AS THE RHS ;
          RB[1] := 1;
          CXLINEQN(N,RA,IA,RA,IA,RB,IB,RAINVERSE[1,*],IAINVERSE[1,*],
                                                        TRUE,FALSE);
          FOR I := 2 STEP 1 UNTIL N DO
          BEGIN
             RB[I] := 1;   RB[I-1] := 0;
 COMMENT       SET UP COLUMNS 2 THRU N AS RIGHT-HAND-SIDES, ONE BY ONE ;
             CXLINEQN(N,RA,IA,RA,IA,RB,IB,RAINVERSE[I,*],IAINVERSE[I,*],
                                                        FALSE,FALSE);
          END;
 COMMENT       THE TRANSPOSE OF THE INVERSE OF A IS NOW STORED IN THE
    ARRAYS AINVERSE AND IAINVERSE.  WRITE IT OUT BY COLUMNS TO GET THE
    INVERSE ITSELF ;
          WRITE(PRINTIT,<//"REAL PART OF A INVERSE"/>);
          FORJ  WRITE(PRINTIT,</(7(R18,10,X1))>,FORI RAINVERSE[I,J]);
          WRITE(PRINTIT,<//"IMAGINARY PART OF A INVERSE"/>);
          FORJ  WRITE(PRINTIT,</(7(R18,10,X1))>,FORI IAINVERSE[I,J]);
 EOF:     ;
     END;
 END.

 3,
 1, 1, 2,
 1, 0, -5,
 1, 0, -8,
 0, 2, 10,
 1, 3, 14,
 1, 5, 20,
```

MATRIXOPERATIONS

```
BEGIN

COMMENT       SAMPLE PROGRAM TO COMPUTE THE MATRICES
%*************************************************************************
    B = (I + A*ATRANSPOSE)=1  AND  C = A*(I + A*ATRANSPOSE)=1
    WHERE =1 DENOTES THE INVERSE MATRIX,
    USES PROCEDURES SYMINVERSEOVERA, READMATRIX, AND THE FILE
    MATHLIB/SYMBOL/MATRIXARITHMETIC ;
%*************************************************************************

$ INCLUDE "MATHLIB/SYMBOL/READMATRIX,"
$ INCLUDE "MATHLIB/SYMBOL/SYMINVERSEOVERA,"
$ INCLUDE "MATHLIB/SYMBOL/MATRIXARITHMETIC,"

DEFINE        FORI = FOR I := 1 STEP 1 UNTIL N DO#,
              FORJ = FOR J := 1 STEP 1 UNTIL N DO #;
FILE          READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=14),
              PRINTIT(KIND=PRINTER, BUFFERS=2, MAXRECSIZE=22);
FORMAT        ROW(8(R15,8,X1)/);
INTEGER       I,J,N;
LABEL         SINGULAR,EOF;
          READ(READIT,/,N);
    BEGIN
ARRAY         A,B,C[0:N,0:N];
          READMATRIX(N,N,A,READIT,PRINTIT,EOF);
          ATIMESATRANSPOSE(N,A,B);
COMMENT       USE C AS TEMPORARY STORAGE TO HOLD IDENTITY MATRIX ;
          FOR I := 1 STEP 1 UNTIL N DO C[I,I] := 1;
          MATPLUSMAT(N,B,C,B);
COMMENT       B NOW HOLDS  I + A*ATRANSPOSE ;
          SYMINVERSEOVERA(N,B,SINGULAR);
          WRITE(PRINTIT,<//"MATRIX (I + A*ATRANSPOSE)=1"//>);
          FORI
          WRITE(PRINTIT,ROW,FORJ B[I,J]);
          MATTIMESMAT(N,A,B,C);
          WRITE(PRINTIT,<//"MATRIX A(I + A*ATRANSPOSE)=1"//>);
          FORI
          WRITE(PRINTIT,ROW,FORJ C[I,J]);
    END;
          IF FALSE THEN
SINGULAR:
          WRITE(PRINTIT,<//"MATRIX IS SINGULAR">);
EOF:
END.

3,
2 0 0
1 2 0
1 0 1.41421356237
```

NONSYMEIGENVECTORS

```
BEGIN

%*************************************************************************
 COMMENT       PROGRAM TO FIND ALL N EIGENVALUES AND EIGENVECTORS OF
    A GENERAL REAL MATRIX.
               USES PROCEDURES NONSYMEIGENVECTORS AND READMATRIX.
*************************************************************************};

 DEFINE        FORI = FOR I := 1 STEP 1 UNTIL N DO #,
               FORJ = FOR J := 1 STEP 1 UNTIL N DO #;

 $ INCLUDE "MATHLIB/SYMBOL/READMATRIX."
 $ INCLUDE "MATHLIB/SYMBOL/NONSYMEIGENVECTORS."

 FILE          READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=14),
               PRINTER(KIND = PRINTER, BUFFERS = 2, MAXRECSIZE = 22);
 FORMAT        FT2(F18.11,X5,F18.11), FT3(//);
 FORMAT        EIGVEC(//"EIGENVECTOR",I3,X40,"RESIDUAL"/),
               F4(R18.11,X5,R18.11,X10,E9.2,X5,E9.2);
 FORMAT        VAL(/"EIGENVALUES"//);
 INTEGER       I,J,N,N1;
 REAL          T1,T2,LAMBDAR,LAMBDAI;
 LABEL         START,EOF;
 START:
         READ(READIT,<I3>,N);
      BEGIN
 ARRAY         A,A1,U[0:N,0:N],RTR,RTI[0:N];
 ARRAY         X,Y,RESIDR,RESIDI[0:N];
 REAL PROCEDURE    INNERPROD(X,Y);
 ARRAY  X,Y[0];
     BEGIN
 INTEGER       I;
 REAL          S;
          WHILE I:=I+1 LEQ N DO S := S + X[I]*Y[I];
          INNERPROD := S;
      END  INNERPROD;
          READMATRIX(N,N,A,READIT,PRINTER,EOF);
          N1 := N+1;
          FORI  WRITE(A1[I,*],N1,A[I,*]);
          T1 := TIME(1);
          NONSYMEIGENVECTORS(N,A1,RTR,RTI,U);
          T2 := TIME(1);
          WRITE(PRINTER,VAL);
          FOR I := 1 STEP 1 UNTIL N DO
          WRITE(PRINTER,FT2,RTR[I],RTI[I]);
          WRITE(PRINTER,FT3);
```

```
          FOR I := 1 STEP 1 UNTIL N DO
          BEGIN
             WRITE(PRINTER,EIGVEC,I);
             IF RTI[I] = 0 THEN
             BEGIN
                LAMBDAR := RTR[I];
                FORJ X[J] := U[J,I];
COMMENT        COMPUTE THE RESIDUAL VECTOR A*X = LAMBDA*X ;
                FORJ RESIDR[J] := INNERPROD(A[J,*],X) - LAMBDAR*X[J];
                WRITE(PRINTER,<R18,11,X33,E9,2>,FORJ [X[J],RESIDR[J]]);
             END  ELSE
             BEGIN
COMMENT        COMPLEX EIGENVALUE ;
                LAMBDAR := RTR[I];   LAMBDAI := RTI[I];
                FORJ X[J] := U[J,I];
                FORJ Y[J] := U[J,I+1];
                FORJ RESIDR[J] := INNERPROD(A[J,*],X) - LAMBDAR*X[J]
                                                      + LAMBDAI*Y[J];
                FORJ RESIDI[J] := INNERPROD(A[J,*],Y) - LAMBDAR*Y[J]
                                                      - LAMBDAI*X[J];
                WRITE(PRINTER,F4,FORJ [X[J],Y[J],RESIDR[J],RESIDI[J]]);
COMMENT        NOW DO COMPLEX CONJUGATE ;
                I := I+1;
                WRITE(PRINTER,EIGVEC,I);
                WRITE(PRINTER,F4,FORJ [X[J],-Y[J],RESIDR[J],-RESIDI[J]]);
             END;
          END;
       END;
          WRITE(PRINTER,<//"ELAPSED TIME FOR EIGENVALUE-EIGENVECTOR "
                "CALCULATION IN SEC. = ",F10.2//>,(T2-T1)/60);
          GO TO START;
EOF:
END.

 4
-2   2   2   2
-3   3   2   2
-2   0   4   2
-1   0   0   5
```

SYMBANDEIGENVALUE

```
BEGIN

%*************************************************************************
 COMMENT       PROGRAM TO FIND ALL THE EIGENVALUES OF A BANDED
    SYMMETRIC MATRIX.  M IS THE NUMBER OF DIAGONALS BELOW (OR ABOVE)
    THE MAIN DIAGONAL AND N IS THE ORDER OF THE MATRIX.
    USES PROCEDURE SYMBANDEIGENVALUE ;
%*************************************************************************

 $ INCLUDE "MATHLIB/SYMBOL/SYMBANDEIGENVALUE."

 FILE          READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=14),
               PRINTIT(KIND=PRINTER, BUFFERS=2, MAXRECSIZE=22);
 INTEGER       I,J,N,M;
 LABEL         START,EOF;
 START:
          READ(READIT,/,N,M) [EOF];
          WRITE(PRINTIT,</"ORDER =",I2,X5,"BANDWIDTH =",I2//>,N,M);
      BEGIN
 ARRAY         A[0:N,0:M], VALUES[0:N];
 COMMENT       READMATRIX CANNOT BE USED HERE BECAUSE THE ZERO-TH
    COLUMN OF A IS USED ;
          FOR I := 1 STEP 1 UNTIL N DO
          READ(READIT,/,FOR J := 0 STEP 1 UNTIL M DO A[I,J]);
          WRITE(PRINTIT,</"INPUT MATRIX"//>);
          FOR I := 1 STEP 1 UNTIL N DO
          WRITE(PRINTIT,<(6(R18,10,X2))/>,
                          FOR J := 0 STEP 1 UNTIL M DO A[I,J]);
          SYMBANDEIGENVALUE(N,M,A,VALUES);
          WRITE(PRINTIT,<///"EIGENVALUES"//>);
          WRITE(PRINTIT,<R20,11>,FOR I := 1 STEP 1 UNTIL N DO VALUES[I]);
      END;
 COMMENT       THE EXACT EIGENVALUES FOR THE TEST MATRIX SUPPLIED ARE
    GIVEN BY 16 * (SIN(K*PI/(2*N+2)))**4, FOR K = 1,2, ...., N ;
          WRITE(PRINTIT,<//"EXACT EIGENVALUES FOR TEST MATRIX"//>);
          FOR I := 1 STEP 1 UNTIL N DO
          WRITE(PRINTIT,<F18.11>,16*(SIN(I*3.14159265359/(2*N+2)))**4);
          GO TO START;
 EOF:
 END.

  7, 2,
 0, 0, 5,
 0, -4, 6,
 1, -4, 6,
 1, -4, 6,
 1, -4, 6,
 1, -4, 6,
 1, -4, 5,
```

COMPLEXEIGENVALUES

```
 BEGIN

%*************************************************************************
 COMMENT      PROGRAM TO FIND ALL N EIGENVALUES OF A COMPLEX MATRIX,
    USES PROCEDURES COMPLEXEIGENVALUES AND READMATRIX ;
%*************************************************************************

 $ INCLUDE "MATHLIB/SYMBOL/READMATRIX."
 $ INCLUDE "MATHLIB/SYMBOL/COMPLEXEIGENVALUES."

 FILE          READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=14),
               PRINT  (KIND=PRINTER, BUFFERS=2,MAXRECSIZE=22);
 FORMAT        FT1(R18.11,X5,R18.11),  EIGEN(/"EIGENVALUES",//);
 INTEGER       I,J,N;
 REAL          T1,T2;
 LABEL         START,EOF;
 START:
         READ(READIT,/,N) [EOF];
         WRITE(PRINT,<//"ORDER =",I3/>,N);
     BEGIN
 ARRAY         AR,AI[0:N,0:N], WR,WI[0:N];
         READMATRIX(N,N,AR,READIT,PRINT,EOF);
         READMATRIX(N,N,AI,READIT,PRINT,EOF);
         T1 := TIME(2);
         COMPLEXEIGENVALUES(N,AR,AI,WR,WI);
         T2 := TIME(2);
         WRITE(PRINT,EIGEN);
         WRITE(PRINT,<R18.11,X5,R18.11>,
                     FOR I := 1 STEP 1 UNTIL N DO [WR[I],WI[I]]);
         WRITE(PRINT,<//"ELAPSED PROC. TIME IN SEC. =",F8.2/>,
                     (T2-T1)/60);
         GO TO START;
     END;
 EOF:
END.

 4,
 5, 5,-6,-7,
 3, 6,-5,-6,
 2, 3,-1,-5,
 1, 2,-3, 0,
 9, 5,-6,-7,
 3,10,-5,-6,
 2, 3, 3,-5,
 1, 2,-3, 4,
```

REALPOLYZEROFINDER

```
BEGIN

%*************************************************************************
 COMMENT      SAMPLE PROGRAM WHICH WILL CALCULATE THE ZEROES OF THE
    REAL POLYNOMIAL P**2 + Q, WHERE P AND Q ARE INPUT POLYNOMIALS.
    USES PROCEDURES POLYMUL, POLYADD, AND REALPOLYZEROFINDER ;
%*************************************************************************

 $ INCLUDE "MATHLIB/SYMBOL/REALPOLYZEROFINDER."
 $ INCLUDE "MATHLIB/SYMBOL/POLYMUL."
 $ INCLUDE "MATHLIB/SYMBOL/POLYADD."

 FILE          READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=14),
               PRINTER(KIND=PRINTER, BUFFERS=2, MAXRECSIZE=22);
 FORMAT        POLY(R10.4,X7,"+ ", 5(R10.4,"*X**",I1,"  + ")/
                    4(R10.4,"*X**",I1,"  + "), 2(R10.4,"*X**",I2," + ")/
                   (6(R10.4,"*X**",I2," + ")));
 INTEGER       I,DEGREEP,DEGREEQ,DEGREER;
 ARRAY         COEFP,COEFQ,ZEROR,ZEROI[0:25], COEFR[0:50];
         READ(READIT,/,DEGREEP,
                       FOR I:=0 STEP 1 UNTIL DEGREEP DO COEFP[I]);
         WRITE(PRINTER,<//"INPUT POLYNOMIAL P"/>);
         WRITE(PRINTER,POLY,COEFP[DEGREEP],
         FOR I := DEGREEP-1 STEP -1 UNTIL 0 DO [COEFP[I],DEGREEP-I]);
         READ(READIT,/,DEGREEQ,
                       FOR I:=0 STEP 1 UNTIL DEGREEQ DO COEFQ[I]);
         WRITE(PRINTER,<//"INPUT POLYNOMIAL Q"/>);
         WRITE(PRINTER,POLY,COEFQ[DEGREEQ],
         FOR I := DEGREEQ-1 STEP -1 UNTIL 0 DO [COEFQ[I],DEGREEQ-I]);
         POLYMUL(DEGREEP,COEFP,DEGREEP,COEFP,DEGREER,COEFR);
         POLYADD(DEGREER,COEFR,DEGREEQ,COEFQ,DEGREER,COEFR);
         WRITE(PRINTER,<//"RESULTANT POLYNOMIAL"/>);
         WRITE(PRINTER,POLY,COEFR[DEGREER],
         FOR I := DEGREER-1 STEP -1 UNTIL 0 DO [COEFR[I],DEGREER-I]);
         REALPOLYZEROFINDER(DEGREER,COEFR,ZEROI,ZEROR);
         WRITE(PRINTER,<//X20,"ZEROES"//
                          X5,"REAL PART",X20,"IMAGINARY PART"/ >);
         WRITE(PRINTER,<E20.10,X10,E20.10>,
               FOR I:=1 STEP 1 UNTIL DEGREER DO [ZEROR[I],ZEROI[I]]);
 END.

 2,  1, -1, 2,
 3,  3, 0, 0, -2,
```

COMPLEXPOLYZEROFINDER

```
BEGIN

%*************************************************************************
 COMMENT      SAMPLE PROGRAM TO FIND THE ROOTS OF A COMPLEX POLYNOMIAL.
    THE REAL PARTS OF THE COEFFICIENTS,BEGINING WITH THE HIGHEST ORDER
    COEFFICIENT, MUST BE ENTERED FIRST, FOLLOWED BY THE IMAGINARY PARTS
    USES PROCEDURE COMPLEXPOLYZEROFINDER ;
%*************************************************************************

 $ INCLUDE "MATHLIB/SYMBOL/COMPLEXPOLYZEROFINDER."

 FILE          CARDREADER(KIND=READER, BUFFERS=2, MAXRECSIZE=14),
               PRINTER(KIND=PRINTER, BUFFERS=2, MAXRECSIZE=22);
 FORMAT        POLY(2(4(R10,4," +I*",R10,4," *Z**",I1,"  +")/),
                      2(R10,4," +I*",R10,4," *Z**",I1,"  +"),
                      2(R10,4," +I*",R10,4," *Z**",I2," +")/
                     (4(R10,4," +I*",R10,4," *Z**",I2," +")));
 FORMAT        ROOTS(E18.11,X5,E18.11);
 INTEGER       I,DEGREE;
 ARRAY         COEFR,COEFI,ZEROR,ZEROI[0:25];
               READ(CARDREADER, /,DEGREE,FOR I := 0 STEP 1 UNTIL DEGREE
                                  DO COEFR[I]);
               READ(CARDREADER, /,       FOR I := 0 STEP 1 UNTIL DEGREE
                                  DO COEFI[I]);
               WRITE(PRINTER,POLY,
               FOR I := DEGREE STEP -1 UNTIL 0 DO [COEFR[I],COEFI[I],
                                                   DEGREE-I]);
               COMPLEXPOLYZEROFINDER(DEGREE,COEFR,COEFI,ZEROR,ZEROI);
               WRITE(PRINTER,<//X20,"ZEROES"//
                             X5,"REAL PART",X20,"IMAGINARY PART"/>);
               WRITE(PRINTER,ROOTS,FOR I := 1 STEP 1 UNTIL DEGREE DO
                                    [ZEROR[I],ZEROI[I]]);
END.

4,  1, -10, 29, -20, -10,
    0, -4, 30, -66,  40,
```

DIFEQNS

```
 BEGIN

%*************************************************************************
 COMMENT       EXAMPLE OF USE OF DIFEQNS TO SOLVE THE 2ND ORDER
    EQUATION  D2Y/DX2 + Y = 0,  THE OUTPUT CAN BE DIRECTED TO A DISK
    FILE, FOR EXAMPLE, BY USING THE LABEL EQUATION CARD
<I> FILE PRINTER(KIND=DISK, TITLE=DIFEQNS/OUTPUT, MAXRECSIZE=22, =
                 BLOCKSIZE=22, SAVEFACTOR=5)
    TO OVERRIDE THE PRINTER FILE DECLARED INSIDE PROCEDURE DIFEQNS ;
%*************************************************************************

 COMMENT       DIFEQNS HAS BEEN LIMITED TO SYSTEMS OF 10 OR FEWER
    EQUATIONS BECAUSE IT MAY NOT BE EFFICIENT FOR LARGER SYSTEMS,
    THIS RESTRICTION CAN BE REMOVED BY CHANGING THE SETTING OF NMAX
    INSIDE THE PROCEDURE ;
 PROCEDURE     XFX(X,Y,DY);
 VALUE X;      REAL X;
 ARRAY         Y,DY[0];
     BEGIN
        DY[1] := Y[2];
        DY[2] := -Y[1];
     END  XFX;

 $ INCLUDE "MATHLIB/SYMBOL/DIFEQNS."

 FILE          READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=14);
 FILE          LINE(KIND=PRINTER, BUFFERS=2,MAXRECSIZE=22);
 FORMAT        JUMPOUT(/"ERROR EXIT TAKEN");
 INTEGER       I,N;
 REAL          SP,XI,XF,HI,EPS;
 ALPHA         ERRCRITERION;
 ARRAY         Y[0:10];
 LABEL         START,EOF,EXIT;
         READ(READIT,/,N);
 START:
         READ(READIT,/,FOR I := 1 STEP 1 UNTIL N DO Y[I]) [EOF];
         READ(READIT,/,SP,XI,XF,HI,EPS);
         READ(READIT,<A2>,ERRCRITERION);
         DIFEQNS(SP,XFX,N,XI,XF,Y,HI,EPS,ERRCRITERION,EXIT);
         GO TO START;
         IF FALSE THEN
 EXIT:   WRITE(LINE,JUMPOUT);
 EOF:
 END.

 2,
 1, 0,
 .2, 0, 10, .2, @-9,
RE
 1, 0,
 0, 0, 10, .1, @-9,
RE
```

SIMPLEDIFEQNS

```
     BEGIN
 COMMENT       SAMPLE PROGRAM ILLUSTRATING THE USE OF A STOPPING
    CONDITION IN PROCEDURE SIMPLEDIFEQNS.  THE FILE MATHLIB/SYMBOL/
    SIMPLEDIFEQNS MUST FIRST BE PATCHED WITH THE DECK

**************************************************************************
<I> COMPILE MATHLIB/NEWSIMPLEDIFEQNS ALGOL LIBRARY
<I> ALGOL FILE TAPE = MATHLIB/SYMBOL/SIMPLEDIFEQNS
<I> ALGOL FILE NEWTAPE = MATHLIB/SYMBOL/NEWSIMPLEDIFEQNS
<I> BCL
<$> SET NEW MERGE SEQ
 BOOLEAN PROCEDURE    ENDTEST(Y)                                    00237000
 ARRAY  Y[0]                                                        00238000
          ENDTEST := ABS(Y[1])+ABS(Y[2])+ABS(Y[3]) LEQ @-3          00239000
          WHILE NOT ENDTEST(YSTART) DO                              00273000
<I> END
**************************************************************************

    IN ORDER TO INSERT THE STOPPING CONDITION GIVEN BY 00239000, WHICH
    CAUSES EXECUTION TO TERMINATE WHEN
         ABS(Y[1]) + ABS(Y[2]) + ABS(Y[3]) LEQ @-3;

 PROCEDURE     XFX(X,Y,DY);
 VALUE  X;    REAL  X;
 ARRAY  Y,DY[0];
     BEGIN
          DY[1] := Y[2];
          DY[2] := Y[3];
          DY[3] := -(6*Y[3] + 11*Y[2] + 6*Y[1]);
 COMMENT       THIS SYSTEM REPRESENTS THE THIRD ORDER EQUATION
    D3Y/DX3 + 6*D2Y/DX2 + 11*D1Y/DX1 + 6*Y = 0;
     END  XFX;

 $ INCLUDE "MATHLIB/SYMBOL/SIMPLEDIFEQNS."

 FILE          READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=22);
 INTEGER       I,N;
 REAL          XI,XF,HI,EPS,ERRCRITERION;
 ARRAY         Y[0:10];
 COMMENT       READ INITIAL VALUES ;
          READ(READIT,/,N);
          READ(READIT,/,FOR I := 1 STEP 1 UNTIL N DO Y[I]);
          READ(READIT,/,XI,XF,HI,EPS);
          READ(READIT,<A2>,ERRCRITERION);
          SIMPLEDIFEQNS(XFX,N,XI,XF,Y,HI,EPS,ERRCRITERION);
 END.

 3,
 6, -14, 36,
 0, 10, .1, @-8,
RE
```

ONEINT

```
BEGIN

%*************************************************************************
COMMENT      SAMPLE PROGRAM WHICH COMPUTES THE INTEGRAL OF THE
   FUNCTION GIVEN BY THE TYPED PROCEDURE F WITH EACH OF THE THREE
   PROCEDURES SIMPSONINT, ROMBERGINT, AND BULIRSCHINT        ;
%*************************************************************************

REAL PROCEDURE    F(X);
VALUE  X;   REAL  X;
        F := X**3 * SIN(X)/(1+X*X) + 1.5 * SINH(X) * EXP(ABS(X));

$ INCLUDE "MATHLIB/SYMBOL/SIMPSONINT."

FILE          READIT(KIND=READER, BUFFERS=2, MAXRECSIZE=14),
              PR(KIND=PRINTER, BUFFERS=2, MAXRECSIZE=22);
FORMAT        TYME( "ELAPSED TIME FOR PROCEDURE IN SEC. = ",F8.2/),
              SIMP(/"SIMPSONINT ANSWER = ",R18.11/),
              ROM(/"ROMBERGINT ANSWER = ",R18.11/),
              BUL(/"BULIRSCHINT ANSWER = ",R18.11,X3,"EST. ERROR = ",
                                                    E9.2/);
REAL          A,B,EPS,INT,ERR,S;
REAL          T1,T2,T3,T4,T5,T6;
LABEL         START,QUIT;
START:  READ(READIT,/,A,B,EPS) [QUIT];
      WRITE(PR,</////"A =",R18.10,X5,"B =",R18.10,X5,"EPS =",E10.2/>,
                                                    A,B,EPS);
        T1 := TIME(1);
        S := SIMPSONINT(F,A,B,EPS);
        T2 := TIME(1);
        WRITE(PR,SIMP,S);
        WRITE(PR,TYME,(T2-T1)/60);
        T3 := TIME(1);
        S := ROMBERGINT(F,A,B,EPS);
        T4 := TIME(1);
        WRITE(PR,ROM,S);
        WRITE(PR,TYME,(T4-T3)/60);
        T5 := TIME(1);
        BULIRSCHINT(F,A,B,EPS,INT,ERR);
        T6 := TIME(1);
        WRITE(PR,BUL,INT,ERR);
        WRITE(PR,TYME,(T6-T5)/60);
        GO TO START;
QUIT:
        WRITE(PR,</////"****************************************">);
        WRITE(PR,<"EXACT ANSWER = ",R18.11>,2*ARCTAN(4));
        WRITE(PR,<   "****************************************">);
END.

-4, 4, @-6,
-4, 4, @-7,
-4, 4, @-8,
-4, 4, @-9,
```

MULINT

```
BEGIN

%*************************************************************************
 COMMENT      SAMPLE PROGRAM WHICH COMPUTES SEVERAL DOUBLE AND TRIPLE
    INTEGRALS,
    USES PROCEDURES SIMPS, DOUBLEINTEGRAL AND TRIPLEINTEGRAL ;
%*************************************************************************

 REAL PROCEDURE    F(X,Y);
 VALUE  X,Y;   REAL  X,Y;
         F := SQRT(1-X**2-Y**2);
 REAL PROCEDURE    G(X,Y);
 VALUE  X,Y;   REAL  X,Y;
         G := X*X*Y*Y + 3*X*Y;
 REAL PROCEDURE    H(X,Y,Z);
 VALUE  X,Y,Z;   REAL  X,Y,Z;
         H := EXP(X+Y+Z);

 $ INCLUDE "MATHLIB/SYMBOL/SIMPS."
 $ INCLUDE "MATHLIB/SYMBOL/DOUBLEINTEGRAL."
 $ INCLUDE "MATHLIB/SYMBOL/TRIPLEINTEGRAL."

 FILE          PR(KIND=PRINTER, BUFFERS=2, MAXRECSIZE=22);
 FORMAT        TYME("PROCESSOR TIME = ",F10.4///);
 REAL          T1,T2,X,Y,Z,S;
 REAL          PI;
         PI := 3.141592653590;
 COMMENT       THIS CALL ON SIMPS WILL GIVE THE AREA OF THE UNIT
    SPHERE,  THE UPPER LIMIT FOR Y COMES FROM THE EQUATION OF THE
    UNIT CIRCLE:  Y = SQRT(1-X**2),  THE CALL
    SIMPS(X,0,1,@-6,SIMPS(Y,0,SQRT(1-X**2),,5@-6,SQRT(1-X**2-Y**2)))
    WOULD SERVE EQUALLY WELL AND BE FASTER ;
         T1 := TIME(2);
         S := 8*SIMPS(X,0,1,@-6,SIMPS(Y,0,SQRT(1-X**2),,5@-6,F(X,Y)));
         T2 := TIME(2);
         WRITE(PR,<"SIMPS ANSWER = ",R18.11>,S);
         WRITE(PR,<"EXACT ANSWER = ",R18.11>,4*PI/3);
         WRITE(PR,TYME,(T2-T1)/60);
 COMMENT       DOUBLEINTEGRAL AND TRIPLEINTEGRAL MAY ONLY BE USED FOR
    RECTANGULAR REGIONS ;
         T1 := TIME(2);
         S := DOUBLEINTEGRAL(G,0,1,1,3,@-6);
         T2 := TIME(2);
         WRITE(PR,<"DOUBLEINTEGRAL ANSWER = ",R18.11>,S);
         WRITE(PR,<"          EXACT ANSWER = ",R18.11>,26/9 + 6);
         WRITE(PR,TYME,(T2-T1)/60);
         T1 := TIME(2);
         S := TRIPLEINTEGRAL(H,0,1,0,1,-1,1,@-6);
         T2 := TIME(2);
         WRITE(PR,<"TRIPLEINTEGRAL ANSWER = ",R18.11>,S);
         WRITE(PR,<"      COMPUTER ANSWER = ",R18.11>, (EXP(1)-1)**2 *
                                                 (EXP(1)-EXP(-1)));
         WRITE(PR,TYME,(T2-T1)/60);
 END.
```

APPENDIX B

BIBLIOGRAPHY

Ahlberg, J. H. 1971. Spline Approximation as an Engineering Tool. In Computer-Aided Engineering: Proceedings of The Symposium Held at The University of Waterloo, May 11-13, 1971. SM Study No. 5. ed. G. M. L. Gladwell. Ontario, Canada: Solid Mechanics Division, University of Waterloo.

Ahlberg, J. H.; Nilson, E. N.; and Walsh, J. L. 1967. The Theory of Splines and Their Applications. New York: Academic Press.

Bauer, F. L.; Rutishauser, H.; and Stiefel, E. 1963. New Aspects in Numerical Quadriture. In Experimental Arithmetic, High Speed Computing, and Mathematics. Proc. of Symposia in Applied Mathematics. 15: 199-218. Providence, R.I.: Amer. Math. Society.

Bjorck, A. and Golub, G. 1967. Iterative Refinement of Linear Least Square Solution by Householder Transformation. BIT 7: 322-337.

Bowdler, H.; Martin, R. S.; Reinsch, C.; and Wilkinson, J. H. 1968. The QR and QL Algorithms for Symmetric Matrices. Numerische Mathematik 11: 293-306.

Bracewell, R. 1965. The Fourier Transform and Its Applications. New York: McGraw Hill.

Brown, K. M. 1969. A Quadratically Convergent Newton-Like Method Based on Gaussian Elimination. SIAM J. Numerical Analysis 6: 560-569.

Bulirsch, R. 1965. Numerical Calculations of Elliptic Integrals and Elliptic Functions. Numerische Mathematik 7: 78-90.

Bulirsch, R. and Stoer, J. 1966. Numerical Treatment of Ordinary Equations by Extrapolation Methods. Numerische Mathematik 8: 1-13.

Bulirsch, R. and Stoer, J. 1967. Numerical Quadrature by Extrapolation. Numerische Mathematik 9: 271-278.

Fiacco, A. V. and McCormick, G. P. 1968. Nonlinear Programming: Sequential Unconstrained Minimization Techniques. New York: John Wiley and Sons.

Forsythe, G. E. and Moler, C. 1967. Computer Solution of Linear Algebraic Systems. Englewood Cliffs, N.J.: Prentice-Hall.

Fox, L. 1951. Practical Solution of Linear Equations and Inversion of Matrices. In Nat. Bur. Stand. AMS 39 ed. O. Taussky. pp. 1-54.

Gautschi, W. 1964. ALGORITHM 236, Bessel Functions of the First Kind. Comm. ACM 7: 479-480.

Greville, T. N. E. 1967. Spline Functions, Interpolation, and Numerical Quadrature. Mathematical Methods for Digital Computers, Vol. 2. New York:John Wiley and Sons.

Hooke, R. and Jeeves, T. A. 1962. Direct Search Solutions of Numerical and Statistical Problems. Journal of ACM 8:212-229.

Isaacson, I. and Keller, H. B. 1966. Analysis of Numerical Methods. New York:John Wiley and Sons.

Jenkins, M. A. 1969. Three Stage Variable-Shift Iterations for the Solution of Polynomial Equations with a Posteriori Error Bounds for the Zeros. Technical Report No. CS 138. Stanford, Calif.:Computer Science Dept., Stanford University.

Jenkins, M. A. and Traub, J. F. 1972. ALGORITHM 419, Zeros of a Complex Polynomial. Comm. ACM 15:97-100.

Keefer, D. L. and Gottfried, B. S. 1970. Differential Constraint Scaling in Penalty Function Optimization. AIIE Transactions 2:281-289.

Lloyd, P. 1966. ALGORITHM 278, GRAPH PLOTTER. Comm. ACM 9:88.

Lyness, J. N. 1969. Notes on the Adaptive Simpson Quadriture Routine. Journal of ACM 16:483-495.

Martin, R. S.; Peters, G.; and Wilkinson, J. H. 1970. The QR Algorithm for Real Hessenberg Matrices. Numerische Mathematik 14:219-231.

-----.1966. Iterative Refinement of the Solution of a Positive Definite System of Equations. Numerische Mathematik 8:203-216.

-----.1965. Symmetric Decomposition of a Positive Definite Matrix. Numerische Mathematik 7:362-383.

Martin, R. S.; Reinsch, C.; and Wilkinson, J. H. 1968. Householder's Tridiagonalization of a Symmetric Matrix. Numerische Mathematik 11:181-195.

Martin, R. S. and Wilkinson, J. H. 1968. The Modified LR Algorithm for Complex Hessenberg Matrices. Numerische Mathematik 12:369-376.

-----.1965. Symmetric Decomposition of Positive Definite Band Matrices. Numerische Mathematik 7:355-361.

McKeemer, W. M. 1962. ALGORITHM 145, Adaptive Numerical Integration by Simpson's Rule. Comm. ACM 5:604.

-----.1962. ALGORITHM 146, Multiple Integration. Comm. ACM 5:604.

Moler, C. B. 1972. Matrix Computation with FORTRAN and Paging. Comm. ACM 15:268-270 and 274.

Muller, D. E. 1956. A Method for Solving Algebraic Equations Using an Automatic Computer. Math. Tables Aids Comput. 10:205-208.

Nilson, E. N. 1970. Cubic Splines on Uniform Meshes. Numerical Mathematics 13:255-258.

Olynyk, F. 1964. ALGORITHM 233, Simpson's Rule for Multiple Integration. Comm. ACM 7:348.

Pankiewicz, W. 1968. ALGORITHM 337, Calculation of a Polynomial and Its Derivative Values by Horner Scheme. Comm. ACM 11:663.

Peters, G. and Wilkinson, J. H. 1970. Eigenvectors of Real and Complex Matrices by LR and QR Triangularizations. Numerische Mathematik 16:181-204.

Reinsch, C. and Bauer, F. L. 1968. Rational QR Transformation with Newton Shift for Symmetric Matrices. Numerische Mathematik 11:264-272.

Reinsch, C. and Wilkinson, J. H. eds. 1971. Linear Algebra. Handbook for Automatic Computation. Vol. 2. New York:Springer-Verlag.

Schwarz, H. R. 1968. Tridiagonalization of a Symmetric Band Matrix. Numerische Mathematik 12:231-241.

Singleton, R. C. 1967. On Computing the Fast Fourier Transform. Comm. ACM 10:647-654.

Varak, James M. 1967. The Computation of Bounds for the Invariant Subspaces of a General Matrix Operator. Technical Report No. CS 66. Stanford, Calif.:Computer Science Dept., Stanford University.

Wilkinson, J. H. 1963. Rounding Errors in Algebraic Processes. New York:Prentice-Hall

INDEX

ACCSLV, 8-6K
ACCSOLVE, 1-15, 1-17
"Adaptive Numerical Integration by Simpson's Rule," 4-2, 14-1, 14-4, 14-5
Ahlberg, J. H., 9-3, 9-5, 9-7, 9-9, 9-11, 9-13, 9-15, 9-17, 9-19, 10-3 thrcugh 10-5
Aitken's Acceleration Method, 11-5
ALGORITHM 145 Adaptive Numerical Integration by Simpson's Rule, 4-2, 14-1, 14-4, 14-5
ALGORITHM 146 Multiple Integration, 4-6, 4-7, 14-4, 14-5
ALGORITHM 233 Simpscn's Rule for Multiple Integration, 4-5
ALGORITHM 236 Bessel Functions of the First Kind, 5-1 through 5-3, 13-2, 13-3
ALGORITHM 278 GRAPH PLOTTER, 6-2
ALGORITHM 337 Calculation of a Polynomial and Its Derivative Values by Horner Scheme, 2-12, 12-11
ALGORITHM 419 Zeros of a Complex Polynomial, 12-3
AMULBT, 8-30
AMULTR, 8-29
"Analysis of Numerical Methods," 11-6, 11-8
Argonne National Laboratories, 8-7, 8-8, 8-10, 8-12, 8-14, 8-16, 8-18, 8-19, 8-21, 8-23, 8-25
ATIMESATRANSPOSE, 1-49
ATIMESBTRANSPOSE, 1-50

BALANC, 8-11 through 8-14
BALANCE, 1-31, 1-33, 1-34
Balancing, 1-29, 1-31, 1-34, 1-39, 1-41, 8-11, 8-14, 8-15, 8-18
BALBAK, 1-29, 1-30, 1-33, 1-41, 8-13, 8-14
Bartky Transformations, 5-8, 5-10
Bauer, F. L., 4-1, 8-23, 8-25, 14-3
Bell Telephone Laboratories, 3-2, 3-6, 10-1, 10-2
Bessel Functions, 5-1 thrcugh 5-3, 13-1, 13-3
"Bessel Functions of the First Kird," 5-1 through 5-3, 13-2, 13-3
BESSFJ, 13-3
Bicubic Spline Interpolation, 9-1
BITREV2, 15-5
Bjorck, Ake, 1-15, 1-17, 8-6H, 8-6K
Bowdler, H., 1-35, 1-36, 1-38, 8-8, 8-9
Bracewell, R., 15-2, 15-3
Brown, K. M., 11-2
BULINT, 14-2
Bulirsch, R., 3-2, 4-3, 4-4, 5-4 throigh 5-10, 10-1, 10-2, 14-2
BULIRSCHINT, 4-3, 7-6

"Calculatior of a Pclynomial and Its Derivative Values by Horner Scheme," 2-12, 12-11
Cardinal Splines, 9-2
CBABK2, 8-17, 8-18
CBAL, 8-15 through 8-18
CEIGVC, 8-17
CEIGVL, 8-15
CEL, 5-10
CEL1, 5-5
CEL2, 5-7

Revised 9/23/75 by
PCN 1076981-001

CEL3, 5-9
Cholesky Decomposition, 1-18, 1-24, 8-6, 8-6C, 8-6E, 8-6N, 8-6P, 8-6X
CNRMLZ, 8-17, 8-18
COMHES, 1-39, 1-41, 1-42, 8-15 through 8-18
COMLR, 1-39, 1-43, 1-44, 8-15, 8-16
COMLR2, 1-30, 1-41, 8-17, 8-18
Compatible Procedures, FORTRAN, 7-1
COMPLEXBALANCE, 1-39, 1-41, 1-42
COMPLEXEIGENVALUES, 1-39, 1-44
COMPLEXEIGENVECTORS, 1-41, 1-44
COMPLEXNORMALIZE, 1-42
COMPLEXPOLYZEROFINDER, 2-2, 7-3
"The Computation of Bounds for the Invariant Subspaces of a General Matrix Operator," 1-44
"Computer Solution of Linear Algebraic Systems," 1-3, 1-28, 8-6S
Cooley-Tukey Method, 15-1 through 15-5
CSDDE(D), 9-4, 9-13
CSDDE2, 9-12
CSDDU(D), 9-8, 9-17
CSDDU2, 9-16
CSIE(D), 9-6, 9-15
CSIE2, 9-14
CSIU(D), 9-10, 9-18
CSIU2, 9-18
Cubic Spline of Interpolation, 9-1, 10-3
"Cubic Splines on Uniform Meshes," 9-5, 9-7
CXBESJ, 13-1
CXDECOMPOSE, 1-26, 1-27, 8-6R
CXIMPROVE, 1-27, 8-6R
CXJBES, 5-3
CXLINEQN, 1-26, 1-27
CXLINQ, 8-6R
CXPOLY, 12-3
CXSOLVE, 1-27, 8-6R

DBLINT, 14-4
DECOMP, 8-2, 8-3, 8-5
DECOMPOSE, 1-1 through 1-13, 8-6U, 8-6W
"Description of ALGOL 60," 4-1
DESP, 10-1
DESUB (FORTRAN), 3-2, 3-6, 10-1, 10-2
DETERMINANT, 1-1, 1-2, 1-5, 1-7, 1-9 through 1-12
Determinants, 1-1, 8-1
DETLINEQN, 1-5
DETLINEQNIMPRV, 1-7
DETLINEQNS, 1-9
DETOFA, 1-12, 8-6W
DIAGTIMESMAT, 1-57
DIFEQNS, 3-1, 3-5, 10-1
"Differential Constraint Scaling in Penalty Function Optimization," 11-13
Differential Equations, 3-1, 10-1
Differertiation, 10-3
"Direct Search Solutions of Numerical and Statistical

Problems," 11-10, 11-13
DMULTM, 8-36
DOT, 8-6T
DOUBLEDOTPROD, 1-6, 1-7, 1-20, 1-22
DOUBLEINTEGRAL, 4-6, 4-7, 7-7
DOUBLEPROD, 1-11
DTLQPR, 8-6M

Eigenvalues, 1-1, 1-29, 1-31, 1-33, 1-35 through 1-37, 1-39, 1-41, 1-43, 8-7
Eigenvectors, 1-1, 1-29, 1-33, 1-36, 1-41, 1-43, 8-7
"Eigenvectors of Real and Complex Matrices by LR and QR Triangularizations," 1-34, 1-42, 8-14, 8-18
EIGVAL, 8-11
EIGVEC, 8-13
EISPACK, 8-7, 8-8, 8-10, 8-12, 8-14, 8-16, 8-18, 8-19, 8-21, 8-23, 8-25
ELIM, 1-4 through 1-12
Elliptic Integrals, 5-4 through 5-10
EL1, 5-4
EL2, 5-6
EL3, 5-8
ENDTEST, 3-6
Equations,
Differertial, 3-1, 10-1
Linear, 1-1, 8-1
Nonlinear, 11-1
ERR, 10-2
Extrema of Functions, 11-9

FBULIR, 7-6
FCXPLY, 7-1, 7-3
FDBINT, 7-7
FFTF, 15-2
FFTR, 15-3
Fiacco, A. V., 11-13
Forsythe, G. E., 1-3, 1-28, 8-6S
FORTRAN,
Compatible Procedures, 7-1
Matrices, 8-1
Numerical Integration, 14-1
Polynomials, 12-1
Special Functiors, 13-1
"The Fourier Transform and Its Applications," 15-2, 15-3
Fourier Trarsforms, 15-1
Fox, L., 1-13, 8-6U
Francis Double-Step QR Algorithm, 1-29, 1-31, 1-33, 8-11, 8-14
FRLPLY, 7-1, 7-2
FRMBRG, 7-4
FSMPSN, 7-5
FTRINT, 7-8
FUNCT, 11-1 through 11-3, 11-5, 11-7, 11-9, 11-11

Revised 9/23/75 by
PCN 1076981-001

Gaussian Elimination (LU Decomposition), 1-1, 1-2 through 1-5, 1-8, 1-10, 1-12 through 1-14, 1-24, 1-26, 1-27, 8-2, 8-3, 8-5, 8-6A, 8-6M, 8-6Q, 8-6R, 8-6T, 8-6U, 8-6W
Gautschi, W., 5-1 through 5-3, 13-1 through 13-3
GCG (Greatest Common Divisor), 2-13, 12-17
Golub, Gene, 1-15, 1-17, 8-6H, 8-6K
Gottfried, B. S., 11-13
GRAPH PLOTTER, 6-2
Greville, T. N. E., 10-3, 10-4

"Handbook for Automatic Computation," 1-18, 1-3C, 8-6A, 8-6D, 8-6F, 8-6Y, 8-7
Hermitian Matrices, 8-7, 8-19, 8-20
Hessenberg,
    Form, 1-29, 1-31, 1-33, 1-34, 1-39, 1-41 through 1-43, 8-11, 8-13 through 8-15, 8-17, 8-18
    Matrices, 1-29, 1-39
Hooke-Jeeves Method, 11-9 through 11-11, 11-13
Hooke, R., 11-10, 11-13
Horner Scheme, 2-12, 12-11
Householder Transformations, 1-14, 1-17, 1-29, 1-31, 1-33, 1-35, 1-36, 8-6H, 8-6K, 8-8, 8-9, 8-11, 8-13, 8-22, 8-24
"Househclder's Tridiagonalization of a Symmetric Matrix," 8-19, 8-21
HQR, 1-31, 8-11, 8-12
HQR2, 1-33, 1-34, 8-13, 8-14
HREGVC, 8-20
HREGVL, 8-19
HTRIBK, 8-20
HTRIDI, 8-19, 8-20

IBESS, 5-2
IMPROVE, 1-1 through 1-3, 1-6, 1-7, 1-11, 1-18
IMPRUV, 8-3
INNERPRCD, 1-4 through 1-12, 1-19 thrcugh 1-22
Integration,
    Multiple, 4-5, 14-4
    Single, 4-1, 10-3, 14-1
Interpolation, 9-1
INVARIANTSUBSPACES, 1-29, 1-30, 1-43
INVDCP, 8-6T
INVEPR, 8-6V
INVERA, 8-6U
INVERS, 8-6U
INVERSE, 1-1 through 1-3, 1-10, 1-13
INVERSECVERA, 1-1, 1-3, 1-13
INVERT, 1-13, 8-6T, 8-6U, 8-6X
INVWITHIMPRV, 1-3, 1-11
Isaacson, E., 11-6, 11-8
"Iterative Refiremert of Linear Least Square Solutions by Householder Trarsformation," 1-15, 1-17, 8-6H, 8-6K
"Iterative Refinement of the Solution of a Positive Definite System of Equations," 8-6F

Jacobi Rotations, 1-29, 1-30, 1-37
Jacobian Zeta Function, 5-6
JBESS, 5-1
Jeeves, T. A., 11-10, 11-13
Jenkins, M. A., 2-1, 2-2, 12-1, 12-3

Keefer, D. L., 11-13
Keller, H. B., 11-6, 11-8

Landen Transformation, 5-4 through 5-7, 5-9
LDU Decomposition, 1-18, 1-25, 8-6C, 8-6E, 8-6N, 8-6P, 8-6X
LEASTSQUARES, 1-14
LEASTSQUARESIMPRV, 1-16
"Linear Algebra," 1-18, 1-30, 8-6A, 8-6D, 8-6F, 8-6Y, 8-7
Linear Equations, 1-1, 8-1
LINEQN, 1-4, 8-2, 8-6N
LINEQNIMPRV, 1-6
LINEQNS, 1-8, 1-9
LINPAC, 1-1, 1-13, 1-18, 1-26, 1-27, 8-6R, 8-6U
LINQNS, 8-6Q
Lloyd, P., 6-2
LLSQPR, 8-6J
LLSQSE, 8-6G
LLSSLV, 8-6H
LNEQNS, 8-5
LNQPRV, 8-3
LR Algorithm, 1-29, 1-30, 1-39, 1-43, 8-7, 8-15
LU Decomposition (See Gaussian Elimination)
Lyness, J. N., 4-2, 14-1

Martin, R. S., 1-18, 1-24, 1-25, 1-32, 1-35, 1-36, 1-38, 1-40, 8-6A, 8-6C, 8-6F, 8-6X, 8-8, 8-9, 8-12, 8-15, 8-19, 8-21
MATCOPY, 1-61
"Mathematical Methods for Digital Computers," 10-3, 10-4
MATMINUSDIAG, 1-55
MATMINUSMAT, 1-46
MATPLUSDIAG, 1-54
MATPLUSMAT, 1-45
Matrices,
    Arithmetic, 1-45, 8-26
    Complex, 1-26, 1-39
    FORTRAN, 8-1
    General, 1-1
    Hermitian, 8-7, 8-19, 8-20
    Hessenberg, 1-29, 1-39
    Inversion, 1-1, 8-1
    Real, 1-1, 1-31
    Rectangular, 1-62
    Square, 1-45
    Symmetric, 1-18, 1-35
"Matrix Computation with FORTRAN and Paging," 8-2, 8-4, 8-5, 8-6T
MATTIMESDIAG, 1-56
MATTIMESMAT, 1-47
MATTIMESVEC, 1-51

Revised 9/23(75 by
PCN 1076981-001

MATVEC, 8-31
McCormick, G. P., 11-13
McKeemar, W. M., 4-2, 4-6, 4-7, 14-1, 14-4, 14-5
MCOPY, 8-40
"A Method for Solving Algebraic Equations Using an Automatic Computer," 11-4
MMINDG, 8-34
MMULTD, 8-35
MMULTM, 8-27
MNEGVL, 8-7, 8-24
"The Modified LR Algorithm for Complex Hessenberg Matrices," 1-40, 8-15
Moler, C. B., 1-3, 1-28, 8-2, 8-4, 8-5, 8-6S, 8-6T
Muller, D. E., 11-4
Muller's Method, 11-3, 11-4
"Multiple Integration," 4-6, 4-7, 14-4, 14-5
MXEGVL, 8-7, 8-22

Naming Conventions, Spline Interpolation, 9-2
NATS, 8-7, 8-8, 8-10, 8-12, 8-14, 8-16, 8-18, 8-19, 8-21, 8-23, 8-25
"New Aspects in Numerical Quadrature," 4-1, 14-3
Nilson, E. N., 9-3, 9-5, 9-7, 9-9, 9-11, 9-13, 9-15, 9-17, 9-19, 10-3 through 10-5
NLNEQN, 11-1
Nonlinear Equations, 11-1
Nonlinear Programming, 11-11
"Nonlinear Programming: Sequential Unconstrained Minimization Techniques," 11-13
NONSYMEIGENVALUES, 1-31
NONSYMEIGENVECTORS, 1-33
NORMALIZE, 1-34
"Notes on the Adaptive Simpson Quadriture Routine," 4-2, 14-1
NRMLZ, 8-13, 8-14
NUMDIF, 10-1, 10-3, 10-5
"Numerical Calculations of Elliptic Integrals and Elliptic Functions," 5-4 through 5-10
Numerical Integration, 4-1, 10-3
"Numerical Quadrature by Extrapolation," 4-4, 14-2
"Numerical Treatment of Ordinary Differential Equations by Extrapolation Methods," 10-2
NUMINT, 10-1, 10-3, 10-4

Olynyk, F., 4-5
"On Computing the Fast Fourier Transform," 15-2, 15-3
OPTHKJ, 11-9
ORTHES, 1-31, 1-33, 1-34, 8-11 through 8-14
ORTRANS, 1-29, 1-33, 1-34, 8-13, 8-14

Pankiewicz, 2-12, 12-11
PENHKJ, 11-11
Peters, G., 1-18, 1-25, 1-32, 1-34, 1-42, 8-6C, 8-6F, 8-6X, 8-12, 8-14, 8-18
PLOTTER, 6-1
POLADD, 12-5
POLDER, 12-9

POLDIV, 12-8, 12-17
POLGCD, 12-17
POLHOR, 12-11
POLINT, 12-10
POLMAD, 12-12
POLMOV, 12-14
POLMUL, 12-7
POLNOR, 12-15
POLPOP, 12-13
POLSOR, 12-16
POLSUB, 12-6
POLYADD, 2-3
POLYDER, 2-8
POLYDIV, 2-6, 2-13
POLYGCD, 2-13
POLYHOR, 2-12
POLYINT, 2-9
POLYMAD, 2-11
POLYMOV, 2-15
POLYMUL, 2-5
Polynomials,
    Expansion in Power Series, 2-12, 12-11
    FORTRAN, 12-1
    Operation on, 2-3, 12-5
    Roots of, 2-1, 12-1
POLYNORM, 2-6, 2-14
POLYPOP, 2-10
POLYSOR, 2-16
POLYSUB, 2-4
POLYVAL, 2-7
Power Series Expansion, 2-12, 12-11
"Practical Solution of Linear Equations and
    Inversion of Matrices," 1-13, 8-6U

QL Algorithm, 1-29
QR Algorithm, 1-29, 8-7
"The QR Algorithm for Real Hessenberg Matrices," 1-32, 8-12
"The QR and QL Algorithms for Symmetric Matrices," 1-35, 1-36, 1-38,
    8-8, 8-9
QR-Decomposition, 1-14, 1-16, 8-6G, 8-6H, 8-6J, 8-6K
"A Quadratically Convergent Newton-Like Method Based
    on Gaussian Eliminations," 11-2

"Rational QR Transformation with Newton Shift for Symmetric
    Matrices," 8-23, 8-25
RATQR, 8-22 through 8-25
READMATRIX, 6-3
READSYMMATRIX, 6-4
REALPOLYZEROFINDER, 2-1, 7-2
RECMATMINUSMAT, 1-63
RECMATPLUSMAT, 1-62
RECMATTIMESMAT, 1-64
RECMMM, 8-42
RECMMU, 8-43

Revised 9/23/75 by
PCN 1076981-001

RECMPM, 8-41
RECURSIMP, 4-2
Reinsch, C., 1-18, 1-30, 1-35, 1-36, 1-38, 8-6A, 8-6D, 8-6F, 8-6Y, 8-8, 8-9, 8-19, 8-21, 8-23, 8-25
RLPOLY, 12-1
RMINM, 8-38
RMINUSMAT, 1-59
RMULTM, 8-39
Romberg Integration, 4-3
ROMBERGINT, 4-1, 7-4
ROMINT, 14-3
Roots of Polynomials, 2-1, 12-1
"Rounding Errors in Algebraic Processes," 11-4, 11-6, 11-8
RPLUSM, 8-37
RPLUSMAT, 1-58
RTIMESMAT, 1-60
Rutishauser, H., 4-1, 14-3

Schwarz, H. R., 1-38
SIMINT, 14-1, 14-5
SIMPLEDIFEQNS, 3-5
SIMPS, 4-5
SIMPSONINT, 4-2, 4-6, 7-5
"Simpson's Rule for Multiple Integration," 4-5
SINCOS, 15-2 through 15-4
Singleton, R. C., 15-2, 15-3
SINGULAR, 1-2, 1-4 through 1-12
SINVRA, 8-6P, 8-6X
SOLVE, 1-1 through 1-11, 8-2 through 8-5, 8-6Q
"Spline Approximation as an Engineering Tool," 9-5, 9-7
"Spline Functions, Interpolation, and Numerical Quadrature," 10-3, 10-4
Stiefel E., 4-1, 14-3
Stoer, J., 3-2, 4-3, 4-4, 10-1, 10-2, 14-2
SYINPR, 8-6P
SYMBANDEIGENVALUES, 1-37
SYMBANDLINEQN, 1-18, 1-23, 1-26, 1-37
SYMBLQ, 8-6
SYMDCP, 8-6C, 8-6E
SYMDECOMPOSE, 1-19 through 1-22, 8-6N
SYMEIGENVALUES, 1-35, 1-37
SYMEIGENVECTORS, 1-36
SYMELIN, 1-19 through 1-22
"Symetric Decomposition of a Positive Definite Matrix," 1-18, 1-25, 8-6X
"Symetric Decomposition of Positive Definite Band Matrices," 1-24, 8-6A, 8-6C, 8-6F
SYMIMP, 8-6E
SYMIMPROVE, 1-20, 1-22
SYMINVERSION, 1-25
SYMINVERSOVERA, 1-18, 1-22, 1-25
SYMINVWITHIMPRV, 1-18, 1-22
SYMLINEQN, 1-18, 1-19
SYMLINEQNIMPRV, 1-18, 1-20
SYMLINEQNS, 1-18, 1-21
SYMLINPAC, 1-18

SYMLNQ, 8-6C
SYMLQP, 8-6E
SYMLQS, 8-6N
SYMMET, 8-6E, 8-6X
SYMMETRIZE, 1-20, 1-22
SYMSLV, 8-6C, 8-6E
SYMSOLVE, 1-19 through 1-22, 8-6N
SYMVAL, 8-8, 8-23, 8-25
SYMVEC, 8-9

Taylor Series, 11-2
"Theory of Splines and Their Applications," 9-3, 9-9, 9-11, 9-13, 9-15, 9-17, 9-19, 10-3 through 10-5
Thomas Algorithm, 9-3, 9-9, 9-11
"Three Stage Variable-Shift Iterations for the Solution of Polynomial Equations with a Posteriori Error Bounds for the Zeros," 2-1, 2-2, 12-1, 12-3
TOFY, 13-1, 13-3
TPLINT, 14-5
TQL1, 8-8, 8-19 through 8-21
TQL2, 8-9, 8-10, 8-20
TRANSP, 8-28
TRANSPOSE, 1-48
Traub, J. F., 2-1, 2-2, 12-1, 12-3
TRBAK1, 8-21
TRED1, 8-8, 8-19, 8-21 through 8-25
TRED2, 8-9, 8-10
TRIANGLEDECOMP, 1-13, 8-6U
"The Tridiagonalization of a Symmetric Band Matrix," 1-38
TRIPLEINTEGRAL, 4-7, 7-8
Truncation, 9-5, 9-7

Varah, James M., 1-44
VECMAT, 8-32
VECPLUSRVEC, 1-53
VECTIMESMAT, 1-52
VPLURV, 8-33

Walsh, J. L., 9-3, 9-9, 9-11, 9-13, 9-15, 9-17, 9-19, 10-3 through 10-5
Wilkinson, J. H., 1-18, 1-24, 1-25, 1-29, 1-30, 1-32, 1-34 through 1-36, 1-38, 1-40, 1-42, 8-6A, 8-6C, 8-6D, 8-6F, 8-6X, 8-6Y, 8-7 through 8-9, 8-12, 8-14, 8-15, 8-18, 8-19, 8-21, 11-4, 11-6, 11-8

XFX, 3-1, 3-5, 10-1, 10-2
XOUTX, 3-3, 10-2

ZEROML, 11-3
ZERONT, 11-7
Zeroes, 11-3
"Zeros of a Complex Polynomial," 12-3
ZEROSS, 11-5

Revised 9/23/75 by
PCN 1076981-001