HYDRO-UNIVERSITY COMPUTING CENTRE

ALGOL PROCEDURES REFERENCE MANUAL

This document contains details of Algol procedures which have been written for and tested on the Elliott 503. Tapes for these procedures are available in the Computing Centre. Users wishing to incorporate these procedures in programs submitted for punching should indicate their requirements on the Algol program sheet thus:

```
TITLE;
begin integer ..................; real...................;

                 LIBRARY LE04

                 LIBRARY NC01

             comment then follows the program text;
                                 :
end;
```

Such instructions will be sufficient to ensure that the appropriate procedures are copied at the places indicated. It will be left to programmers to ensure that the order of copying of requested procedures is a valid order of declaration. This is important in those cases where one procedure uses another as indicated in the procedure descriptions.

The manual also contains an index of HUCC Library procedures, together with an index of other known Algol procedures and their sources. In most cases copies of these are available in the Computing Centre. Procedure tapes, however, are available only for HUCC procedures.

CLASSIFICATIONS

C		COMMERCIAL
	CA	INTEREST
D		INFORMATION HANDLING-DATA PROCESSING
	DH	SORTING
	DP	PLOTTING
	DZ	MISCELLANEOUS
F		FUNCTIONS-EVALUATION OF-ETC
	FB	BESSEL
	FC	COMPLEX
	FD	POWERS AND EXPONENTIALS
	FE	ELLIPTIC INTEGRALS
	FP	POLYNOMIALS-INC CHEBYSHEV ETC
	FS	SPECIAL
	FT	TRIGONOMETRICAL FUNCTIONS
	FZ	MISCELLANEOUS
G		DIFFERENTIAL EQUATIONS
	GA	ORDINARY-NOT LINEAR OR IST ORDER
	GL	LINEAR
	GP	PARTIAL
L		LINEAR ALGEBRA
	LA	CHANGE FORM OF MATRIX
	LB	BOOLEAN MATRICES
	LE	LINEAR EQUATIONS AND INVERSION
	LF	FORM SPECIAL MATRIX
	LG	ARITHMETIC FUNCTIONS-ONE MATRIX
	LH	ARITHMETIC FUNCTIONS-TWO MATRICES
	LL	LATENT ROOTS
	LO	DETERMINANTS
	LR	READ OR INPUT
	LZ	MISCELLANEOUS
M		MATHEMATICAL METHODS
	MC	CURVE AND SURFACE FITTING
	MD	DIFFERENTIATION-MAX AND MIN
	ME	ERROR ANALYSIS
	MG	GEOMETRY
	MH	HARMONIC ANALYSIS
	MR	ROOTS OF EQUATIONS
	MS	INTEGRATION AND SUMMATION
	MT	INTERPOLATION AND DIFFERENCES
	MZ	MISCELLANEOUS
N		INTEGERS AND NUMBER THEORY
	NC	PERMUTATIONS AND COMBINATIONS
	NP	PRIME NUMBERS
	NR	PARTITIONS
	NZ	MISCELLANEOUS
O		OPERATIONAL RESEARCH
	OL	LINEAR PROGRAMMING
	OP	PERT-CRITICAL PATH ANALYSIS
P		PHYSICS
	PH	HEAT
	PN	NUCLEAR ENGINEERING
	PQ	QUANTUM MECHANICS
S		STATISTICS
	SA	ANALYSIS
	SB	SMOOTHING
	SC	CORRELATION
	SM	MOMENTS
	SR	REGRESSION
	SS	STOCHASTIC PROCESSES
	SV	ANALYSIS OF VARIANCE
Z		MISCELLANEOUS
	ZA	COMPILER TECHNIQUES
	ZZ	MISCELLANEOUS

ABBREVIATIONS

ACM	COMMUNICATIONS OF ACM
AIS	ALGOL INFORMATION SHEET
ALY	ALGORYTMY
BIT	NORDISK TIDSKRIFT FOR INF BEHANDLG
CJ	COMPUTER JOURNAL
JCM	JOURNAL OF THE A C M
MCA	MATHEMATISCH CENTRUM AMSTERDAM
NUM	NUMERISCHE MATHEMATIK
ORN	OAK RIDGE NATIONAL LABORATORY
SUC	STANFORD UNIVERSITY CALIFORNIA
BOO1	SELECTED NUMERICAL METHS BY C GRAM
NPL	NATIONAL PHYSICAL LABORATORY

INDEX of HUCC ALGOL LIBRARY PROCEDURES.

DH 01	SORT REAL NUMBERS INTO ASCENDING ORDER
DH 02	LOCATE ELEMENT IN LIST
DH 03	SORT REAL NUMBERS IN ASCENDING ORDER
DH 04	SORT ROWS OR COLUMNS OF MATRIX
DP 01	OUTPUT GRAPH OF VECTOR ELEMENTS
DP 02	OUTPUT PLOT OF ELEMENTS OF TWO VECTORS
FC 01	COMPLEX ASSIGNMENT
FC 02	MULTIPLE COMPLEX ASSIGNMENT
FC 03	COMPLEX MULTIPLICATION
FC 04	COMPLEX DIVISION
FC 05	ASSIMILATE REAL IN COMPLEX OPERATION
FC 06	ASSIMILATE IMAGINARY IN COMPLEX OPERATION
FC 07	COMPLEX CONJUGATE
FC 08	MODULUS OF COMPLEX NUMBER
FC 09	ARGUMENT OF COMPLEX NUMBER
FC 10	POLAR FORM
FC 11	MULTIPLICATION BY IMAGINARY OPERATOR
FC 12	COMPLEX SQUARE
FC 13	COMPLEX RECIPROCAL
FC 14	COMPLEX ROOT
FC 15	COMPLEX LOGARITHM
FC 16	COMPLEX EXPONENTIAL
FC 17	HYPERBOLIC FUNCTIONS
FC 18	COMPLEX SINE
FC 19	COMPLEX COSINE
FC 20	COMPLEX TANGENT
FC 21	COMPLEX INVERSE SINE
FC 22	COMPLEX INVERSE COSINE
FC 23	COMPLEX INVERSE TANGENT
FC 24	COMPLEX POWER OF COMPLEX VARIABLE
FC 25	TEST FOR EVEN INTEGER
FC 26	LOGARITHM OF COMPLEX NUMBER
FP 01	SUM SERIES OF CHEBYSHEV POLYNOMIALS
FS 01	LOGARITHM OF FACTORIAL(n)
GL 01	FOURTH ORDER RUNGE-KUTTA
LE 01	MATRIX INVERSION
LE 02	SOLVE LINEAR EQUATIONS - ONE R.H. SIDE
LE 03	MATRIX DECOMPOSITION Ax=b
LE 04	FORWARD AND BACK SOLUTION Ax=b
LE 05	MATRIC DECOMPOSITION BY CROUTS METHOD
LE 06	TRIANGULAR MATRIX PRODUCT LU=A
LE 07	INVERT QUASI LOWER TRIANGULAR MATRIX
LE 08	INVERT QUASI UPPER TRIANGULAR MATRIX
LE 09	TRIANGULAR MATRIX PRODUCT UL=A
LE 10	procedure ATOLIU

LE 11	**procedure** LIUSOL
LE 12	**procedure** INVL1
LE 13	**procedure** INVU
LE 14	**procedure** UL1TOA
LE 15	SOLVE TRIDIAGONAL LINEAR EQUATIONS
LE 16	DECOMPOSE SYMMETRIC POSITIVE DEFINITE MATRIX
LE 17	INVERT LOWER TRIANGULAR MATRIX IN SITU
LE 18	MATRIX MULTIPLICATION
LE 19	DECOMPOSE BANDMATRIX INTO LOWER AND UPPER TRIANGLES
LE 20	FORWARD SOLUTION AND BACK SUBSTITUTION FOR BAND EQUATION Ax=b
LE 21	SOLVE BAND EQUATIONS Ax=b
LE 22	REARRANGE PERMUTED TRIANGULAR MATRIX
LE 23	**procedure** MOVEL
LE 24	DECOMPOSE SYMMETRIC POSITIVE DEFINITE MATRIX
LE 25	FORWARD AND BACK SOLUTION Ax=b,SYMMETRIC A
LE 26	DECOMPOSE BAND MATRIX WITH PIVOTING
LE 27	FORWARD AND BACK SOLUTION Ax=b,BANDMATRICES
LE 28	DECOMPOSE SYMMETRIC POSITIVE DEFINITE MATRIX
LE 29	FORWARD AND BACKWARD SOLUTION OF Ax=b,SYMMETRIC
LL 01	EIGENVALUES AND EIGENVECTORS OF SYMMETRIC MATRIX
LL 02	SOLUTION OF THE EIGENPROBLEM (A-λB)x=0
LZ 01	GENERATE UNSYMMETRIC TES MATRIX
MC 01	LEAST SQUARES POLYNOMIAL FIT
MD 01	LOCATE MINIMUM OF FUNCTION f(x)
MG 01	CONVERT SEXAGESIMAL ANGLES TO RADIANS
MG 02	CONVERT RADIAN ANGULAR MEASURE TO SEXAGESIMAL UNITS
MG 03	CONVERT CENTESIMAL ANGLES TO RADIANS
MG 04	CONVERT RADIAN ANGULAR MEASURE TO CENTESIMAL UNITS
MG 05	COMPUTE DISTANCE AND GRID BEARING
MG 06	COMPUTE POLAR CO-ORDINATES OF POINT X,Y
MG 07	COMPUTE POINT OF INTERSECTION FROM ANGLES
MG 08	COMPUTE POINT OF INTERSECTION FROM BEARINGS
MG 09	THREE POINT RESECTION BY ANGLES
MG 10	MULTIPLE POINT RESECTION FROM ANGLES
MG 11	THREE POINT RESECTION FROM DISTANCES
MG 12	MULTIPLE POINT RESECTION USING DISTANCES
MG 13	THE INACCESSIBLE BASE PROBLEM
MH 01	COMPUTE FOURIER COEFFICIENTS
MH 02	SUM FOURIER SERIES
MR 01	LOCATE ROOT OF f(x)=0
MS 01	GENERAL SUM SERIES
MS 02	ADAPTIVE SIMPSON INTEGRATION
MS 03	EVALUATE DEFINITE INTEGRAL
MT 01	LAGRANGE INTERPOLATION
MT 02	AITKEN UNEQUAL INTERVAL INTERPOLATION
MT 03	AITKEN EQUAL INTERPOLATION
MT 04	AITKEN Nth ORDER INTERPOLATION
MT 05	ESTIMATION OF DERIVATIVE OF GIVEN FUNCTION - UNEQUAL INTERVALS
MT 06	ESTIMATION OF DERIVATIVE OF GIVEN FUNCTION EQUAL INTERVALS

TITLE	REFERENCES
	CA
PERIODIC PAYMENT AS FN OF UNPAID PRINC	ACM45 4-61 9-63
	DH
SORTING-MATHSORT	ACM23 11-60 5-61
PARTITION-USED BY QUICKSORT ACM64	ACM63 7-61 8-62 8-63
QUICKSORT	ACM64 7-61 8-62 8-63
FIND	ACM65 7-61 8-62 8-63
SORTING PROCEDURES	ACM76 1-62 6-62
TREESORT-SUPERSEDED BY TREESORT 1 ACM143	ACM113 8-62
TREESORT 1	ACM143 12-62
TREESORT 2	ACM144 12-62
SHUTTLE SORT-EXCHANGING PAIRS	ACM175 6-63 10-63 12-63 5-64
SHELLSORT-SHELLS METHOD	ACM201 8-63 6-64
STRINGSORT	ACM207 10-63 10-64
RADIX EXCHANGE ETC	ACM 5-63 P209
SORTING BY DISTRIBUTION COUNTING	ORNL3148
HEAP SORT	ACM 232 6-64
TREESORT 3	ACM245 12-64
SEARCH IN A LIST	JACM 1962(23)
INSERTION IN A LIST	JACM 1962(23)
DELETION FROM A LIST	JACM 1962(24)
SORTING WITH MIN STORAGE	JACM 1962(27)
	DP
COMPUTE CODE STRING TO MOVE PLOTTING PEN	ACM162 4-63 8-63 8-64
PROCEDURES FOR GRAPH PLOTTING	NPL 0003
	DZ
ADD ITEM TO CHAIN LINKED LIST	ACM100 6-62
REMOVE ITEM FROM CHAIN-LINKED LIST	ACM101 6-62
ENLARGEMENT OF A GROUP	ACM136 11-62
LOCATE VECTOR IN LEXICOG ORDERED LIST	ACM151 2-63
	FB
BESSEL FN 1-SERIES EXPANSION	ACM5 4-60
BESSEL FN 1-ASYMPTOTIC EXPANSION	ACM6 4-60
BESSEL FN FOR SET OF INTEGER ORDERS	ACM21 11-60
RICCATI-BESSEL FNS OF 1ST AND 2ND KIND	ACM22 11-60
BESSEL FUNCTIONS BY RECURSION	ACM44 4-61
EVALUATE BER OR BEI FN	ACM57 4-61 7-62 8-62
Q BESSEL FUNCTIONS	ACM214 11-63 6-64
BESSEL FUNCTIONS-CHEBYSHEV-CLENSHAW	AIS13
BESSEL FCNS 1 ST KIND	ACM236 8-64
Q BESSEL FCN	ACM228 5-64
	FC
COMPLEX EXPONENTIAL INTEGRAL	ACM13 7-60 2-61
EXPONENTIAL OF COMPLEX NUMBER	ACM46 4-61 6-62
LOG OF COMPLEX NUMBER	ACM48 4-61 6-62 7-62 8-64
NTH ROOTS OF A COMPLEX NUMBER	ACM53 4-61 7-61
COMPLEX NUMBER TO A REAL POWER	ACM106 7-62 11-62
COMPLEX DIVISION	ECM116 8-62
COMPLEX ARITHMETIC	ACM186 7-63
COMPLEX POWER	ACM190 7-63
HYPERGEOMETRIC FN-COMPLEX PARAMETERS	ACM191 7-63 4-64
CONFLUENT HYPERGEOM FN-COMPLEX PARAMS	ACM192 7-63 4-64
FREQUENCY RESPONSE	AIS16
ARSENAL OF COMPLEX ALGOL PROCEDURES	BIT 2 4 P237
COMPLEX SQUARE ROOT	MCA X1 AP216
ZERO OF COMPLEX FUNCTION	MCA X1 AP217
MULTIPLY COMPLEX VALUES	MCA X1 AP218
COMPLEX PART OF POLYNOMIAL	MCA X1 AP219
LOG OF COMPLEX NUMBER	ACM243 11-64

	FD
GENERAL ORDER ARITHMETIC	ACM93 6-62 10-62
EXPONENTIATION OF SERIES -COEFFS	ACM134 11-62 7-63
EXPONENTIATION OF SERIES-COEFFS	ACM158 3-63 7-63 9-63
	FE
COMPLETE ELLIPTIC INTEGRAL 1ST KIND	ACM55 4-61 4-63
COMPLETE ELLIPTIC INTEGRAL 2ND KIND	ACM56 4-61
INCOMPLETE ELLIPTIC INTEGRALS	ACM73 12-61 10-62 2-63
COMPLETE ELLIPTIC INTEGRAL	ACM149 12-62 4-63
COMPLETE ELLIPTIC INTEGRALS	ACM165 4-63
PERIOD OF ELLIPTIC FUNCTION	NUM 4-5 P396
ARSENAL ELLIPTIC INTEGRALS AND FUNCTIONS	NUM 5-4 P296
	FP
CHEBYSHEV POLYNOMIAL BY RECURSION	ACM10 6-60 4-61
HERMITE POLYNOMIAL BY RECURSION	ACM11 6-60
LAGUERRE POLYNOMIAL BY RECURSION	ACM12 6-60
LEGENDRE POLYNOMIAL BY RECURSION	ACM13 6-60 4-61
POLYNML ORIGIN AND AXIS TRANSFORMER	ACM29 11-60
EVALUATE TABLE OF CHEBYSHEV POLYNMLS	ACM36 3-61
REDUCE DEGREE OF APPROXG POL-TELESCOPE 1	ACM37 3-61 8-62 8-63
REDUCE DEGREE OF APPROXG POL-TELESCOPE 2	ACM38 3-61 8-63
COEFFS OF QUOTIENT OF 2 POWER SERIES	ACM131 11-62
REVERSION OF SERIES	ACM193 7-63 12-63
CHEBYSHEV COEFFS	ACM 2-62 P93
CHEBYSHEV POLY COEFF	ACM227 5-64
POISSON-CHARLIER POLY	ACM234 7-64
	FS
COMPLEX EXPONENTIAL INTEGRAL	ACM13 7-60 2-61
GAMMA FUNCTION	ACM31 2-61 12-62 1-63
GAMMA FUNCTION	ACM34 2-61 7-62
ASSOC LEGENDRE FNS 1ST KIND-REAL OR IMAG	ACM47 4-61 8-63
SPHERICAL NEUMANN FUNCTION	ACM49 4-61
GAMMA FN-RANGE 1 TO 2	ACM54 4-61
SET ASSOC LEGENDRE POLYNMLS 2ND KIND	ACM62 7-61 12-61
RECIPROCAL GAMMA FN-REAL ARGUMENT	ACM80 3-62
JACOBI SYMBOL BY QUADRATIC RECIPROCITY	ACM99 6-62 11-62
REAL ERROR FUNCTION	ACM123 9-62 6-63 10-63 3-64
HANKEL FUNCTION 1ST KIND	ACM124 9-62
PSIF-LOG DERIV OF FACTORIAL FUNCTION	ACM147 12-62 4-63
MODIFIED HANKEL FUNCTION	ACM163 4-63 9-63
INCOMPLETE BETA RATIO	ACM179 6-63
ERROR FUNCTION-LARGE X	ACM180 6-63
COMPLEMENTARY ERROR FN-LARGE X	ACM181 6-63
HYPERGEOMETRIC FN-COMPLEX PARAMETERS	ACM191 7-63
CONFLUENT HYPERGEOM FN-COMPLEX PARAMS	ACM192 7-63
CALENDAR DATE TO AND FROM JULIAN DAY	ACM199 8-63
GAUSS FUNCTION	ACM209 10-63 3-64
FRESNEL INTEGRALS	ACM213 10-63
CALCULATION OF EASTER	ACM4-62 P209 11-62 P556
COMPUTING OF ROMAN FUNCTION	ALY 1 1 P25
CALCN OF NORMAL DISTRIBUTION FUNCTION	ALY 1 1 P33
COMPLEMENTARY FRESNEL INTEGRALS	BIT 2 3 P192
COMPUTATION OF THE FERMI FUNCTION	BIT 3 2 P141
CONTD FRCTN EXPSN FOR BINML QUADTC SURDS	NUM 5-2 P115
EXPONENTIAL INTEGRAL BY EPSILON ALGTHM	MCA MR60
GRAMMA FCN	ACM 221 3-64 10-64
INCOMPLETE BETA FCN	ACM 222 3-64 4-64
GAMMA FCN CONTROLLED ACCY	ACM 225 5-64 10-64
NORMAL DIST FCN	ACM226 5-64

	FT
ARCCOSSIN-ARCCOS AND ARCSIN	ACM206 9-63
TAN-LAMBERTS CONTINUED FRACTION	ACM 2-62 P89
ARCTAN ALD	MCA X1 AP116
ARCTAN ALS	MCA X1 AP117
SELECTIVE SUMMATION OF FOURIER SERIES	CJ 6-3 P248
ELEMENTARY FCNS CONT. FRAC.	ACM229 5-64
ARCTAN(Z)	ACM241 9-64
	FZ
PRODUCT OF TERMS OF A FN-CONTINUED PROD	MCA X1 AP202
ALGORITHMS FOR SPECIAL FUNCTIONS 1	NUM 4-5 P409
	GA
SECOND ORDER EQN-ADAMS METHOD	ACM 2-62 P88
INTEGRATION OF ORDINARY DIFFERENTL EQNS	MCA R743
	GL
RUNGE KUTTA	ACM9 5/60
ZERSOL-ZEROS OF SOLN OF 1ST EQUNS	ACM194 8-63
KUTTA MERSON	ACM218 12-63 10-64
FREQUENCY RESPONSE	AIS16
RUNGE KUTTA	NUM 2 2 P131
	GP
MULTIREGION TWO GROUP PROGRAMS	AIS14
USE OF METHOD OF KERNEL FUNCTION	NUM 3 3 P219
SOLVE 5 PT APPRXN DIRICHLET PROBLEM	B0001 P49
SOLVE 9 PT APPRXN DIRICHLET PROBLEM	B0001 P49
SOLVE 5 PT APPRXN DIRICHLET PROBLEM 2	B0001 P55
INTEGRATE HEAT EQUATION	B0001 P83
CANONICAL COEFFICIENTS FOR DIRICHLET PRO	NPL0001
	LA
REDUCTION OF ARRAY TO JACOBI FORM	ACM104 7-62
REAL SYMM MATRIX TO TRIDIAG FORM	ACM122 9-62 3-64
ORTHONORMALISING PROCEDURE	ACM127 10-62
SYMM BANDMATRIX TO TRIDIAG FORM	ACM183 6-63
ORTHONORMALISE COLUMNS	ACM 2-62 P91
REDUCTION OF A MATRIX TO CODIAGONAL FORM	CJ 4 P175
TRANSFORM MATRIX SYMMETRIC TO TRIDIAG	MCA X1 AP210
	LB
ANCESTOR	ACM96 6-62 3-63
SHORTEST PATH	ACM97 6-62
PATH MATRIX	ACM141 11-62

	LE
SIM EQNS-CROUT WITH PIVOTING	ACM16 9-60 10-60 3-61
TRIDIAGONAL SIM EQNS	ACM17 9-60
TRIDIAGONAL LINEAR EQUATIONS	ACM24 11-60
INVERT MATRIX	ACM42 4-61 11-61 1-63 8-63
SIM EQNS-CROUT WITH PIVOTING 2	ACM43 4-61 8-63
INVERSE OF FINITE SEG OF HILBERT MATRIX	ACM50 4-61 1-62 1-63
ADJUST INVERSE OF PERTURBED MATRIX	ACM51 4-61 7-62
MATRIX INVERSION-GAUSS-JORDAN	ACM58 5-61 6-62 8-62 12-62
INVERSION-SQ RT METHOD	ACM66 7-61 1-62 6-62
SIM EQUNS AND INVERSION - BY ITERATION	ACM92 5-62
SIM EQNS-GAUSS METHOD	ACM107 7-62 1-63 8-63
MATRIX INVERSION 2-GAUSS JORDAN	ACM120 8-62 1-63 8-63
SIM EQNS-GAUSS METHOD	ACM126 10-62
CROUT WITH EQUILIBRATION AND ITERATION	ACM135 11-62 7-64
MATRIX INVERSION	ACM140 11-62 8-63
INVERSE OF SYMMETRIC MATRIX 2	ACM150 2-63 7-63 3-64
1 ROW OF INVERSE MATRIX-MONTE CARLO	ACM166 4-63 9-63
BANDSOLVE-BAND EQUATIONS	ACM195 8-63
GAUSS SEIDEL	ACM220 12-63 6-64
INVERSION-GAUSS JORDAN	ACM 2-62 P94
BAND MATRICES	AIS11
GAUSSIAN SOLUTION OF LINEAR EQUATIONS	BIT 2 4 P256
SOLVE LINEAR EQUATIONS	MCA X1 AP205
INVERT	MCA X1 AP206
DETERMINANT AND INVERSE	MCA X1 AP208
SOLVE USING AP220 EVAL OF DETERMINANT	MCA X1 AP221
INVERT USING AP220	MCA X1 AP222
INVERT USING AP220 2	NUM5-2 P194
CONJUGATE GRADIENT METHOD	NUM MR63 AP225
SOLVE USING AP224	MCA MR63 AP225
INVERT USING AP224	MCA MR63 AP226
INVERT USING AP224 2	MCA MR63 AP227
SOLVE USING AP228	MCA MR63 AP229
GAUSSIAN ELIMINATION	B0001 P25
SOLVE BY OVERRELAXATION	B0001 P26
SOLVE BANDMATRIX	BIT 3-3 P207
SOLVE DETERMINANT INVERT SYMMETRIC	NPL 0004
SOLVE DETERMINANT SYMMETRIC BAND	NPL 0005
MATRIX PERMUTATION	ACM230 6-64
MATRIX INVERSION COMP. DIV.	ACM231 6-64
CONT. GRAD. METHOD	ACM238 8-64
	LF
FORM SET OF TEST MATRICES	ACM52 4-61 8-61 11-61 8-62
FORM SET OF TEST MATIECES	ACM52 1-63 8-63
	LG
SYMMETRIC MATRICES-MATRIX SCHEME	AIS21
	LH
MATRIX DIVISION-SQUARE ROOT METHOD	ACM197 8-63 3-64
TERM BY TERM ARITHUETRIC PROCEDURES	AIS20
SYMMETRIC MATRICES -MATRIX SCHEME	AIS21
	LL
MATRIX EIGEN VALUES AND VECTORS-JACOBI	ACM85 4-62 8-62 8-63
PREPARE FOR GIVENS METHOD AP212	MCA X1 AP211
GIVENS METHOD	MCA X1 AP212
COMPUTE EIGENVECTOR	MCA X1 AP213
TRANSFORM EIGENVECTOR	MCA X1 AP214
EIGENVALUES AND VECTORS OF SYM MATRIX	MCA X1 AP215
HOUSEHOLDERS METHOD FOR SYMMETRIC MATRIX	NUM 4-4 P357
EVALUES OF SYM TRIDIAG MAT BY BISECTION	NUM 4-4 P363
EVECS OF SYM TRIDIAG MAT BY INV ITERATN	NUM 4-4 P371
EVALUES OF SYM BAND MATRIX BY CHOLESKI	NUM 5-3 P277
EIGENVALUES AB MINUS K1-SYMMETRIC	NPL 0007
EIGENVALUES A MINUS KB-SYMMETRIC	NPL 0006

	LQ
DETERMINANT BY TRIANGULARISATION	ACM41 4-61 9-63 3-64
DETERMINANT-COMBINATORIAL METHOD	ACM159 3-63 12-63
REDUCTION OF MATRIX WITH POLYNML ELEMS	ACM170 4-63 8-63 7-64
DETERMINANT OF SYMMETRIC POSITIVE MATRIX	MCA MR 63 AP224
FORM DETERMINANT	MCA X1 AP 204
DETERMINANT AND SOLUTION	MCA X1 AP207
DETERMINANT AND SOLUTION	MCA X1 AP207
DETERMINANT AND INVERSE	MCA X1 AP208
DETERMINANT OF A BAND MATRIX	MCA X1 AP209
DETERMINANT OF SYMMETRIC POSITIVE MATRIX	MCA X1 AP220
DETERMINANT OF SYMMETRIC POSITIVE MATX 2	MCA MR63 AP228
EVALUATION OF DETERMINANT	ACM224 4-64 12-64
	LR
CRAM-READ SQUARE SYMM MATRIX AS VECTOR	ACM67 7-61 6-62
	LZ
SYMMETRIC MATRICES-MATRIX SCHEME	AIS21
	MC
LEAST SQUARE FIT BY ORTHOGONAL POLYNOMIALS	ACM28 11-60 12-61
CURVE FITTING WITH CONSTRAINTS	ACM74 1-62 6-63
CHEBYSHEV CURVE FIT	ACM91 5-62 4-63 5-64
SURFACE FIT -ORTHOGL POLYNOMIAL LST SQS	ACM164 4-63 8-63
LEAST SQUARES SURFACE FIT	ACM176 6-63
LEAST SQUARES SOLUTION WITH CONSTRAINTS	ACM177 6-63 7-63
ERLANG PROBABILITY FOR CURVE FITTING	ACM184 7-63
NORMAL PROBABILITY FOR CURVE FITTING	ACM185 7-63
LEAST SQUARES SOLUTION	ACM 2-62 P92
SPLINE CURVE	BIT 2 2 P81
	MD
INTERPOLATN-DIFFERENTIATN AND INTEGRATN	ACM77 2-62 6-62 8-63 11-63
MIN OF FN OF N VARABLES-STEEPEST DESCENT	ACM129 11-62 9-63
DIRECT SEARCH-MIN OF FN OF N VARIABLES	ACM178 6-63
DIFFERENCES AND DERIVATIVES	ACM187 7-63
STEEP 1-MIN BY STEEPEST DESCENT	ACM203 9-63 10-63
STEEP 2-MIN STEEPEST DESCENT	ACM204 9-63
ATIVE-USED BY ACM 203	ACM205 9-63
MAXIMUM OF A FUNCTION	MCA X1 AP201
	ME
PROCEDURES FOR RANGE ARITHMETIC	ACM61 7-61
CALCN OF NORMAL DISTRIBUTION FUNCTION	ALY 1 1 P33
ALGOL PROCEDURES FOR RANGE ARITHMETIC	SUC 15
	MG
POSITION OF POINT WRT POLYGON	ACM112 8-62 12-62
COMPUTE CODE STRING TO MOVE PLOTTING PEN	ACM162 4-63 8-63
PERSPECTIVE DRAWING PROGRAM-MARK 3	AIS4
	MH
SUMMATION OF FOURIER SERIES	ACM128 10-62 7-64
FOURIER SERIES APPROXIMATION	ACM157 3-63 9-63 10-63
EPSILON ALGORITHM	11-63 5-64
EPSILON ALGORITHM	NUM MATH V6

	MR
ROOTFINDER-FAST ITERATION METHOD	ACM2 2-60 6-60 8-60 3-61
ROOTS OF POLY-BAIRSTOW-HITCHCOCK METHOD	ACM3 2-60 6-60 2-61 3-61
ROOTS BY ITERATED BISECTION	ACM4 3-60 3-61
ROOTFINDER 2-ITERATION	ACM15 8-60 11-60 3-61
REAL ZEROS OF AN ARBITRARY FUNCTION	ACM25 11-60 3-61
ROOTFINDER 3	ACM26 11-60 3-61
ROOTS OF POLY-NEWTON BAIRSTOW ETC	ACM30 12-60 5-61 1-62
ZEROS REAL POLYNML BY RESULTNT PROCEDURE	ACM59 5-61
FACTORS OF POLYNOMIALS	ACM75 1-62 7-62 8-62
ROOTS OF POLYNOMIALS-INT-COEFFS	ACM78 2-62 3-62 8-62
ZEROS OF POLYNOMIALS-NEWTON MAEHLY	ACM105 7-62 7-63
BOUNCS ON A ZERO OF A POLYNOMIAL	ACM174 6-63
ROOTS OF ARBITRARY FN-MULLERS METHOD	ACM196 8-63
ROOTFINDERS-LINEAR AND QUADRATIC	AIS15
ZERO OF COMPLEX FUNCTION	MCA X1 AP217
GRAIFEE PROCESS	JCM 7 P93 83
ZERO OF A FUNCTION	MCA X1 AP200
SMALL INTEGERS AS PRIMITIVE ROOTS	NUM 4-4 P338
QUOTIENT DIFFERENCE METHOD	NUM 5-2 P96
ZERO OF AN ARBITRARY FUNCTION	BIT 3-2 P205

	MS
QUADRATURE INT-SEVERAL FNS-SAME LIMITS	ACM 2-60
EULER SUMMATION OF SERIES	ACM8 5-60 11-63
REAL EXPONENTIAL INTEGRAL	ACM20 10-60 2-61 4-61
MYLTIPLE INTEGRATION-GAUSS	ACM32 2-61 2-63
ROMBERG INTEGRATION	ACM60 6-61 3-62 5-62 7-64
INTERPOLATN-DIFFERENTIATN AND INTEGRATN	ACM77 2-62 6-62 8-63 11-63
SIMPSONS INTEGRATION	ACM84 4-62 7-62 8-62 11-62
FRESNEL SIN AND COS INTEGRALS-ASYMP EXPN	ACM88 5-62 10-63
FRESNEL SIN INTEGRAL	ACM89 5-62 10-63
FRESNEL COS INTEGRAL	ACM90 5-62 10-63
DEFINITE COMPLEX LINE INTEGRALS	ACM103 6-62
SIMPSONS RULE INTEGRATOR	ACM103 6-62
DEFINITE EXPONENTIAL INTEGRALS A	ACM108 7-62
DEFINITE EXPONENTIAL INTEGRALS B	ACM109 7-62
WEIGHT COEFFS FOR GAUSS QUADRATURE	ACM125 10-62
ADAPTIVE INTN BY SIMPSONS RULE	ACM145 12-62 4-63 5-64
MULTIPLE INTEGRATION	ACM146 12-62
NONRECURSIVE ADAPTIVE INTEGRATION	ACM182 6-63 4-64
ADAPTIVE AND MULTIPLE INTEGRATION	ACM195 8-63
GAUSS FUNCTION	ACM209 10-63
FRESNEL INTEGRALS	ACM213 10-63 11-64
COMPLEMENTARY FRESNEL INTEGRALS	BIT 2 3 P192
THE LIMIT OF A CONVERGING SEQUENCE	BIT 1 1 P64
SIMPSON INTEGRATION VARIABLE H	BIT 1 4 P290
INTEGRATION BY SIMPSONS RULE	MCA X1 AP203
SIMPSONS RULE IN ALGOL 58	NUM 1 1 P60
EULER-SUMMATION OF SERIES	NUM 2 2 P130
TRANSFORMATION OF SLOWLY CONVERGR SERIES	NUM 4-1 P10
MULTIPLE INT. SIMPS. RULE	ACM233 6-64
ROMBERG INTEGRATION	NUM MATH V6(15)
FRESNEL INTEGRAL	ACM244 11-64

	MT
INTERPOLATION BY CONTINUED FRACTIONS	ACM18 9-60 3-62
INTERPOLATION BY AITKEN ITERATION	ACM70 11-61 7-62
INTERPOLATN-DIFFERENTIATN AND INTEGRATION	ACM77 2-62 6-62 8-63 11-63
DIFFERENCE EXPRESSION COEFFICIENTS	ACM79 2-62 3-63
CONFLUENT DIVIDED DIFFERENCES	ACM167 4-63 9-63
BACKWARD DIVIDED DIFFS	ACM168 4-63 9-63
FORWARD DIVIDED DIFFS	ACM169 4-63 9-63
DIFFERENCES AND DERIVATIVES	ACM187 7-63
LAGRANGIAN INTERPOLATION	ACM210 10-63 10-63 P619
HERMITE INTERPOLATION	ACM211 10-63
NEVILLE INTERPOLATION	AIS17
INTERPOLATION WITH RATIONAL FUNCTIONS	NUM 3-3 P302
	MZ
ORTHONORMALISING PROCEDURE	ACM127 10-62
INOUT-ALGEBRA OF SETS-SPECIAL SUM	ACM156 3-63 8-63
CALENDAR DATE TO AND FROM JULIAN DAY	ACM199 8-63
SHANKS	ACM215 11-63
ORTHONORMALISE COLUMNS	ACM 2-62 P91
CALCULATION OF EASTER	ACM 4-62 P209 11-62 P556
SUM	MCA X1 AP119
INNERPRODUCT	MCA X1 AP120
FACTOR	MCA X1 AP125
REMAINDER	MCA X1 AP126
SOLVE LICHTENSTEIN-GERSHGORIN EQUATION	B0001 P242
CONFORMAL MAPPING OF NEARBY CIRCULAR REG	B0001 P256
CONVERGING FACTOR FOR CONTINUED FRACTION	NUM 5-4 P341
DSPLAY EPSILON ALGORITHM	BIT 3-3 P183
CALCULATE EPSILON ALGORITHM ARRAY	BIT 3-3 P184
DISPLAY SINGULAR EPSILON ALGORITHM	BIT 3-3 P191
CALC SINGULAR EPSILON ALGORITHM ARRAY	BIT 3-3 P192
	NC
BINOMIAL COEFFICIENTS BY RECURSION	ACM19 10-60 6-62 8-62
FACTORIAL	ACM33 2-61
PERMUTATIONS OF INTEGERS 0 TO N	ACM71 11-61 4-62 8-62
COMPOSITION GENERATOR	ACM72 11-61 8-62
PERMUTE FIRST N COMPONENTS OF VECTOR X	ACM86 4-62 8-62
PERMUTATION GENERATOR-LEXICOGRAHIC ORDER	ACM87 4-62 8-62 10-62
COMBINATION GENERATOR	ACM94 6-62 11-62 12-62
PERMUTATION IN LEXICOGRAPHICAL ORDER	ACM102 6-62
PERMUTE FIRST N COMPONENTS OF VECTOR X	ACM115 8-62 10-62 12-62
PERMUTATION GENERATOR	ACM130 11-62
LOCATE VECTOR IN LEXICOG ORDERED LIST	ACM151 2-63
NEXCOM-COMBS OF ELEMNTS IN COLUMN VECTOR	ACM152 2-63 7-63
COMBINATION IN LEXICOGRAPHICAL ORDER	ACM154 3-63 8-63
COMBINATION IN ANY ORDER	ACM155 3-63 8-63
CALC COMBS OF M THINGS N AT A TIME	ACM160 4-63 8-63 10-63
COMBS FROM 1 TO N AT A TIME OF M THINGS	ACM161 4-63 8-63 10-63
PERMUTATIONS IN LEXICOGRAPHICAL ORDER	ACM202 9-63
RANDOM PERMUTATION	ACM235 7-64
PERMS WITH REPETITION	ACM242 10-64
	NP
SIEVE-PRIME NUMBERS UP TO N	ACM35 3-61 4-62 8-62
PRIME TWINS	ACM223 4-64
GREATEST COMMON DIVISOR	ACM237 8-64 12-64
	NR
GENERATE PARTITIONS IN PART COUNT FORM	ACM95 6-62
GENERATE PARTITIONS WITH CONSTRAINTS	ACM114 8-62

	NZ
EUCLIDEAN ALGORITHM	ACM7 4-60
AUGMENTATION OF X BY Y	ACM68 8-61 11-61
CHAIN TRACING	ACM69 9-61
JACOBI SYMBOL BY QUADRATIC RECIPROCITY	ACM99 6-62 11-62
MAGIC SQUARE-EVEN ORDER	ACM117 8-62 1-63 3-63
MAGIC SQUARE-ODD ORDER	ACM118 8-62 12-62 1-63
DIOPHANTINE EQUATION	ACM139 11-62
TERM OF MAGIC SQUARE	ACM148 12-62 4-63
CALENDAR DATE TO AND FROM JULIAN DAY	ACM199 8-63
CALCULATION OF EASTER	ACM 4-62 P209 11-62 P556
	OL
ASSIGNMENT	ACM27 11-60 10-63 12-63
ECONOMISING A SEQUENCE 1	ACM81 3-62
ECONOMISING A SWQUENCE 2	ACM 82 3-62
OPTIMAL CLASSIFICATION OF OBJECTS	ACM83 3-62
SHORTEST PATH	ACM97 6-62
INTEGER SOLNS OF LINEAR PROG PROBLEMS	ACM153 2-63 8-63
DISCRETE CONVOLUTION	ACM208 10-63
	OP
CRITICAL PATH SCHEDULING	ACM40 3-61 9-61 10-62 6-64
EVALUATION OF A PERT NETWORK	ACM119 8-62
MINIMUM EXCESS COST CURVE	ACM217 12-63
TOPOLOGICAL ORDERING FOR PERT NETWORKS	ACM219 21-63
A PERT PROGRAM IN ALGOL 60	MCA MR56
	PH
CARBON DIOXIDE AND AIR FUNCTIONS	AIS12
THERMODYNAMIC PROPS OF STEAM AND WATER	AIS19
	PN
MULTIREGION TWO GROUP PROGRAMS	AIS14
	PQ
QUANTUM MECH INTS-SLATER TYPE ORBITALS	ACM110 7-62
MOLECULAR ORBITAL CALCULATION	ACM111 7-62
QUANTUM MECH INTS-SLATER TYPE INTEGRALS	ACM132 11-62
	SA
CHI SQUARED	AIS8
STATISTICAL ANALYSIS-CARD INPUT	AIS22
	SB
SMOOTHING BY GRAMS FORMULAS-1	ACM188 7-63
SMOOTHING BY GRAMS FORMULAS-2	ACM189 7-63
SMOOTH-FOURTH ORDER SMOOTHING	ACM216 11-63
	SC
CORRELATION COEFFS WITH MATRIX MULT	ACM39 3-61
	SM
FREQUENCY DISTRIBUTION	ACM212 10-63
MEAN STANDARD DEVIATION AND VARIANCE	AIS6
MEAN STANDARD DEV VAR AND DEG OF FREEDOM	AIS7
CALCN OF NORMAL DISTRIBUTION FUNCTION	ALY 1 1 933
	SR
TRIANGULAR REGRESSION	ACM142 12-62
MULTIPLE REGRESSION-MARK 1	AIS18

	SS
POS NORMAL DEVIATE FROM RANDOM NUMBER	ACM121 9-62
GENERATE RANDOM NUMBER	ACM133 11-62 12-62 3-63
ERLANG PROBABILITY FOR CURVE FITTING	ACM184 7-63
NORMAL PROBABILITY FOR CURVE FITTING	ACM185 7-63
NORMAL RANDOM	ACM200 8-63
DISCRETE CONVOLUTION	ACM208 10-63
QUASI-RANDOM POINT SEQUENCE	ACM247 12-64
	SV
VARIANCE HOMOGENEITY TEST-BARTLETT	AIS5
MEAN STANDARD DEVIATION AND VARIANCE	AIS6
MEAN STANDARD DEV VAR AND DEG OF FREEDOM	AIS7
TWO WAY ANALYSIS OF VARIANCE	AIS9
2 TO THE K FACTORIAL ANALYSIS-YATES	AIS10
	ZA
NESTING OF FOR STATEMENT 1	ACM137 11-62
NESTING OF FOR STATEMENT 2	ACM138 11-62
DIAGRAM	ACM 1-61 P55
FUSBUDGET	ACM 1-61 P64
COLLAPSE OWN STORAGE	ACM 1-61 P64
BOOLEAN OPERATOR NOT FLAG	ACM 1-61 P71
STORAGE ALLOCATION PROCEDURES	ACM 10-61 P444
REDUNDANCY CHECK	ACM 6-62 P340
ALGOL 60 SYNTAX CHECKER FOR THE IBM 7090	ORNL3399
	ZZ
ASSIGN	ACM173 6-63 10-63
CALENDAR DATE TO AND FROM JULIAN DAY	ACM199 8-63 11-64
CALCULATION OF EASTER	ACM 4-62 P209 11-62 P556
ALGOL INFORMATION SHEETS-THEIR PURPOSES	AIS1
ALGOL ON PEGASUS	AIS2
ALGOL ON DEUCE	AIS3
FREE FIELD READ	ACM239 8-64
COORDS FO ELLIPSOID	ACM240 9-64
GRAYCODE	ACM246 12-64

LOCATE ELEMENT IN LIST

```
procedure search(p)lowerbound:(ℓ)upperbound:(u)relation:(b);
        integer p,ℓ,u;  boolean b;
    comment  HUCC LIBRARY PROCEDURE DH02:
             AUTHOR:  J. BOOTHROYD
```

A Jensen procedure which searches an ordered array a between the inclusive limits $a[\ell]$ and $a[u]$ for an element of given value x, say. For arrays sorted in ascending order the relations $b = (x \leq a[p])$ and $b = (x < a[p])$ yield exit values of ℓ and u satisfying $a[\ell] < x \leq a[u]$ and $a[\ell] \leq x < a[u]$ respectively.

To locate an element of value y in the third column of a matrix A[1:10,1:5] for example, use the call

```
ℓ:=1;  u:=10;
search(p,ℓ,u,y≤A[p,3])
```

and to find an element of value y in the second row of the same matrix perform

```
ℓ:=1;  u:=5:
search(p,ℓ,u,y ≤ A[2,p])
```

Note from these examples that ℓ and u specify the extent of the search in the call as well as serving as result variables on exit from the procedure.

DH01

SORT REAL NUMBERS INTO ASCENDING ORDER

procedure shellsort (a, n); value n; real array a; integer n;

comment HUCC LIBRARY PROCEDURE DH01:

AUTHOR J. BOOTHROYD :

Elements a[1] through a[n] of real array a[1:n] are rearranged in ascending order. The method is that of D.A. SHELL (A High Speed Sorting Procedure. COMM. A.C.M. 2 (1959) 30-32) with subsequences chosen such that m_1, the first value of the partition parameter, is given by

$$m_1 = 2^k - 1 \text{ for } 2^k \leq n < 2^{k+1}$$

To implement Shell's original choice of $m_1 = [n/2]$ change the first statement of the procedure to m: = n. Note that shellsort specifies the array as type real. It may thus not be used to reorder elements of integer arrays unless the specification of a is changed to integer array a and in this case the local working variable w should preferably be declared as integer w;

SORT REAL NUMBERS IN ASCENDING ORDER

<u>procedure</u> exsort(a,n); value n; <u>integer</u> n; <u>array</u> a;

<u>comment</u> HUCC LIBRARY PROCEDURE DH03:

AUTHOR J. BOOTHROYD :

A procedure which sorts the elements a[1] through a[n] of <u>real</u> <u>array</u> a[1:n] into ascending order. The method used is that of exchanging element pairs. After the first pass, which ranges over all element positions a[1] to a[n], each pass is restricted to the range bounded by the first and last exchanges which occurred on the previous pass. This significantly improves the efficiency of the method.

The procedure is simple and may be easily modified to sort numbers into decending order by changing the Boolean expression a[i+1] < a[i] to a[i+1] $\geq$ a[i].

A version of exsort (under the identifier jensort) suitable for sorting rows or columns of an m × n matrix is included in the library as DH04;

SORT ROWS OR COLUMNS OF MATRIX

<u>procedure</u> jensort(ai,aiplus1,i,n); <u>value</u> n; <u>real</u> ai,aiplus1;
<u>integer</u> i,n;

<u>comment</u> HUCC LIBRARY PROCEDURE DH04:

AUTHOR J. BOOTHROYD :

A Jensen modification of exsort (DH03) which permits the sorting into ascending order of rows or columns of arrays of any number of dimensions. To sort the 3rd row of an array declared as A[1:n,1:m] use the call jensort(A[3,i],A[3,i+1],i,m). To sort each column of B[1:10,1:20] use the construction
<u>for</u> p:= 1 <u>step</u> 1 <u>until</u> 20 <u>do</u> jensort(B[i,p],B[i+1,p],i,10).
These examples, and a study of the differences between DH03 and DH04 should make clear the significance of the parameters ai and aiplus1;

DH01

```
procedure shellsort(a,n);  comment ACM 201;  value n;  real array a;  integer n;
     begin integer i,j,k,m;  real w;  switch s:=endj;
          for i:=1 step i until n do m:=2*i-1;
          for m:=m div 2 while m ≠ 0 do
               begin k:=n-m;
                    for j:=1 step 1 until k do
                         begin for i:=j step -m until 1 do
                                   begin if a[i+m] ≥ a[i] then goto endj;
                                        w:=a[i];  a[i]:=a[i+m];  a[i+m]:=w
                                   end i;
                    endj: end j
               end m
end shellsort;
```

DH02

```
          procedure search(p) lowerbound: (l)upperbound: (u)relation:(b);
          integer p,l,u;  boolean b;
          comment a Jensen procedure which searches an ordered array a between the inclusive limits a[l] and a[u] for
                    an element of given value, say x.  For arrays sorted into ascending order the relations
                    b=(x≤a[p]) and b=(x<a[p] produce exit values of l and u satisfying a[l] < x ≤ a[u] and
                    a[l] ≤ x < a[u] respectively;
               begin for p:=(u+l) div 2 while u ≥ l do if b then u:=p-1 else l:=p+1;
                    u:=u+1;  l:=l-1
               end search;
```

DH03

```
procedure exsort(a,n);  value n;  integer n;  array a;
begin integer i,k,r;  switch s:=L1,L2;  real temp;
  L1: k:=2;
  L2: r:=n-1;  n:=0;
      for i:=k-1 step 1 until r do
      if a[i+1]<a[i] then
            begin temp:=a[i];  a[i]:=a[i+1];  a[i+1]:=temp;
                  if n=0 then k:=i;  n:=i
            end;
      if n≠0 then goto if k≠1 then L2 else L1
end exsort;
```

DH04

```
procedure jensort(ai,aiplus1,i,n);  value n;  real ai,aiplus1;  integer n,i;
      begin integer k,r;  switch s:=L1,L2;  real temp;
       L1:  k:=2;
       L2:  r:=n-1;  n:=0;
            for i:=k-1 step 1 until r do
            if aiplus1 < ai then
                  begin temp:=ai;  ai:=aiplus1;  aiplus1:=temp;
                        if n=0 then k:=i;  n:=i
                  end;
            if n ≠ 0 then goto if k ≠ 1 then L2 else L1
      end jensort;
```

OUTPUT GRAPH OF VECTOR ELEMENTS

```
procedure Graphic(a,m,n);
value m,n;  array a;  integer m,n;
comment  HUCC LIBRARY PROCEDURE DP01:
         AUTHOR:  W. G. WARNE:
```

Plots the values of the elements of the nth order vector a on an (m+1)×n grid. Each element is plotted on a new line as a + character. Scale is chosen so that the maximum of a_i occurs m spaces from the left, and the minimum zero spaces from the left. Three newline characters are output before the first line marked by a row of m+1 dots and two newlines are output after the last line marked by a row of m+1 dots. A column of vertical bar characters marks the baseline (left margin).

Use is made of tabs to ensure optimum print speed. Tabs must be set at 12 spaces.

DP01

```
procedure Graphic(a,n,m);
value m,n;  array a;  integer m, n;
begin       integer j, k, h, r;  real max, min, x, y;
            print ££13??;
            for j := 0 step 1 until m do print £.?;
            k := 0;
            for j := 1 step 1 until n do
              k := if a[k] > a[j] then k else j;
            max := a[k];
            k := 0;
            for j := 1 step 1 until n do
              k := if a[k] < a[j] then k else j;
            min := a[k];
            x := m /(max-min);
            for h := 0 step 1 until n do
            begin       print ££l??;
                        k := x*(a[h]-min);
                        r := k div 12;
                        for j := 1 step 1 until r do print ££t??;
                        r := k - r*12;
                        for j := 1 step 1 until r do print ££s??;
                        print £+?;
            end;
            print ££l??;
            for j := 0 step 1 until m do print £.?;
            print ££12??
end Graphic;
```

OUTPUT PLOT OF ELEMENTS OF TWO VECTORS

procedure Bigraphic(a,b,n,m);

value n,m; **array** a,b; **integer** m,n;

comment HUCC LIBRARY PROCEDURE DP02

AUTHOR: W. G. WARNE:

Plots values of the elements of two nth order vectors a, and b on an $(m+1) \times n$ grid.

Each pair of elements is plotted on a new line, a_j being represented by _ (underline character) and b_j by a dot. Scale is chosen so that the maximum of a_j or b_j occurs m spaces from the left and the minimum of a_j, b_j at zero spaces, and both elements are plotted with the same scale and origin.

Otherwise format is as for Graphic(qv).

```
procedure Bigraphic(a,b,n,m);
value n,m; array a,b; integer m,n;
      begin integer j,ka,kb,h,sp,l;
            real min, max,x;
            procedure position;
                  begin integer i;
                        if l+sp<12 then
                              begin for i:=1 step 1 until sp do print ££s??;
                                    l:=l+sp;
                                    sp:=0
                              end
                                    else if l<10 then
                                          begin for i:=(l+sp)div 12 step -1 until 1 do print ££t??;
                                                l:=l+sp-(l+sp) div 12 * 12; sp:=0;
                                                for i:=1 step 1 until l do print ££s??
                                          end else
                                          begin for i:=12-l step -1 until 1 do print ££s??;
                                                sp := sp-12+l;  l:=0;
                                                position
                                          end
                  end   position;
            print ££l2??;
                        for j:=0 step 1 until m do print £.?;
                        max:= if a[0]>b[0] then a[0] else b[0];
                        for j:=1 step 1 until n do
                        max:= if a[j]>max or b[j]>max then (if a[j]>b[j] then
                         a[j] else b[j]) else max;
                        min:= if a[0]<b[0] then a[0]  else b[0];
                        for j:=1 step 1 until n do
                        min:= if a[j]<min or b[j]<min then (if a[j]<b[j] then
                        a[j] else b[j]) else min;
                        x:=m/(max-min);
                        for h:=0 step 1 until n do
            begin print ££l??;
                        ka:=x*(a[h]-min);
                        kb:=x*(b[h]-min);
                        sp:=0;
                        l:=0;
```

```
for j:=0 step 1 until m do
    begin if ka=j then
              begin position;
                    print £?
              end;
          if kb = j then
          begin  position;
                    l:=l+1 ;
                    print £.?
          end
                    else sp:=sp+1 ;
    end;
end;
       print ££l??;
        for j:=0 step 1 until m do print £.?;
        print ££l2??
end Bigraphic;
```

DZ01

CHARACTER PACK AND UNPACK

<u>procedure</u> pack(x) in position:(n)error exit:(label);

<u>comment</u> global <u>integer</u> MODE equal to the number of bits in x

must be declared and assigned before procedure is called;

<u>value</u> x,n;

<u>integer</u> x,n;

<u>label</u> label;

HUCC LIBRARY PROCEDURE DZ01:

AUTHOR: P. W. FORD:

DZ02

<u>procedure</u> unpack(x)from position:(n);

<u>comment</u> as for pack procedure;

<u>value</u> n;

<u>integer</u> x,n;

HUCC LIBRARY PROCEDURE DZ02:

AUTHOR: P. W. FORD:

This pair of procedures packs and unpacks sets of words containing MODE binary digits into the available ALGOL free store. In any one program, MODE must be constant and less than 39. The packing is dense, that is, each location in store is completely filled. If more than MODE binary digits are presented to be packed, the superfluous bits are ignored. n must not be less than 1. Storing starts at location 316 and works up the store until the program's data space is reached. If current data space is about to be overwritten, the procedure pack outputs on the typewriter "STORE FULL" followed by the machine location about to be overwritten and the number (n) of the word which would be placed in the location. The procedure then exits through the label without overwriting the store. Procedure unpack does not check the upper bound since, although rubbish may be extracted, the program itself cannot be destroyed by this procedure.

The machine is not aware of any packed information, so that subsequent data space allocation by program could overwrite some of the packed store. Thus the rule is, do not declare any variables, arrays, etc. after procedure pack is called, at least until the packed data has been processed. To pack and unpack one word takes of the order of two milliseconds.

ALGOL Tape 1 is destroyed by these procedures, but the owncode system is unaffected.

DZ01

```
procedure pack(x)in position:(n) error exit:(label);
                comment global MODE equal to the number of bits in x must be
                declared and assigned before procedure is called;
                value x,n;
                integer x,n;
                label label;
          begin integer a,b,c,divcount,shift,count;
                array A[1:1];
                switch s:=error,end,unset;
                a:=address(A)-1;
                b:=316;
                c:=39;
                elliott(0,2,0,0,2,7,n);
                elliott(3,0,MODE,0,5,2,n);
                elliott(5,7,0,0,0,0,0);
                elliott(1,6,count,0,5,6,c);
                elliott(0,4,b,0,2,0,divcount);
                elliott(0,7,a,0,4,1,error);
                elliott(3,0,divcount,0,0,5,b);
                elliott(5,2,c,0,5,7,0);
                elliott(0,7,count,0,0,7,c);
                elliott(2,0,shift,0,6,7,divcount);
                elliott(3,0,0,0,6,7,shift);
                elliott(5,0,0,0,1,0,x);
                elliott(6,7,MODE,0,5,0,0);
                elliott(3,0,x,0,6,7,shift);
                elliott(5,4,0,0,6,7,divcount);
                elliott(2,0,0,0,0,2,shift);
                elliott(0,7,MODE,0,4,1,unset);
                elliott(5,4,39,0,2,0,x);
                elliott(0,2,divcount,0,0,0,0);
                elliott(1,0,x,1,2,0,0);
unset:          elliott(4,3,end,0,4,0,end);
error:          print punch(3),£
STORE FULL?,sameline,divcount,n;
                goto label;
 end:           end pack;
```

DZ02

```
procedure unpack(x) from position:(n);
                value n;
                integer n,x;
          begin integer a,b,c,divcount,shift,count,diff,temp;
                switch ss:=rt,pu,collate,next;
                b:=316;
                c:=39;
                elliott(0,2,0,0,2,1,temp);
                elliott(3,0,c,0,0,5,MODE);
                elliott(1,0,temp,1,5,1,0);
                elliott(2,0,a,0,3,0,n);
                elliott(5,2,MODE,0,5,7,0);
                elliott(1,6,count,0,0,0,0);
                elliott(5,6,c,0,0,4,b);
                elliott(2,0,divcount,0,0,5,b);
                elliott(5,2,c,0,5,7,0);
                elliott(0,7,count,0,4,2,pu);
                elliott(2,0,diff,0,0,7,c);
                elliott(2,0,shift,0,6,7,divcount);
                elliott(3,0,0,0,2,0,temp);
                elliott(0,2,diff,0,0,7,MODE);
                elliott(4,1,rt,0,3,0,temp);
                elliott(5,0,39,0,6,7,divcount);
                elliott(2,7,8191,0,6,7,diff);
                elliott(5,4,0,0,4,0,collate);
 rt:            elliott(3,0,temp,0,6,7,shift);
                elliott(5,1,0,0,4,0,collate);
pu:             elliott(0,0,divcount,1,2,7,8191);
collate:        elliott(2,3,a,0,4,3,next);
next:           x:=a
         end unpack;
```

DZ03 to DZ11

Double Length Floating Point Arithmetic Package.

Author: W.G. Warne

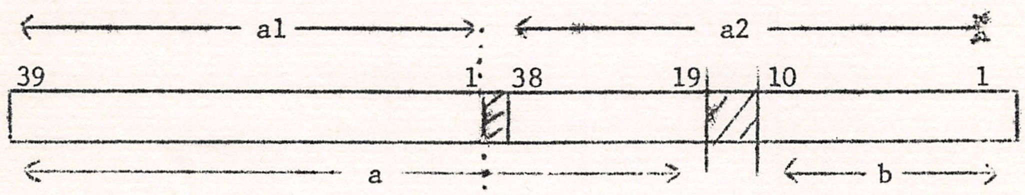
Procedures DZ03 to DZ11 provide facilities for performing real arithmetic and yield results which have twice the accuracy of the standard Algol and arithmetic operations.

Within this scheme a double length floating point number is represented by a number pair each of whose members is of type integer. For the number pair (a1, a2), for example, the floating point number $a*2^b$ is distributed within a1, a2 as follows :-

a is held as bits 39 to 1 of a1 and bits 38 to 19 of a2.

b is held as bits 10 to 1 of a2.

bits 39 and 18 to 11 of a2 are unused.



With this representation the double length floating number range is

$-2^{511} \leq x < -2^{-512}$, 0 , $2^{-512} \leq x < 2^{511}$ with a mantissa precision of 57 binary digits, roughly 17 decimal digits.

No provision is made for printing the decimal equivalent of a double length floating point number. The procedures are intended for special programming occasions when it is essential to minimise round-off errors. These procedures will be available only to those who convice either the officer-in-charge or the second officer-in-charge that their particular problem requires such precision. This restriction is imposed for the following reasons :-

(a) TIME An average time for all procedures is 1 to 1.5 milliseconds.

(b) STORAGE Total storage for the package is 659 locations.

ERROR INDICATIONS

Where the result of an operation exceeds the stated range, and also when the single length result of a package operation exceeds normal single length range, an error indication is given and the program is stopped. An error indication is also given if single length underflow is encountered ($0<|x|<2^{-256}$).

DZ03

procedure expand (s) single length real to double : (d1,d2);

value s; real s; integer d1,d2;

DZ04

procedure intexp (i) becomes : (d1,d2); value i; integer i, d1, d2;

DZ05

procedure contract (d1,d2) double to single : (s);

value d1,d2; integer d1,d2; real s;

comment error is "contract error";

DZ06

procedure mpyssd (s1) and :(s2) multiplied to give double :(d1,d2);

value s1,s2; integer d1,d2; real s1,s2;

DZ07

procedure divdss (d1,d2) divided by: (s) gives: (r);

value d1,d2,s; real s,r; integer d1,d2;

comment error is "divdss error";

DZ08

procedure negd (d1,d2) has complement :(n1,n2);

value d1,d2; integer d1,d2,n1,n2;

DZ09

procedure addddd (a1,a2) plus: (b1,b2) gives: (r1,r2);

value a1,a2,b1,b2; integer a1,a2,b1,b2,r1,r2;

DZ10

procedure mpyddd (a1,a2) multiplied by :(b1,b2) gives :(r1,r2);

value a1,a2,b1,b2; integer a1,a2,b1,b2,r1,r2;

comment error is "dfptoobig";

DZ11

procedure divddd(a1,a2) divided by: (b1,b2) gives :(t1,t2);

value a1,a2,b1,b2; integer a1,a2,b1,b2,t1,t2;

comment in the result (t1,t2) of this procedure the mantissa accuracy is 38 binary digits, (about 12 decimal digits).

BESSEL FUNCTION FOR SET OF INTEGER ORDERS

procedure BESSEL(x,n,eps,J); **value** x,n,eps; **real** x,eps;

integer n; **real** **array** J;

comment HUCC LIBRARY PROCEDURE FB01:

SOURCE : ACM 21:

Computes the values of the Bessel functions Jp(x) for real x for the set of all integer orders 0≤p≤n. These values are stored as J[p] in array J[0:n]. See also Comm ACM 3 (1960) p 600 and 8 (1964) p 219;

```
            procedure BESSEL(x,n,eps,J);  value x,n,eps;  real x,eps;  integer n;  real array J;
            comment Based on Algorithm 21, Communications of the A.C.M., 3,(1960),page 600, as
            corrected and certified Vol.8(1964),page 219;
                  begin real dist,rec0,rec1,rec2,sum,max,MAX,abx,err,z;
                        integer m,k,p,q;  boolean s;  real array Jbar[0:n];
                        switch SS:=exit,rec,norm;
                        if x=0.0 then begin J[0]:=1.0;
                                                for p:=1 step 1 until n do J[p]:=0.0;  goto exit
                                      end;
                        abx:=abs(x);
                        dist:= if abx ≥ 8.0 then 5.0 *abx↑(1.0/3.0)
                                              else 10.0;
                        k:=entier(( if abx ≥ n then abx else n)+dist)+1;
                        s:= false;  MAX:=0.25₁₀+74;
rec:                    rec0:=sum:=0.0;  rec1:=1.0;
                        z:=MAX*abs(x/k);
                        for p:=k step -1 until 1 do
                              begin if p>n+1 then q:=n else q:=p-1;
                                    J[q]:=rec2:=2.0*p/x*rec1-rec0;
                                    if abs(rec2)>z then
                                          begin rec1:=rec1/z;  rec2:=rec2/z;  sum:=sum/z;
                                                for m:=n step -1 until p-1 do J[m]:=J[m]/z
                                          end;
                                    if p=1 then sum:=sum+rec2
                                            else if (p div 2*2)≠p then sum:=sum+2.0*rec2;
                                                rec0:=rec1;  rec1:=rec2;
                              end recursion;
norm:                   for p:=0 step 1 until n do J[p]:=J[p]/sum;
                        if s then begin max:=0;
                                     for p:=0 step 1 until n do
                                           begin err:=abs(J[p]-Jbar[p]);
                                                 if err>max then max:=err
                                           end maximum error;
                                     if max ≤ eps then goto exit;
                                  end
                        else s:= true;
                        for p:=0 step 1 until n do Jbar[p]:=J[p];
                        k:=entier(k+dist);  goto rec;
exit:             end BESSEL;
```

P. WYNN'S ARSENAL OF COMPLEX ARITHMETIC PROCEDURES

Procedures FC01 to FC26 are taken from BIT Vol 2, No.4, p.237. Within these procedures any complex number written z in ordinary algebraic notation is represented by an ALGOL array of two elements, its real and imaginary parts.

It should be declared thus array z[0:1];

The elements z[0] and z[1] are respectively the real and imaginary parts. Outside the Wynn procedures the complex number is referred to as z[i]. The subscript i must be declared as an integer and the declaration must be valid in all blocks in which statements of procedures FC01 to FC26 occur. Inside the procedures i is assigned the value 0 before operations on real parts, 1 before operations on imaginary parts. Outside the procedures i should not be used.

Vectors, matrices and arrays with more dimensions having complex elements may be accommodated within the general scheme. For example an array with typical element Ap,q,r might be declared as

array A[ℓ1:u1,ℓ2:u2,ℓ3:u3,0:1] and

referred to by A[p,q,r,i]

or alternatively **array** A[0:1,ℓ1:u1,ℓ2:u2,ℓ3:u3]

appropriately referenced by A[i,p,q,r]

Call by name is used extensively in these procedures permitting the use of linear expressions of complex variables as actual parameters, e.g. $a_1z_1 + a_2z_2 + a_3z_3 + \ldots\ldots$ where a_1,a_2,a_3 are real numbers and $z_1,z_2,z_3 \ldots\ldots$ complex scalars or complex subscripted variables.

FC01

COMPLEX ASSIGNMENT

procedure eq(one,other); **real** one, other;

Examples: eq(z2[i],z1[i]) effect is "$z_2 := z_1$"

eq(z3[i],z1[i]+z2[i]) "$z_3 := z_1+z_2$"

eq(z3[i],z1[i]-z2[i]) "$z_3 := z_1-z_2$"

FC02

MULTIPLE COMPLEX ASSIGNMENT

<u>procedure</u> seqeq(third,second,first); <u>real</u> third,second,first;

Example: seqeq(z4[i],z3[i],z2[i]+z1[i]) "$z_4 := z_3 := z_2+z_1$"

FC03

COMPLEX MULTIPLICATION

<u>procedure</u> cm(res,one,other); <u>real</u> res,one,other;

Example: cm(z3[i],z2[i],z1[i]) "$z_3 := z_2 * z_1$"

FC04

COMPLEX DIVISION

<u>procedure</u> cd(res,one,other); <u>real</u> res,one,other;

Example: cd(z3[i],z2[i],z1[i]) "$z_3 := z_2 / z_1$"

FC05

ASSIMILATE REAL IN COMPLEX OPERATION

<u>real</u> <u>procedure</u> real(variable); <u>real</u> variable;

Example: eq(z2[i],z1[i]+real(a)) "$z_2 := z_1+a$"

FC06

ASSIMILATE IMAGINARY IN COMPLEX OPERATION

<u>real</u> <u>procedure</u> imaginary(variable); <u>real</u> variable;

Example: eq(z2[i],z1[i]+imaginary(a)) "$z_2 := z_1+j*a$"

(where,here,$j^2 = -1$)

FC07

COMPLEX CONJUGATE

<u>real</u> <u>procedure</u> cxconj(it); <u>real</u> it;

Example: eq(z2[i],cxconj(z1[i]))

if z1 = a + j * b then z_2 = a - j * b   ($j^2 = -1$)

FC08

MODULUS OF COMPLEX NUMBER

<u>real</u> <u>procedure</u> mod(it); <u>real</u> it;

Example: r:= mod(z1[i])

evaluates $r = |z|$ where $|z| = (a^2+b^2)^{\frac{1}{2}}$ for z = a+j*b

FC09

ARGUMENT OF COMPLEX NUMBER

<u>real</u> <u>procedure</u> arg(it); <u>real</u> it;

Example: theta:= arg(z1[i]) assigns to theta the argument of z

for $z = re^{j\theta}$ the argument lies in the range $-\pi<\theta\leq\pi$

FC10

POLAR FORM

<u>procedure</u> polar form(res,r,theta);

Example: polar form(z1[i],R,phi) assigns to the elements of z1 the

values $r\cos\phi, r\sin\phi$.

FC11

MULTIPLICATION BY IMAGINARY OPERATOR

<u>procedure</u> imult(res,it); <u>real</u> res, it;

Example: imult(z1[i],z2[i])   "$z_1 := j*z_2$"

Procedures FC12 to FC23 (except FC17) are all of the form

procedure function(res,it) and effect the operation

res:= function(it).

FC12

COMPLEX SQUARE

procedure compsq(res,it); real res,it;

FC13

COMPLEX RECIPROCAL

procedure comprecip(res,it); real res,it;

FC14

COMPLEX ROOT

procedure cxsqrt(res,it); real res,it;

N.B. USES FC08, FC09 and FC10.

FC15

COMPLEX LOGARITHM

procedure compln(res,it); real res,it;

FC16

COMPLEX EXPONENTIAL

procedure compexp(res,it); real res,it;

FC17

HYPERBOLIC FUNCTIONS

procedure hyp(sinh,cosh,y); value y; real sinh,cosh,y;

evaluates $\cosh y=(e^y+e^{-y})/2$ and $\sinh y=(e^y-e^{-y})/2$
Special care with precision is taken in evaluating sinhy for small values of y.

FC18

COMPLEX SINE

<u>procedure</u> compsin(res,it); <u>real</u> res,it;

N.B. USES FC17.

FC19

COMPLEX COSINE

<u>procedure</u> compcos(res,it); <u>real</u> res,it;

N.B. USES FC17.

FC20

COMPLEX TANGENT

<u>procedure</u> comptan(res,it); <u>real</u> res,it;

N.B. USES FC04 and FC17.

FC21

COMPLEX INVERSE SINE

<u>procedure</u> cxarcsin(res,it); <u>real</u> res,it;

FC22

COMPLEX INVERSE COSINE

<u>procedure</u> cxarccos(res,it); <u>real</u> res,it;

N.B. USES FC01, FC05, FC21.

FC23

COMPLEX INVERSE TANGENT

<u>procedure</u> cxarctan(res,it); <u>real</u> res,it;

N.B. FC01, FC04, FC05, FC12 and FC14.

FC24

COMPLEX POWER OF COMPLEX VARIABLE

<u>procedure</u> onehochother(res,one,other); <u>real</u> res,one,other;

Example: onehochother(z3[i],z2[i],z1[i]) has the effect of

$z_3 := z_2 \uparrow z_1$

<u>TEST FOR EVEN INTEGER</u>

<u>boolean</u> <u>procedure</u> even(integer); <u>integer</u> integer;

This procedure takes the value <u>true</u> if 2 is a factor of integer.

LOGARITHM OF COMPLEX NUMBER

<u>procedure</u> LOGC(a,b,c,d,FAIL); <u>value</u> a,b; <u>real</u> a,b,c,d; <u>label</u> FAIL;

<u>comment</u> HUCC LIBRARY PROCEDURE FC26:

AUTHOR ACM 243 :

The procedure computes the number c + di which is equal to the principal value of the natural logarithm of a + bi, i.e. such that $-\pi < d < +\pi$. A label parameter FAIL permits exit from the procedure in the event that the real part of the result approaches minus infinity. Where required in the procedure the numerical values of π, $\pi/2$ and $\ln(\sqrt{8})$ are provided;

```
procedure eq(one,other);  real one,other;
for i:=0,1 do one:=other;
```

FC01

```
procedure seqeq(third,second,first);  real third,second,first;
for i:=0,1 do third:=second:=first;
```

FC02

```
procedure cm(res,one,other);  real res,one,other;
begin real  Reone,Imone,Reother,Imother;
      i:=0;  Reone:=one;  Reother:=other;
      i:=1;  Imone:=one;  Imother:=other;
      res:=Reone*Imother+Imone*Reother;  i:=0;
      res:=Reone*Reother-Imone*Imother
end;
```

FC03

```
procedure cd(res,one,other);  real res,one,other;
begin real Reone,Imone,Reother,Imother,denom;
      i:=0;  Reone:=one;  Reother:=other;
      i:=1;  Imone:=one;  Imother:=other;
      denom:=Reother*Reother+Imother*Imother;
      res:=(Imone*Reother-Reone*Imother)/denom;
      i:=0;  res:=(Reone*Reother+Imone*Imother)/denom
end;
```

FC04

```
real procedure real(variable);  real variable;
real :=(if i=0 then variable else 0.0);
```

FC05

should not be underlined.

FC 06

```
real procedure imaginary(variable);  real variable;
        imaginary:=(if i=0 then 0.0 else variable);
```

FC 07

```
real procedure cxconj(it);  real it;
        cxconj:=(if i=0 then it else -it);
```

FC 08

```
real procedure mod(it);  real it;
        begin real Reit,Imit;
              i:=0;  Reit:=it;  i:=1;  Imit:=it;
              mod:=sqrt(Reit*Reit+Imit*Imit)
        end;
```

FC 09

```
real procedure arg(it);  real it;
        begin real Reit,Imit;
              i:=0;  Reit:=it;  i:=1;  Imit:=it;
              arg:=(if Reit>0.0 then arctan (Imit/Reit) else
              if abs(Imit)>0.0 then
              sign(Imit)*1.57079633 -
              arctan(Reit/Imit))
              else 3.14159265 )
        end;
```

N.b. error
bracket in wrong place

```
procedure polar form(res,r,theta);  real res,r,theta;
begin real r1,theta1;
      r1:=r;  theta1:=theta;  i:=0;  res:=r1*cos(theta1);
      i:=1;  res:=r1*sin(theta1)
end;
```

FC10

```
procedure imult(res,it);  real res,it;
begin real aux;
      i:=0;  aux:=it;  i:=1;  res:=aux;  aux:=it;
      i:=0;  res:=-aux
end;
```

FC11

```
procedure compsq(res,it);  real res,it;
begin real Reit,Imit;
      i:=0;  Reit:=it;  i:=1;  Imit:=it;
      res:=2.0*Reit*Imit;
      i:=0;  res:=Reit*Reit-Imit*Imit
end;
```

FC12

```
procedure comprecip(res,it);  real res,it;
begin real Reit,Imit,denom;
      i:=0;  Reit:=it;  i:=1;  Imit:=it;
      denom:=Reit*Reit+Imit*Imit;
      res:=-Imit/denom;  i:=0;  res:=Reit/denom
end;
```

FC13

FC 14

```
procedure cxsqrt(res,it);  real res,it;
        polar form(res,sqrt(mod(it)),0.5*arg(it));
```

FC 15

```
procedure compln(res,it);  real res,it;
        begin real aux;
              aux:=ln(mod(it));  i:=0;  res:=aux;
              aux:=arg(it);  i:=1;  res:=aux
        end;
```

FC 16

```
procedure compexp(res,it);  real res,it;
        begin real aux1,aux2;
              i:=0;  aux1:=exp(it);  i:=1;  aux2:=it;
              res:=aux1*sin(aux2);
              i:=0;  res:=aux1*cos(aux2)
        end;
```

FC17

```
procedure hyp(sinh,cosh,y);  value y;  real sinh,cosh,y;
begin real y1;
      y1:=exp(y);  cosh:=0.5*(y1+1.0/y1);
      if abs(y)≥1.0  then sinh:=0.5*(y1-1.0/y1)
      else begin integer r;  real br,brplus1,brplus2;
            array CWC[0:5];
            CWC[0]:=1.13031 821;
            CWC[1]:=4.43368 498;
            CWC[2]:=5.42926 312;
            CWC[3]:=3.19843 646₁₀-6;
            CWC[4]:=1.10367 7₁₀-8;
            CWC[5]:=2.498₁₀-11;
            brplus1:=brplus2:=0.0;
            y1:=2.0*(2.0*y*y-1.0);
            for r:=5 step -1 until 0 do
                  begin br:=y1*brplus1-brplus2+CWC[r];
                        if r≠0 then begin brplus2:=brplus1;
                                          brplus1:=br
                                    end
                  end;
            sinh:=y*(br-brplus1)
            end
end;
```

```
procedure compsin(res,it);  real res,it;
begin real Reit,Imit,sinhImit,coshImit;
      i:=0;  Reit :=it;  i:=1;  Imit:=it;
      hyp(sinhImit,coshImit,Imit);
      res:=cos(Reit)*sinhImit;  i:=0;
      res:=sin(Reit)*coshImit
end;
```

FC18

```
procedure compcos(res,it);  real res,it;
begin real Reit,Imit,sinhImit,coshImit;
      i:=0;  Reit:=it;  i:=1;  Imit:=it;
      hyp(sinhImit,coshImit,Imit);
      res:=-sin(Reit)*sinhImit;  i:=0;
      res:=cos(Reit)*coshImit
end;
```

FC19

FC20

```
procedure comptan(res,it);  real res,it;
begin real Reit,Imit,sinhImit,coshImit,sinReit,cosReit;
      array aux1;  aux2[0:1];
      i:=0;  Reit:=it;  i:=1;  Imit:=it;
      hyp(sinImit,coshImit,Imit);
      sinReit:=sin(Reit);  cosReit:=cos(Reit);
      aux1[1]:=cosReit*sinhImit;
      aux2[1]:=-sinReit*sinhImit;
      aux1[0]:=sinReit*coshImit;  aux2[0]
      aux2[0]:=cosReit*coshImit;
      cd(res,aux1[i],aux2[i])
end;
```

```
procedure cxarcsin(res,it);  real res,it;
begin real x,y,d1,d2,d3,d4;
      i:=0;  x:=it;  i:=1;  y:=-it;
      d3:=x*x;  d4:=y*y;  d1:=d3+d4;  d2:=d3-d4;
      d3:=d1*d1-2.0*d2;  d4:=sqrt(d3+1.0);
      res:=sign(y)*0.5*
      ln(d1+d4+sqrt(d3+d1*(d1+2.0*d4)));
      i:=0;  res:=arctan(x*sqrt(2.0/(1.0-d2+d4)))
end;
```

FC21

```
procedure cxarccos(res,it);  real res,it;
begin array aux[0:1];
      cxarcsin(aux[i],it);
      eq(res,real(1.57079 633)-aux[i])
end;
```

FC22

```
procedure cxarctan(res,it);  real res,it;
begin array aux1,aux2[0:1];
      eq(aux1[i],it);  compsq(aux2[i],aux1[i]);
      cxsqrt(aux2[i],real(1.0)+aux2[i]);
      cd(aux2[i],aux1[i],aux2[i]);
      cxarcsin(res,aux2[i])
end;
```

FC23

```
procedure onehochother(res,one,other);  real res,one,other;
begin array aux1[0:1];
      compln(aux1[i],one);  cm(aux1[i],other,aux1[i]);
      compexp(res,aux1[i])
end;
```

FC24

```
boolean procedure even(integer);  integer integer;
even:=(integer=(integer div 2)*2);
```

FC25

FC26

```
procedure LOGC(a,b,c,d,FAIL);  value a,b;  real a,b,c,d;  label FAIL;
        if a=0 and b=0 then goto FAIL else
            begin real e,f;
                e:=0.5*a;  f:=0.5*b;
                if abs(e) < 0.5 and abs(f) < 0.5 then
                    begin c:=abs(2*a)+abs(2*b);
                        d:=8*a/c*a+8*b/c*b;
                        c:=0.5*(ln(c)+ln(d))-1.03972077
                    end
                else  begin c:=abs(0.5*e)+abs(0.5*f);
                        d:=0.5*e/c*e+0.5*f/c*f;
                        c:=0.5*(ln(c)+ln(d))+1.03972077
                    end;
                d:=if a ≠ 0 and abs(e) ≥ abs(f) then arctan(b/a)+(if sign(a) ≠ -1 then 0 else
                        if sign(b) ≠ -1 then 3.14159265 else -3.14159265) else -arctan(a/b)+1.57079633 * sign(b)
            end LOGC;
```

FP01

SUM SERIES OF CHEBYSHEV POLYNOMIALS

real procedure Chebsum(x,P,m,s);

value x,m,s; **real** x; **integer** m,s; **array** P;

comment HUCC LIBRARY PROCEDURE FP01

AUTHOR: W. G. WARNE:

Evaluates the sums:-

$$\sum_{i=0}^{m} P[i]T_i(x) \quad \text{if } s = 1$$

$$\sum_{i=0}^{m} P[i]T_{2i}(x) \quad \text{if } s = 2$$

$$\sum_{i=0}^{m} P[i]T_{2i+1}(x) \quad \text{if } s = 3$$

where $T_i(x)$ is the ith Chebyshev polynomial defined by $T_i(x) = \cos(i\arccos(x))$.
The evaluation uses Chenshaw's method involving the recurrence relation for Chebyshev polynomials.

see Chenshaw, C.W.: A note on the summation of Chebyshev Series M.T.A.C. 9 188-120 (1955).

FP01

```
real procedure Chebsum (x,P,m,s);
value x,m,s; real x;integer m,s; array P;
comment This procedure evaluates the sum
 m,
 > P T (x) where T (x)= cos(i*arcos(x)) and the prime denotes
i=0 i i             i
halving of the first term,when s=1.
                         m,          m
For s=2 and s=3,the sums > P T  (x) and > P T   (x) are evaluated
                        i=0 i 2i       i=0 i 2i+1
respectively.;
begin       real C0,C1,C2,y,mult;
            integer i;
            switch S1:= L1,L2,L6;
            switch S2:= L3,L7,L4,L5;
            y:=x;
            goto S1[s];
L2:L6:      y:= 2*x*x - 1.0;
L1:         C1:=C0:=C2:=0.0;
            mult := 2.0*y;
            for i:=m step -1 until 0 do
            begin       C2:= C1; C1:= C0;
                        C0:= mult*C1 -C2 + P[i]
            end;
            goto S2[s];
L3:L7:      Chebsum := 0.5 * (C0 - C2); goto L5;
L4:         Chebsum := x* (C0 - C1);
L5: end of Chebsum;
```

FS01

LOGARITHM OF FACTORIAL(n)

real procedure ℓogfac(a); **integer** a;

comment HUCC LIBRARY PROCEDURE FS01:

AUTHOR COMPUTER BULLETIN :

Evaluates ℓn(factorial(a)). For values of $a \leq 7$ the procedure computes factorial(a) and uses the standard function ℓn to evaluate ℓn(factorial(a)). For values of $a > 7$ the procedure uses the approximation

$$\ell n(\text{factorial}(a)) = \ell n(\text{gamma}(a+1))$$
$$\simeq \ell n(\sqrt{2\pi}) + (a+0.5)*\ell n(a) - a + B1/(2a) - B2/(12a)$$

where B1(=1/6) and B2(=1/30) are the first two Bernoulli numbers.

```
real procedure logfac(a);  integer a;
comment evaluates ln(factorial(a));
      begin real aa,k,b;
            aa:=a;
            if aa>7.5 then begin k:=1.0/aa;
            logfac:=(aa+0.5)*ln(aa)+k*(0.0833333333-k*k/360.0)-aa+0.9189385333
                           end
            else begin aa:=aa+0.5;  b:=1.0;
            for k:=2.0 step 1.0 until aa do b:=b*k;
            logfac:=ln(b)
                 end
      end logfac;
```

FOURTH ORDER RUNGE-KUTTA

```
procedure RKFOUR(x,y,f,h,n);  value h,n;  integer n;
real x,h;  array y,f;
comment  HUCC LIBRARY PROCEDURE GL01:
         AUTHOR:  J. BOOTHROYD      :
```

A procedure for solving n first order linear differential equations of the form

$$\frac{dy_1}{dx} = f_1(x,y_1,y_2\ldots\ldots y_n)$$

$$\frac{dy_2}{dx} = f_2(x,y_1,y_2\ldots\ldots y_n)$$

$$\frac{dy_n}{dx} = f_n(x,y_1,y_2\ldots\ldots y_n)$$

for known starting values $(y_1)_0, (y_2)_0, \ldots\ldots (y_n)_0$.

The correct use of RKFOUR requires the following preliminary operations.

1. Array y must be initialised so that, for y[1:n],

   $y[i] = (y_i)_0 \quad i = 1,2,\ldots\ldots\ldots n$

2. The right-hand sides of the equations must be specified by the declaration of a procedure function such that the operation of function places in array f[1:n] the appropriate f[i], i = 1,2,....n.

   Examples: A

$$\frac{dy_1}{dx} = y_2$$

$$\frac{dy_2}{dx} = -k^2 y_1$$

```
procedure function(x,y,f);  real x;  array y,f;
  begin x:=x;  f[1]:=y[2];  f[2]:= -k*k*y[1] end;
```

B

$$\frac{dy_1}{dx} = y_2 + x$$

$$\frac{dy_2}{dx} = y_3$$

$$\frac{dy_3}{dx} = -y_2\, y_3 - y_1\, x^2$$

```
procedure function(x,y,f);  real x;  array y,f;
   begin f[1]:= y[2]+x;  f[2]:= y[3];
         f[3]:= -y[2]*y[3]-y[1]*x*x end;
```

A single call of RKFOUR replaces the values y[i],i=1,2,...n appropriate to some x by the corresponding values at x+h. The user must, therefore, arrange to call RKFOUR successively the appropriate number of times consistent with h, the interval of interest and the values of the argument at which values of the solution are required.

Suppose values of y[1] are required at intervals of 0.2 from x=0 to x=5 inclusive and the computation step interval h is 0.05.

```
    x:= 0;
for i:= 1 step 1 until 26 do
   begin print x,y[1];
      for j:= 1 step 1 until 4 do RKFOUR(        )
   end;
```

GL01

```
procedure RKFOUR(x,y,f,h,n);  value h,n;  integer n;  real x,h;  array y,f;
    begin real hby2,hby6;  integer i;  array ybar,p[1:n];
        procedure step(a,b,k1,k2);  value k1,k2;  real k1,k2;  array a,b;
        for i:=1 step 1 until n do
            begin p[i]:=p[i]+k1*f[i];  a[i]:=y[i]+k2*b[i]  end step;
        hby2:=h/2.0;  hby6:=h/6.0;
        for i:=1 step 1 until n do p[i]:=0.0;
        function(x,y,f);  step(ybar,f,1.0,hby2);  x:=x+hby2;
        function(x,ybar,f);  step(ybar,f,2.0,hby2);
        function(x,ybar,f);  step(ybar,f,2.0,h);  x:=x+hby2;
        function(x,ybar,f);  step(y,p,1.0,hby6)
    end RKFOUR;
```

MATRIX INVERSION

```
procedure mxinvert(a,n,eps,singular);  value n,eps;
     array a;  integer n;  real eps;  label singular;
     comment  HUCC LIBRARY PROCEDURE LE01:
              AUTHOR:  J. BOOTHROYD
```

Inverts a matrix in its own space using the Gauss-Jordan method with complete matrix pivoting. I.e., at each stage the pivot has the largest absolute value of any element in the remaining matrix. The coordinates of the successive matrix pivots used at each stage of the reduction are recorded in the successive element positions of the row and column index vectors r and c. These are later called upon by the procedure mxperm which rearranges the rows and columns of the matrix. If the matrix is singular the procedure exits to an appropriate label in the main program.

This procedure uses procedure mxperm, and for correct operation either NC02 or NC04 must previously have been declared.

```
procedure mxinvert (a,n,eps,singular);  value n,eps;  array a;  integer n;  real eps;  label singular;
begin integer i,j,k,pivi,pivj,p,ri,ci,rk,cj,iless1;  real pivot;  integer array r,c[1:n];
            comment set row and column index vectors;
            for i:=1 step 1 until n do r[i]:=c[i]:=i;
            comment find initial pivot;  pivi:=pivj:=1;  pivot:=a[1,1];
            for i:=1 step 1 until n do for j:=1 step 1 until n do
            if abs(a[i,j])>abs(pivot) then begin pivi:=i; pivj:=j;  pivot:=a[i,j] end;
            comment start reduction;
            for i:=1 step 1 until n do
                  begin ri:=r[pivi]; r[pivi]:=r[i]; r[i]:=ri; ci:=c[pivj]; c[pivj]:=c[i]; c[i]:=ci; iless1:=i-1;
                        if eps > abs(a[ri,ci]) then
                              begin print punch(3),sameline,digits(3),££l?MATRIX SINGULAR?,££l?i=?,i,££l?PIVOTS FOLLOW?;
                                    for i:=1 step 1 until n do
                                                print punch(3),sameline,digits(3),££l??,r[i],££s4??,c[i];
                                    goto singular
                              end;
                        for j:=1 step 1 until iless1,i+1 step 1 until n do
                                    begin cj:=c[j];  a[ri,cj]:=a[ri,cj]/pivot end;
                                          a[ri,ci]:=1.0/pivot;  pivot :=0;
                        for k:=1 step 1 until iless1,i+1 step 1 until n do
                              begin rk:=r[k];
                                    for j:=1 step 1 until iless1,i+1 step 1 until n do
                                          begin cj:=c[j]; a[rk,cj]:=a[rk,cj]-a[ri,cj]*a[rk,ci];
                                                if k>i and j>i and abs(a[rk,cj]) >abs(pivot) then
                                                      begin pivi:=k;  pivj:=j; pivot:=a[rk,cj] end conditional
                                          end jloop;
                                    a[rk,ci]:=-a[ri,ci]*a[rk,ci]
                              end kloop
                  end iloop and reduction;                 what is p.
            comment rearrange rows;     mxperm(a[j,p],a[k,p],j,k,r,c,n,p);
            comment rearrange columns;  mxperm(a[p,j],a[p,k],j,k,c,r,n,p)
end mxinvert;
```

SOLVE LINEAR EQUATIONS - ONE R.H. SIDE

<u>procedure</u> SOLVEQ(a,n); <u>value</u> n; <u>integer</u> n; <u>real</u> <u>array</u> a;

<u>comment</u> HUCC LIBRARY PROCEDURE LE02:

AUTHOR: J. BOOTHROYD

A procedure which solves Ax = b where A is [1:n,1:n] and b is [1:n]. The elements of A and b must occupy the elements of a[1:n,1:n+1] as follows:-

a[i,j] = A[i,j] i = 1,2,...,n, j = 1,2,....n.

a[i,n+1] = b[i] i = 1,2,...,n.

i.e. the R.H.S. vector b occupies column n+1 of the augmented matrix a. On exit from the procedure the solutions x[i] occupy a[i,n+1] i = 1,2,...n.

The method is Gauss-Jordan reduction to diagonal form with partial pivoting. No row exchanges are performed (the matrix is reduced in-situ), but a final permutation of a[i,n+1] is performed to reorder the solution vector.

The procedure <u>does not</u> include a test for singularity or ill-condition, and should not be used if the equations may be suspected of either attribute.

LE02

```
procedure SOLVEQ(a,n);  value n;  integer n;  real array a;
begin integer i,j,k,piv,ri,rk,n1;  real pivot,arki;  integer array r[1:n];  switch S:=L;
        for i:=1 step 1 until n do r[i]:=i;  n1:=n+1;
        for i:=1 step 1 until n do
            begin piv:=i;  ri:=r[i];  pivot:=a[ri,i];
                for k:=i+1 step 1 until n do
                if abs(a[r[k],i])>abs(pivot) then
                    begin piv:=k;  pivot:=a[r[k],i] end;   k const.
                ri:=r[piv];  r[piv]:=r[i];  r[i]:=ri;
                for j:=i+1 step 1 until n1 do a[ri,j]:=a[ri,j]/pivot;
                for k:=1 step 1 until i-1,i+1 step 1 until n do
                    begin rk:=r[k];  arki:=a[rk,i];
                        for j:=i+1 step 1 until n1 do a[rk,j]:=a[rk,j]-arki*a[ri,j]
                    end for k;
            end reduction;
        for i:=n step -1 until 2 do
            begin k:=r[i];
L:              if k≠i then
                    begin if k>i then begin k:=r[k];  goto L end;
                        pivot:=a[i,n1];  a[i,n1]:=a[k,n1];  a[k,n1]:=pivot
                    end
            end
end SOLVEQ;
```

LE03

MATRIX DECOMPOSITION Ax = b

procedure ATOLU1(a,r,n,eps,singular); value n,eps;

integer n; real array a; integer array r; label singular;

comment HUCC LIBRARY PROCEDURE LE03

AUTHOR: J. BOOTHROYD

Performs an in situ decomposition of a matrix A into quasi triangular matrices L, a lower triangle, and U a strictly upper triangle, by Gaussian elemination with partial pivoting. The successive pivots of the reduction constitute the quasi diagonal elements of L. These are chosen to be the elements of maximum modulus in the leading column of successive reduced matrices. The pivotal row subscripts are recorded in the non-local vector r[1:n], the ith pivotal row subscript occupying r[i]. If the matrix is singular (eps is the appropriate tolerance) the procedure records the ith stage of the reduction at which singularity is detected by changing the sign of r[i] before it exists to the main program via the label parameter singular.

```
procedure ATOLU1(a,r,n,eps,singular);  value n,eps;  integer n; real eps;
real array a;  integer array r;  label singular;
      begin integer i,j,k,piv,ri,rk;  real pivot,arki;
            for i:=1 step 1 until n do r[i]:=i;
            for i:=1 step 1 until n do
                  begin piv:=i;  ri:=r[i];  pivot:=a[ri,i];
                        for k:=i+1 step 1 until n do
                        if abs(a[r[k],i])>abs(pivot) then begin piv:=k; pivot:=a[r[k],i] end;
                        if abs(pivot)<eps then
                              begin print punch(3),££l?MATRIX IS SINGULAR?; r[piv]:=-r[piv];
                                    goto singular
                              end;
                        ri:=r[piv]; r[piv]:=r[i]; r[i]:=ri;
                        for j:=i+1 step 1 until n do a[ri,j]:=a[ri,j]/pivot;
                        for k:=i+1 step 1 until n do
                              begin rk:=r[k];  arki:=a[rk,i];
                                    for j:=i+1 step 1 until n do a[rk,j]:=a[rk,j]-arki*a[ri,j]
                              end for k
                  end for i
      end ATOLU1;
```

FORWARD AND BACK SOLUTION Ax = b

```
procedure LU1SOL(a,b,r,n);  value n;  real array a,b;
          integer array r;  integer n;
comment   HUCC LIBRARY PROCEDURE LE04
          AUTHOR:  J. BOOTHROYD
```

Solves the equation Ax=b by the successive operations Lf=b and Ux=f where L and U are respectively a lower and strictly upper triangle such that L(U+I)=A. The elements of L and U share the array a in accordance with the pivoting strategy of procedure AT01U1.

i.e. L[i,j] occupy a[r[i],j] $j \leq i$ i= 1,2,.....,n

and U[i,j] occupy a[r[i],j] $j > i$ i= 1,2,.....,n.

r[1:n] is a non-local permutation vector in which is stored the a array row subscripts of successive rows of L. The procedure uses procedure sigma (MS01).

Notes: For a matrix A[1:n,1:n], vector b[1:n], the procedure

calls	ATOLU1(A,r,n,eps,bad)	LE03
	LU1SOL(A,b,r,n)	LE04
	PERMB(b,r,n)	NC01

Solve Ax=b for one right hand side.

For large sets of equations with many right hand sides use:-

ATOLU1(A,r,n,eps,bad)	LE03
→ Read b	
LU1SOL(A,b,r,n)	LE04
PERMB(b,r,n)	NC01
— Print b	

Alternatively the output procedure ZP01 may be used in place of PERMB, Print b.

LE04

```
procedure LU1SOL(a,b,r,n); value n; real array a,b; integer array r;  integer n;
comment solves the equation Ax=b by the successive operations Lf=b and Ux=f
where L and U are respectively a lower and strictly upper triangle such that
L(U+I)=A. The elements of L and U share the array a in accordance with the
pivoting strategy of procedure ATOLU1.
            i.e. L[i,j] occupy a[r[i],j]  j≤i i= 1,2,.....,n
            and  U[i,j] occupy a[r[i],j]  j>i i= 1,2,.....,n.
r[1:n] is a non-local permutation vector in which is stored the a array row
subscripts of successive rows of L.  The procedure uses procedure sigma;
begin integer i,j,ri,iless1;
   for i:=1 step 1 until n do
      begin ri:=r[i];  iless1:=i-1;
         b[ri]:=(b[ri]-sigma(a[ri,j]*b[r[j]],j,1,iless1))/a[ri,i]
      end;
   for i:=n-1 step -1 until 1 do
      begin ri:=r[i];
         b[ri]:=b[ri]-sigma(a[ri,j]*b[r[j]],j,i+1,n)
      end
end LU1SOL;
```

MATRIC DECOMPOSITION BY CROUT'S METHOD

<u>procedure</u> crout(a,r,n,eps,singular); <u>value</u> n,eps; <u>integer</u> n;
<u>real</u> eps; <u>real</u> <u>array</u> a; <u>integer</u> <u>array</u> r; <u>label</u> singular;
<u>comment</u> HUCC LIBRARY PROCEDURE LE05:
AUTHOR: J. BOOTHROYD :

Performs an in situ decomposition of a matrix A into quasi triangular matrices L, a lower triangle and U, a strictly upper triangle, by Crout's method with partial pivoting.
The successive pivots of the reduction constitute the quasi diagonal elements of L. These are chosen to be the elements of maximum modulus in the leading column of successive reduced matrices. The pivotal row subscripts are recorded in the non-local vector r[1:n], the ith pivotal row subscript occupying r[i]. If the matrix is singular (eps is the appropriate tolerance) the procedure records the ith stage of the reduction at which singularity is detected by changing the sign of r[i] before it exits to the main program via the label parameter singular. This procedure uses the procedure sigma.
The results of this procedure are identical with those of LE03. LE05 is not a direct replacement for LE03 since its identifier is different and LE05 uses MS01.

```
procedure crout(a,r,n,eps,singular);  value n,eps;  integer n;  real eps;
real array a;  integer array r;  label singular;
      begin integer i,j,k,piv,ri,rk,iless1;  real pivot,arki;
            for i:=1 step 1 until n do r[i]:=i;
            for i:=1 step 1 until n do
                  begin  pivot:=0.0;  piv:=i;  iless1:= i-1;
                        for k:=i step 1 until n do
                              begin rk:=r[k];
                                    arki:=a[rk,i]:=a[rk,i]-sigma(a[rk,j]*a[r[j],i],j,1,iless1);
                                    if abs(arki)>abs(pivot) then begin piv:=k; pivot:=arki end;
                              end;
                        if abs(pivot)<eps then begin print punch(3),££1?MATRIX IS SINGULAR?;
                                                        r[piv]:=-r[piv];  goto singular
                                                  end;
                        ri:=r[piv]; r[piv]:=r[i]; r[i]:=ri;
                        for j:=i+1 step 1 until n do a[ri,j]:=(a[ri,j]-sigma(a[ri,k]*a[r[k],j],k,1,iless1))/pivot
                  end for i
      end crout;
```

TRIANGULAR MATRIX PRODUCT LU=A

<u>procedure</u> LU1TOA(a,r,n); <u>value</u> n; <u>integer</u> n; <u>real</u> <u>array</u> a;

<u>integer</u> <u>array</u> r;

<u>comment</u> HUCC LIBRARY PROCEDURE LE06:

AUTHOR: J. BOOTHROYD :

Performs an in situ matrix multiplication of quasi-triangular matrices L, a lower triangle, and U, a strictly upper triangle to form a matrix A given, in effect, by A=L(U+1). The elements of L and U are distributed within the array a in accordance with the pivoting strategy of procedure ATOLU1,

i.e. elements L[i,j] occupy positions a[r[i],j] j≤i, i=1,2,..,n

elements U[i,j] occupy positions a[r[i],j] j>i, i=1,2,..,n.

r[1:n] is a non-local permutation vector in which is recorded the a array row subscripts of successive rows of L.

Note: LE06 is the inverse of LE03. For a matrix A the operations

ATOLU1(A,r,n,eps,singular)

LU1TOA(A,r,n)

will leave A unchanged except for corruptions to the elements of A caused by computational errors.

```
procedure LU1TOA(a,r,n);  value n;  integer n;  real array a;  integer array r;
comment performs an in situ matrix multiplication of quasi-triangular matrices L,
a lower triangle, and U, a strictly upper triangle to form a matrix A given, in
effect, by A=L(U+I).  The elements of L and U are distributed within the array
a in accordance with the pivoting strategy of procedure ATOLU1, i.e.
   elements L[i,j] occupy positions a[r[i],j] j≤i, i=1,2,........, n
   elements U[i,j] occupy positions a[r[i],j] j>i, i=1,2,........, n.
r[1:n] is a non-local permutation vector in which is recorded the a array row
subscripts of successive rows of L;
begin integer i,j,k,ii,ri;  real sum;
   for i:=n step -1 until 1 do
      begin ii:=i+1;  ri:=r[i];
         for j:=n step -1 until ii do
            begin sum:=a[ri,j]*a[ri,i];
               for k:= i-1 step -1 until 1 do sum:=sum+a[ri,k]*a[r[k],j];
               a[ri,j]:=sum
            end;
         for j:= i step -1 until 1 do
            begin sum:=a[ri,j];
               for k:= j-1 step -1 until 1 do sum:=sum+a[ri,k]*a[r[k],j];
               a[ri,j]:=sum
            end
      end
end LU1TOA;
```

INVERT QUASI LOWER TRIANGULAR MATRIX

procedure INVL(a,r,n); **value** n; **integer** n; **real** **array** a;

integer **array** r;

comment HUCC LIBRARY PROCEDURE LE07:

AUTHOR: J. BOOTHROYD :

Inverts a quasi triangular matrix L in its own space. The elements of the lower triangle L are distributed within the array a in accordance with the pivoting strategy of procedure ATOLU1, i.e.

elements L[i,j] occupy positions a[r[i],j] $j \leq i, i=1,2,..,n$

r[1:n] is a non-local permutation vector in which is recorded the a array row subscripts of successive rows of L. The procedure uses procedure sigma (MS01).

INVERT QUASI UPPER TRIANGULAR MATRIX

procedure INVU1(a,r,n); **value** n; **integer** n; **real array** a;

integer array r;

comment HUCC LIBRARY PROCEDURE LE08:

AUTHOR: J. BOOTHROYD :

Inverts a quasi-triangular matrix in its own space. The elements of U, a strictly upper triangle, are distributed within the array a in accordance with the pivoting strategy of procedure ATOLU1, i.e.

elements U[i,j] occupy positions a[r[i],j] j>i i=1,2,..,n.

r[1;n] is a non-local permutation vector in which is recorded the a array row subscripts of successive rows of U. The procedure uses procedure sigma (MS01).

LE07

```
procedure INVL(a,r,n);  value n;  integer n;  real array a;  integer array r;
begin        integer i,j,k,ri,iless1;  real arii;
             for i:=1 step 1 until n do
                   begin ri:=r[i];  iless1:=i-1;
                         arii:=a[ri,i];
                         for j:=1 step 1 until iless1 do a[ri,j]:=-sigma(a[ri,k]*a[r[k],j],k,j,iless1)/arii;
                         a[ri,i]:=1.0/arii
                   end
end INVL;
```

LE08

```
procedure INVU1(a,r,n);  value n;  integer n;  real array a;  integer array r;
begin        integer i,j,k,iplus1,jless1,ri;
             for i:=n-1 step -1 until 1 do
                   begin ri:=r[i];  iplus1:=i+1;
                         for j:=n step -1 until iplus1 do
                               begin jless1:=j-1;
                                     a[ri,j]:=-(a[ri,j]+sigma(a[ri,k]*a[r[k],j],k,iplus1,jless1))
                               end
                   end
end INVU1;
```

TRIANGULAR MATRIX PRODUCT UL=A

procedure U1LTOA(a,r,n); **value** n; **integer** n; **integer array** r;

array a;

comment HUCC LIBRARY PROCEDURE LE09:

AUTHOR: J. BOOTHROYD :

Performs an in-situ matrix product (U+I)L=A where U and L are respectively a strictly upper triangle and a lower triangle occupying the array a in accordance with the pivoting strategy of procedure ATOLU1

i.e. L[i,j] occupy a[r[i],j] $j \leq i$ i=1,2,....,n

and U[i,j] occupy a[r[i],j] $j > i$ i=1,2,....,n.

r[1:n] is a non-local permutation vector in which is recorded the a array row subscripts of successive rows of L.

The procedure uses procedure sigma (MS01).

```
procedure U1LTOA(a,r,n); value n; integer n; integer array r; array a;
comment performs an in- situ matrix product (U+I)L=A where U and L are respectively
a strictly upper triangle and a lower triangle occupying the array a in accordance
with the pivoting strategy of procedure ATOLU1
            i.e.  L[i,j] occupy a[r[i],j]  j≤i  i=1,2,....,n
            and   U[i,j] occupy a[r[i],j]  j>i  i=1,2,....,n.
r[1:n] is a non-local permutation vector in which is recorded the a array row
subscripts of successive rows of L. The procedure uses procedure sigma;
begin integer i,j,k,nless1,ri;
      nless1:=n-1;
      for i:= 1 step 1 until nless1 do
            begin ri:= r[i];
                  for j:= 1 step 1 until i do a[ri,j]:=a[ri,j]+sigma(a[ri,k]*a[r[k],j],k,i+1,n);
                  for j:=i+1 step 1 until n do a[ri,j]:= sigma(a[ri,k]*a[r[k],j],k,j,n)
            end
end U1LTOA;
```

LIBRARY PROCEDURES LE03 to LE09 produce and perform operations on quasi triangular matrices L and U1 where L is a lower triangle and U a strictly upper triangle. LIBRARY PROCEDURES LE10 to LE14 correspond to some, but not all of LE03 to LE09 for matrices L1 and U respectively a strictly lower and an upper triangle.

LE10

```
procedure ATOL1U(a,r,n,eps,singular);  value n,eps;  integer n;
real eps;  integer array r;  label singular;
comment    HUCC LIBRARY PROCEDURE LE10:
           AUTHOR:  J. BOOTHROYD      :
           see note above and LE03.
```

LE11

```
procedure L1USOL(a,b,r,n);  value n;  real array a,b;
integer array r;  integer n;
comment    HUCC LIBRARY PROCEDURE LE11:
           AUTHOR:  J. BOOTHROYD      :
           see note above and LE04.
```

LE12

```
procedure INVL1(a,r,n);  value n;  real array a;  integer array r;
comment    HUCC LIBRARY PROCEDURE LE12:
           AUTHOR:  J. BOOTHROYD      :
           see note above and LE07.
```

```
procedure INVU(a,r,n);  value n;  integer n;  real array a;
integer array r;
comment   HUCC LIBRARY PROCEDURE LE13:
          AUTHOR:  J. BOOTHROYD       :

          see note overleaf and LE08.
```

LE14

```
procedure UL1TOA(a,r,n);  value n;  integer n;  array a;
comment   HUCC LIBRARY PROCEDURE LE14:
          AUTHOR:  J. BOOTHROYD       :

          see note overleaf and LE09.
```

```
procedure ATOL1U(a,r,n,eps,singular);  value n,eps;  integer n;  real eps;
real array a;  integer array r;  label singular;
begin       integer i,j,k,ri,rk,piv;  real pivot,arki;
            for i:=1 step 1 until n do r[i]:=i;
            for i:=1 step 1 until n do
                 begin piv:=i;  pivot:=a[r[i],i];
                      for k:=i+1 step 1 until n do
                      if abs (a[r[k],i])>abs(pivot)then begin piv:=k;  pivot:=a[r[k],i] end;
                      if abs(pivot)<eps then
                           begin print punch(3),££1?MATRIX IS SINGULAR?;  r[piv]:=-r[piv];
                                goto singular
                           end;
                      ri:=r[piv];  r[piv]:=r[i];  r[i]:=ri;
                      for k:=i+1 step 1 until n do
                           begin rk:=r[k];  arki:=a[rk,i]:=a[rk,i]/pivot;
                                for j:=i+1 step 1 until n do a[rk,j]:=a[rk,j]-arki*a[ri,j]
                           end
                 end
end ATOL1U;
```

LE11

```
procedure L1USOL(a,b,r,n);  value n;  real array a,b;  integer array r;  integer n;
begin integer i,j,ri,iless1;
   for i:=2 step 1 until n do
      begin ri:=r[i];  iless1:=i-1;
         b[ri]:=b[ri]-sigma(a[ri,j]*b[r[j]],j,1,iless1);
      end;
   for i:=n step -1 until 1 do
      begin ri:=r[i];
         b[ri]:=(b[ri]-sigma(a[ri,j]*b[r[j]],j,i+1,n))/a[ri,i]
      end
end L1USOL;
```

LE12

```
procedure INVL1(a,r,n);  value n;  integer n;  real array a;  integer array r;
begin       integer i,j,k,ri,iless1,jplus1;
            for i:=2 step 1 until n do
                  begin ri:=r[i];  iless1:=i-1;
                        for j:=1 step 1 until iless1 do
                              begin jplus1:=j+1;
                                    a[ri,j]:=-(a[ri,j]+sigma(a[ri,k]*a[r[k],j],k,jplus1,iless1))
                              end
                  end
end INVL1;
```

LE13

```
procedure INVU(a,r,n);  value n;  integer n;  real array a;  integer array r;
begin      integer i,j,k,ri,iplus1;  real arri;
           for i:=n step -1 until 1 do
               begin ri:=r[i];  iplus1:=i+1;  arri:=a[ri,i];
                   for j:=n step -1 until iplus1 do a[ri,j]:=-sigma(a[ri,k]*a[r[k],j],k,iplus1,j)/arri;
                   a[ri,i]:=1.0/arri
               end
endINVU;
```

LE14

```
procedure UL1TOA(a,r,n);  value n; integer n;  integer array r;  array a;
    begin integer i,j,k,nless1,iless1,ri;
        nless1:=n-1;
        for i:=1 step 1 until n do
            begin ri:=r[i];  iless1:=i-1;
                for j:=1 step 1 until iless1 do a[ri,j]:=sigma(a[ri,k]*a[r[k],j],k,i,n);
                for j:=i step 1 until nless1 do a[ri,j]:=a[ri,j]+sigma(a[ri,k]*a[r[k],j),k,j+1,n)
            end
    end UL1TOA;
```

SOLVE TRIDIAGONAL LINEAR EQUATIONS

procedure tridiag(a,b,c,d,ℓo,hi,eps,fail); **value** ℓo,hi,eps;

integer ℓo,hi; **real** eps; **label** fail; **real array** a,b,c,d;

comment HUCC LIBRARY PROCEDURE LE15:

AUTHOR J. BOOTHROYD :

Solves the equations

$$\begin{bmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & c_3 & & \\ & & & & & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \\ d_{n-1} \\ d_n \end{bmatrix}$$

by the recurrence formulae

$$x_n = d_n' \qquad x_r = d_r' - c_r' x_{r+1}$$

$$d_1' = d_1/b_1 \qquad c_1' = c_1/b_1$$

$$d_r' = \frac{d_r - a_r d_{r-1}'}{b_r - a_r c_{r-1}'} \qquad c_r' = \frac{c_r}{b_r - a_r c_{r-1}'} \qquad r=2,3,\ldots\ldots,n.$$

The coefficients occupy the elements of arrays a,b,c,d as follows:

a_2 to a_n	occupy	a[ℓo] to a[hi-1]	of	a[ℓo:hi-1]
b_1 to b_n	occupy	b[ℓo] to b[hi]	of	b[ℓo:hi]
c_1 to c_{n-1}	occupy	c[ℓo] to c[hi-1]	of	c[ℓo:hi-1]
d_1 to d_n	occupy	d[ℓo] to d[hi]	of	d[ℓo:hi]

eps is a singularity tolerance. If $b_r - a_r c_{r-1}' < \text{eps}$ for any r the set is presumed singular and exit occurs through the label parameter fail. The values of $b_r - a_r c_{r-1}'$ and r are recorded on the control typewriter. The solution is delivered in array d and array c is overwritten in the process;

```
procedure tridiag(a,b,c,d,lo,hi,eps,fail);  value lo,hi,eps;  integer lo,hi;
real eps;  label fail;  real array a,b,c,d;
      begin            integer i,iless1;  real w,ailess1;
            d[lo]:=d[lo]/b[lo];  w:=b[lo];
            for i:=lo+1 step 1 until hi do
                  begin iless1:=i-1;  ailess1:=a[iless1];
                        c[iless1]:=c[iless1]/w;
                        w:=b[i]-ailess1*c[iless1];
                        if w<eps then begin print punch(3),scaled(3),digits(3),
                                                      ££l?TRIDIAG MATRIX SINGULAR w=?,w,££t?i=?,i;
                                                   goto fail
                                          end;
                        d[i]:=(d[i]-ailess1*d[iless1])/w
                  end;
            for i:=hi-1 step -1 until lo do d[i]:=d[i]-c[i]*d[i+1]
      end tridiag;
```

LE 16

DECOMPOSITE SYMMETRIC POSITIVE DEFINITE MATRIX

procedure choleski (a,n,fail); value n; integer n; array a; label fail;

comment HUCC LIBRARY PROCEDURE LE 16 (revised)

Author : J. Boothroyd.

Performs the decomposition of a symmetric matrix A into lower and upper triangles L,L^T such that $LL^T=A$. The elements of the lower triangle of A are replaced by the elements of L. The strictly upper triangle of A is not used. If the decomposition fails, indicating that A is non positive definite the procedure exits through the label fail. The procedure uses procedure sigma (MS 01);

LE 17

INVERT LOWER TRIANGULAR MATRIX

procedure linv (a,n); value n; integer n; array a;

comment HUCC LIBRARY PROCEDURE LE 17 (revised)

Author : J. Boothroyd

Replaces the lower triangular portion L of a[1:n,1:n] by the elements of L^{-1}. The procedure uses MS 01;

LE 18

MATRIX MULTIPLICATION

procedure mxmult (a,b,c,m,n,p); value m,n,p;
integer m,n,p; array a,b,c;

comment HUCC LIBRARY PROCEDURE LE 18

Author : J. Boothroyd.

c[1:m,1:p] := a[1:m,1:n] * b[1:n,1:p];

This procedure uses MS 01.

LE16

```
procedure choleski(a,n,fail); value n; integer n; array a; label fail;
begin integer i,j,k,iless1; real aii;
      for i:= 1 step 1 until n do
            begin iless1:= i-1;
                  aii:=a[i,i]-sigma(a[i,k]*a[i,k],k,1,iless1);
                  if aii<0 then goto fail;
                  aii:=a[i,i]:=sqrt(aii);
                  for j:= i+1 step 1 until n do
                        a[j,i]:= (a[j,i]-sigma(a[i,k]*a[j,k],k,1,iless1))/aii
            end
endcholeski;
```

LE17

```
procedure linv(a,n); value n; integer n; array a;
comment inverts the lower triangle of a in situ;
begin integer i,j,k,iless1; real aii;
      for i:=1 step 1 until n do
            begin iless1:= i-1; aii:=a[i,i];
                  for j:= 1 step 1 until iless1 do
                        a[i,j]:= -sigma(a[i,k]*a[k,j],k,j,iless1)/aii;
                  a[i,i]:= 1.0/aii
            end
end linv;
```

LE18

```
procedure mxmult(a,b,c,m,n,p); value m,n,p; integer m,n,p; array a,b,c;
comment c[m p]:= a[m n]  b[n p];
begin integer i,j,k;
      for i:= 1 step 1 until m do
            for j:= 1 step 1 until p do c[i,j]:=sigma(a[i,k]*b[k,j],k,1,n)
endmxmult;
```

DECOMPOSE BANDMATRIX INTO LOWER AND UPPER'TRIANGLES'

procedure bandlr (a,n,m,eps,escape); **value** n,m;

integer n,m; **array** a; **real** eps; **label** escape;

comment HUCC LIBRARY PROCEDURE LE 19

Author : J. Boothroyd.

Decomposes the n^{th} order band matrix of width m stored in a[1:n,1:m] into a lower 'triangle' L and a strictly upper triangle U. At entry to the procedure the elements of band matrix A[1:n,1:n] must occupy the matrix a[1:n,1:m] in accordance with the following rules :-

A[i,j]	occupies	a[i,j]	$1 \le i < (m+1)$ **div** 2
A[i,j]	occupies	a[i,j-i+(m+1) **div** 2]	$(m+1)$ **div** $2 < i \le n$

Diagrammatically

A11	A12	A13	0	0	0		A11	A12	A13	0	0
A21	A22	A23	A24	0	0		A21	A22	A23	A24	0
A31	A32	A33	A34	A35	0	→	A31	A32	A33	A34	A35
0	A42	A43	A44	A45	A46		A42	A43	A44	A45	A46
0	0	A53	A54	A55	A56		A53	A54	A55	A56	0
0	0	0	A64	A65	A66		A64	A65	A66	0	0
	Array A[1:n,1:n]							array a[1:n,1:m]			

UNUSED ELEMENTS of matrix a must be set to zero.

This procedure does not incorporate a pivoting strategy. eps is a tolerance which provides a criterion for deciding whether any leading submatrix is singular in which event the procedure exits to the label escape;

FORWARD SOLUTION AND BACK SUBSTITUTION FOR BAND EQUATION Ax=b

```
procedure bandsol(a,n,mb); value n,m;
integer n,m; array a,b;
```

comment HUCC LIBRARY PROCEDURE LE 20

Author : J. Boothroyd

Solves the equation Ax=b where A is a bandmatrix by the successive operations Lf=b and Ux=f where L and U are respectively a lower triangle and a unit upper triangle occupying the array a[1:n;1:m] following the use of procedure LE 19. The solution vector replaces the vector b[1:n].

Procedures LE 19 and LE 20 may together be used to solve Ax=b for unlimited right-hand sides by some scheme similar to:-

```
Read Matrix A
bandlr (A,n,m,eps,escape)        LE 19
Read vector b
bandsol (A,n,m,b)                LE 20
Print solution vector b
```

LE 19

```
procedure bandlr(a,n,m,eps,escape);  value n,m;  integer n,m;  real array a;  label escape;  real eps;
begin      integer i,j,k,jlim,klim,nless1;  real pivot;
           jlim:=klim:=(m+1) div 2;  nless1:=n-1;
           for i:=1 step 1 until nless1 do
                 begin pivot:=a[i,1];  if abs(pivot)<eps then goto escape;
                       for j:=2 step 1 until jlim do a[i,j]:=a[i,j]/pivot;
                       for k:=i+1 step 1 until klim do
                             begin pivot:=a[k,1];
                                   for j:=2 step 1 until jlim do a[k,j-1]:=a[k,j]-pivot*a[i,j];
                                   for j:=jlim +1 step 1 until m do a[k,j-1]:=a[k,j];
                                   a[k,m]:=pivot
                             end;
                       if klim≠n then klim:=klim+1
                 end
end;
```

LE 20

```
procedure bandsol(a,n,m,b);  value n,m;  integer n,m;  real array a,b;
begin      integer i,j,k,lim,jlim;  real sum;
           lim:=(m+1) div 2;  jlim:=1;
           for i:=1 step 1 until n do
                 begin sum:=0.0;  k:=0;
                       for j:=i-1 step -1 until jlim do
                             begin sum:=sum+b[j]*a[i,m-k];  k:=k+1  end;
                                   b[i]:=(b[i]-sum)/a[i,1];
                                   if i-jlim+2>lim then jlim:=jlim+1
                 end forward solution;
           jlim:=n;
           for i:=n-1 step -1 until 1 do
                 begin sum:=0.0;  k:=2;
                       for j:=i+1 step 1 until jlim do
                             begin sum:=sum+b[j]*a[i,k];  k:=k+1  end;
                                   b[i]:=b[i]-sum;
                                   if jlim+2-i>lim then jlim:=jlim-1
                 end back substitution
end bandsol;
```

SOLVE BAND EQUATIONS Ax=b

procedure bandmx (a,n,m,b,eps,singular); value n,m,eps;

real eps; integer m,n; label singular; array a,b;

comment HUCC LIBRARY PROCEDURE LE 24

Author : J. Boothroyd

Solves the n^{th} order linear equation Ax=b where the elements of A form a band matrix of width m. The band elements of A[1:n,1:n] must occupy the array a[1:n,1:m] in accordance with the rules:-

A[i,j] occupies a[i,j] $1 \le i \le (m+1)$ div 2

A[i,j] occupies a[i,j-i+(m+1) div 2] $(m+1)$ div 2 $< i \le n$

Diagrammatically

A11	A12	A13	0	0	0		A11	A12	A13	0	0
A21	A22	A23	A24	0	0		A21	A22	A23	A24	0
A31	A32	A33	A34	A35	0	→	A31	A32	A33	A34	A35
0	A42	A43	A44	A45	A46		A42	A43	A44	A45	A46
0	0	A53	A54	A55	A56		A53	A54	A55	A56	0
0	0	0	A64	A65	A66		A64	A65	A66	0	0

Array A[1:n,1:n] array a[1:n,1:m]

UNUSED ELEMENTS of matrix a must be set to zero.

The procedure uses maximum column pivots and the solution vector replaces the elements of vector b[1:n]. eps is a singularity tolerance and the procedure exits to the label singular if the matrix is singular or excessively ill conditioned;

```
procedure bandmx(a,n,m,b,eps,singular);  value n,m,eps;  real eps;  integer m,n;  label  singular;  array a,b;
begin      integer i,j,k,nless1,lim,piv,ri,rk,iless1;  real pivot,bri,ark1,sum;  integer array r[1:n];
           for i:=1 step 1 until n do r[i]:=i;  nless1:=n-1;  lim:=(m+1) div 2;
           for i:=1 step 1 until nless1 do
                begin piv:=i;  pivot:=a[r[i],1];

                      for k:=i+1 step 1 until lim do
                      if abs(a[r[k],1])>abs(pivot) then begin piv:=k;  pivot:=a[r[k],1] end;
                      if abs(pivot)<eps then begin print punch(3),  ££l?MATRIX SINGULAR?;  goto singular end;
                      ri:=r[piv];  r[piv]:=r[i];  r[i]:=ri;
                      bri:=b[ri]:=b[ri]/pivot;
                      for j:=2 step 1 until m do a[ri,j]:=a[ri,j]/pivot;
                      for k:=i+1 step 1 until lim do
                          begin rk:=r[k];  ark1:=a[rk,1];
                                b[rk]:=b[rk]-ark1*bri;
                                for j:=2 step 1 until m do a[rk,j-1]:=a[rk,j]-ark1*a[ri,j];
                                a[rk,m]:=0.0
                          end;
                      if lim≠n then lim:=lim+1
                end;
           ri:=r[n];  b[ri]:=b[ri]/a[ri,1];
           lim:=2;

           for i:=nless1 step -1 until 1 do
                begin iless1:=i-1;  sum:=0.0;  ri:=r[i];
                      for j:=2 step 1 until lim do sum:=sum+a[ri,j]*b[r[iless1+j]];
                      b[ri]:=b[ri]-sum;  if lim≠m then lim:=lim+1
                end;
           for i:=1 step 1 until n do a[i,1]:=b[r[i]];
           for i:=1 step 1 until n do b[i]:=a[i,1]
end bandmx;
```

LE 22

REARRANGE PERMUTED TRIANGULAR MATRIX

procedure MOVEU(a)to:(b)order:(n)pivots:(r); **value** n;

integer n; **array** a,b; **integer array** r;

comment HUCC LIBRARY PROCEDURE LE 22

Author : J. Boothroyd

LIBRARY PROCEDURE LE 10 decomposes a matrix a into quasi triangular interlocking matrices L (a unit lower triangle) and U, an upper triangle which occupy the array a. This procedure separates L and U, replacing the elements of U formerly in a by zeroes and rearranging U so that it is a proper upper triangular matrix in array b. The permuted 'diagonal' elements of L are set equal to one;

LE 23

procedure MOVEL(a) to: (b) order: (n) pivots : (r); **value** n;

integer n; **array** a,b; **integer array** r;

comment HUCC LIBRARY PROCEDURE LE 23

Author : J. Boothroyd

LIBRARY PROCEDURE LE 03 decomposes a matrix a into quasi triangular interlocking matrices L, a lower triangle and U, a unit upper triangle, which together occupy the array a. This procedure separates L and U, replacing the elements of L formerly in array a by zeroes and rearranging L so that it becomes a proper triangular matrix in array b.

The permuted 'diagonal' elements of U are at equal to one;

LE 22

```
procedure MOVEU(a)to:(b)order:(n)pivots:(r);  value n;
integer n;  array a,b;  integer array r;
begin integer i,ri;
      for i:=1 step 1 until n do
            begin ri:=r[i];
                  b[i,i]:=a[ri,i];  a[ri,i]:=1.0;
                  for j:=i+1 step 1 until n do
                        begin b[i,j]:=a[ri,j];  b[j,i]:=a[ri,j]:=0.0 end
            end
endMOVE;
```

LE 23

```
procedure MOVEL(a)to:(b)order:(n)pivots:(r);  value n;
integer n;  array a,b;  integer array r;
begin integer i,ri;
      for i:=1 step 1 until n do
            begin ri:=r[i];
                  b[i,i]:=a[ri,i];  a[ri,i]:=1.0;
                  for j:=i-1 step -1 until 1 do
                        begin b[i,j]:=a[ri,j];  b[j,i]:=a[ri,j]:=0.0 end
            end
end MOVE;
```

LE 24

DECOMPOSE SYMMETRIC POSITIVE DEFINITE MATRIX

```
procedure SYMDET(a,n,symdet,fail); value n;
integer n; real symdet; array a; label fail;
```

comment HUCC LIBRARY PROCEDURE LE 24:

AUTHOR J. BOOTHROYD:

Procedure choleski (LE 16) performs the symmetric decomposition of a symmetric matrix stored in conventional form. This procedure performs an in-situ decomposition of the upper triangle of a symmetric matrix stored in vector form, permitting the solution, in Algol, of symmetric equations of order 100 in a 503 with 8K store. The time is approximately 90 seconds.

The upper triangle of A[1:n,1:n] is stored in a[1:n*(n+1)div2] by columns.

i.e. The upper triangle of

A11	A12	A13	A14
A21	A22	A23	A24
A31	A32	A33	A34
A41	A42	A43	A44

is stored as :-

a										
	A11	A12	A22	A13	A23	A33	A14	A24	A34	A44

and element Ai,j is referenced by a[i+j*(j-1)div2].

To avoid repeated evaluation of subscripts of this form it is preferable to set up a column index vector c[1:n] such that c[j]:=j*(j-1)div2, permitting references to Ai,j as a[i+c[j]]. This procedure assumes the existance of such a global vector which is best initialised by a recursion relation before entry to the procedure using :-

```
r:=0; for j:=1 step 1 until n do begin c[j]:=r;r:=r+j end
```

If the procedure is non-positive-definite exit to the label fail occurs. On normal exit from the procedure symdet yields the determinant of the matrix A;

FORWARD AND BACK SOLUTION Ax=b, SYMMETRIC A

procedure SYMSOL(a,b,n); **value** n; **integer** n; **array** a,b;

comment HUCC LIBRARY PROCEDURE LE 25:

AUTHOR J. BOOTHROYD:

Solves the equation Ax=b for symmetric A[1:n,1:n] by the successive operations Lf=b,L^Tx=f where A=LL^T. The triangular matrix L^T should occupy the vector a[1:n*(n+1)**div**2] using column storage, as it will if generated by LE 24, in the description of which procedure find details of the method of storage. The procedure assumes the existance of a global array c[1:n] initialised before entry to the procedure so that c[j]:=j*(j-1)**div**2, permitting access to an element L^Ti,j by reference to a[i+c[j]]. On exit from the procedure the elements of the solution vector occupy b[1:n];

LE 24

```
procedure SYMDET(a,n,symdet,fail);  value n;  integer n;  real symdet;  array a;  label fail;
begin              integer i,j,k,iless1,ci,ii,cj,ij;
        real det,aii,aki,aij;
        det:=1.0;
        for i:=1 step 1 until n do
              begin iless1:=i-1;  ci:=c[i];  ii:=i+ci;  aii:=a[ii];
                    for k:=1 step 1 until iless1 do
                          begin aki:=a[k+ci];  aii:=aii-aki*aki  end;
                                if aii<0.0 then goto fail;
                                det:=det*aii;
                                aii:=a[ii]:=sqrt(aii);
                                for j:=i+1 step 1 until n do
                                      begin cj:=c[j];  ij:=i+cj;  aij:=a[ij];
                                            for k:=1 step 1 until iless1 do aij:=aij-a[k+ci]*a[k+cj];
                                            a[ij]:=aij/aii
                                      end j
              end i;
        symdet:=det
end SYMDET;
```

LE 25

```
procedure SYMSOL(a,b,n);  value n;  integer n;  real array a,b;
begin              integer i,j,jless1,cj;  real bi,bj;
        for j:=1 step 1 until n do
              begin bj:=b[j];  jless1:=j-1;  cj:=c[j];
                    for i:=1 step 1 until jless1 do bj:=bj-a[i+cj]*b[i];
                    b[j]:=bj/a[j+cj]
              end;
        for i:=n step -1 until 1 do
              begin bi:=b[i];
                    for j:=i+1 step 1 until n do bi:=bi-a[i+c[j]]*b[j];
                    b[i]:=bi/a[i+c[i]]
              end
end SYMSOL;
```

LE 26

DECOMPOSE BAND MATRIX WITH PIVOTING

procedure piv[illegible]rband(a,n,m,r,eps,singular); **value** n,m,eps;
integer n,m; **real** eps; **array** a; **label** singular; **integer array** r;

comment HUCC LIBRARY PROCEDURE LE 26:

AUTHOR J. BOOTHROYD:

Decomposes the n^{th} order bandmatrix of width m stored in columns 1:m of a[1:n,1:m+m div2] into a lower 'triangle'L and a strictly upper 'triangle' U. At entry to the procedure the elements of band matrix A[1:n,1:n] must occupy the first m columns of a in accordance with the rules:-

A[i,j] occupies a[i,j] $1 \le i < (m+1)$ div2

A[i,j] occupies a[i,j-i+(m+1) div2] $(m+1)$ div2 $< i \le n$

Diagrammatically :-

A11	A12	A13	0	0	0
A21	A22	A23	A24	0	0
A31	A32	A33	A34	A35	0
0	A42	A43	A44	A45	A46
0	0	A53	A54	A55	A56
0	0	0	A64	A65	A66

Array A[1:n,1:n]

→

A11	A12	A13	0	0	0	0
A21	A22	A23	A24	0	0	0
A31	A32	A33	A34	A35	0	0
A42	A43	A44	A45	A46	0	0
A53	A54	A55	A56	0	0	0
A64	A65	A66	0	0	0	0

Array a[1:n,1:m+m div2]

UNUSED ELEMENTS OF MATRIX a should be set to zero.

On exit from the procedure the elements of L occupy column 1 and the last m div2 columns of a. The elements of U (omitting the unit diagonal elements) occupy the remainder of matrix a. The procedure utilises column pivoting without row interchanges with two consequences (a) the algorithm is fast and (b) the storage scheme for elements of L is complicated. All information relating to the succession of pivotal rows is however retained in the index vector r[1:n] for subsequent use by LE 27. If the matrix is singular, eps is the tolerance, the procedure exits via the label parameter;

LE 26

```
procedure pivlrband(a,n,m,r,eps,singular);  value n,m,eps;  integer n,m;
real eps;  label singular;  array a;  integer array r;
begin          integer i,j,k,p,ri,rk,piv,nless1,lim;  real pivot,ark1;
     for i:=1 step 1 until n do r[i]:=i;
     lim:=(m+1) div 2;  nless1:=n-1;
     for i:=1 step 1 until nless1 do
          begin piv:=i;  ri:=r[i];  pivot:=a[ri,1];
               for k:=i+1 step 1 until lim do
               if abs(a[r[k],1])>abs(pivot) then begin piv:=k;  pivot:=a[r[k],1]  end;
               if abs(pivot)<eps then begin print punch(3),££l?MATRIX SINGULAR?;  goto singular  end;
               if piv≠i then begin ri:=r[piv];  r[piv]:=r[i];  r[i]:=ri  end;
               for j:= 2 step 1 until m do a[ri,j]:=a[ri,j]/pivot;
               p:=m;
               for k:=i+1 step 1 until lim do
                    begin p:=p+1;  rk:=r[k];  ark1:=a[ri,p]:=a[rk,1];
                         for j:=2 step 1 until m do a[rk,j-1]:=a[rk,j]-ark1*a[ri,j];
                         a[rk,m]:=0.0
                    end;
               if lim≠n then lim:=lim+1
          end
end pivlrband;
```

FORWARD AND BACK SOLUTION Ax=b, BANDMATRICES

```
procedure pivsolband(a,b,n,m,r); value m,n; integer m,n;
array a,b; integer array r;

comment  HUCC LIBRARY PROCEDURE LE 27:
         AUTHOR   J. BOOTHROYD:

         Solves the equation Ax=b where A is a bandmatrix of order n, width m
         by the successive operations Lf=b, Ux=f where the 'triangular' matrices
         L and U, such that LU=A, occupy the array a[1:n,1:m+mdiv2] following
         the use of the decomposition procedure LE 26.  r[1:n] is a pivotal
         index vector whose elements are generated by LE 26.  These are
         subsequently used by LE 27 to effect the correct sequence of operations
         in the forward solution and back substitution;
```

```
procedure pivsolband(a,b,n,m,r);  value m,n;  array a,b;  integer array r;   integer n,m;
begin              integer i,j,k,ri,rk,lim;  real bri;
       lim:=(m+1) div 2;
             begin integer array row[1:n];  switch s:=scan;
                   for i:=1 step 1 until n do row[i]:=i;
                   for i:=1 step 1 until n do
                         begin ri:=r[i];  j:=i;
                               if ri≠row[i] then begin scan:  j:=j+1;   if row[j]≠ ri then goto scan;
                                                       row[j]:=row[i];  row[i]:=ri
                                                 end;
                               bri:=b[ri]:=b[ri]/a[ri,1];
                               j:=m;
                               for k:=i+1 step 1 until lim do
                                     begin j:=j+1;  rk:=row[k];  b[rk]:=b[rk]-bri*a[ri,j]  end;
                               if lim≠ n then lim:=lim+1
                         end
             end forward solution;
             begin real array bi[1:n];
                   bi[n]:=b[r[n]];
                   for i:=n-1 step -1 until 1  do
                         begin ri:=r[i];  bri:=b[ri];
                               k:=1;
                               for j:=i+1 step 1 until lim do
                                     begin k:=k+1;  bri:=bri-b[r[j]]*a[ri,k]  end;
                                           bi[i]:=b[ri]:=bri;
                                           if k=m then lim:=lim-1
                         end;
                               for i:=1 step 1 until n do b[i]:=bi[i]
             end backsubstitution
end pivsolband;
```

LE 28

DECOMPOSE SYMMETRIC POSITIVE DEFINITE MATRIX

<u>procedure</u> SYMDET(a,n,symdet,fail); <u>value</u> n; <u>integer</u> n;
<u>real</u> symdet; <u>array</u> a; <u>label</u> fail;

<u>comment</u> HUCC LIBRARY PROCEDURE LE 28:
AUTHOR J. BOOTHROYD:

Performs the same function as LE 24, using the same method of storing the elements of the upper triangle of A[1:n,1:n] as elements of a[1:n*(n+1)<u>div</u>2]. This procedure <u>does not</u>,however, use the global addressing vector c[1:n] necessary with LE 24. It is approximately 7% faster than LE 24.

LE 29

FORWARD AND BACKWARD SOLUTION OF Ax=b, SYMMETRIC

<u>procedure</u> SYMSOL (a,b,n); <u>value</u> n; <u>integer</u> n; <u>array</u> a,b;

<u>comment</u> HUCC LIBRARY PROCEDURE LE 29:
AUTHOR J. BOOTHROYD:

Performs the same function as LE 25 but avoids the need for the addressing vector c[1:n];

LE 28

```
procedure SYMDET(a,n,symdet,fail);  value n;  integer n;  real symdet;  array a;  label fail;
begin           integer i,j,ki,ii,k1,kiless1,ij,kj;
        real det,aii,aki,aij;
        det:=1.0;  ii:=1;
        for i:=1 step 1 until n do
               begin aii:=a[ii];  k1:=ii-i+1;  kiless1:=ii-1;
                      for ki:=k1 step 1 until kiless1 do
                             begin aki:=a[ki];  aii:=aii-aki*aki  end;
                      if aii<0 then goto fail;
                      det:=det*aii;
                      aii:=a[ii]:=sqrt(aii);
                      ij:=ii+i;
                      for j:=i+1 step 1 until n do
                             begin aij:=a[ij];  kj:=ij-i+1;
                                    for ki:=k1 step 1 until kiless1 do
                                           begin aij:=aij-a[ki]*a[kj];  kj:=kj+1  end;
                                                  a[ij]:=aij/aii;  ij:=ij+j

                             end;
                      ii:=ii+i+1;
               end;
        symdet:=det
end SYMDET;
```

LE 29

```
procedure SYMSOL(a,b,n);  value n;  integer n;  array a,b;
begin integer i,j,k,ii;  real sum;
       ii:=1;
       for i:=1 step 1 until n do
       begin k:=ii-1;  sum:=b[i];
        for j:=i-1 step -1 until 1 do
        begin sum:=sum-b[j]*a[k];  k:=k-1  end;
        b[i]:=sum/a[ii];  ii:=ii+i+1
       end forward solution;
       k:=ii-1;  ii:=ii-n-1;
       for i:=n step -1 until 1 do
       begin sum:=b[i];
        for j:=i+1 step 1 until n do
        begin sum:=sum-b[j]*a[k];  k:=k+j  end;
        b[i]:=sum/a[ii];  k:=ii-1;  ii:=ii-i
       end back substitution
end SYMSOL;
```

LE30

REDUCE MATRIX TO UPPER HESSENBERG FORM

<u>procedure</u> hessenberg(a,n); <u>value</u> n; <u>integer</u> n;

<u>comment</u> HUCC LIBRARY PROCEDURE LE30:

AUTHOR : J. BOOTHROYD:

Transforms a real matrix a[1:n,1:n] to upper Hessenberg form by a sequence of elementary similarity transformations. The transformation is performed using a pivoting strategy which involves row and column interchanges. The method is similar to Gaussian elimination. For a compact CROUT-like method which converts a matrix to lower Hessenberg form see <u>procedure</u> TRINGLE in Parlett's paper "Languerre's Method Applied to the Matrix Eigenvalue Problem" Math. Comp v18 1964 469 - 485;

```
procedure hessenberg(a,n);  value n;  integer n;  array a;
begin integer i,j,k,iless1,iplus1,piv,nless1;  real pivot,mk,aki,temp;
      array m[3:n];  boolean nonzero;  switch s:=pivsearch;
      pivot:=0.0;  nless1:=n-1;
      for i:=1 step 1 until nless1 do
      begin iplus1:=i+1;
            if pivot=0.0 then goto pivsearch;
            if i≠piv then
            begin for j:=iless1 step 1 until n do begin temp:=a[i,j];  a[i,j]:=a[piv,j];  a[piv,j]:=temp  end;
                  for k:=1 step 1 until n do begin temp:=a[k,i];  a[k,i]:=a[k,piv];  a[k,piv]:=temp  end
            end;
            nonzero:= false;
            for k:=iplus1 step 1 until n do
            begin mk:=m[k]:=a[k,iless1]/pivot;
                  if mk≠0.0 then begin nonzero:= true;  a[k,iless1]:=0.0  end
            end;
            if nonzero then
            begin for k:=iplus1 step 1 until n do
                  begin mk:=m[k];
                        for j:=i step 1 until n do a[k,j]:=a[k,j]-mk*a[i,j]
                  end;
                  piv:=iplus1;  pivot:=0.0;
                  for k:=1 step 1 until n do
                  begin aki:=a[k,i];
                        for j:=iplus1 step 1 until n do aki:=aki+m[j]*a[k,j];
                        a[k,i]:=aki;
                        if k>i then
                        begin if abs(aki)>abs(pivot) then begin piv:=k;  pivot:=aki end end
                  end
            end
                  else
pivsearch:  begin piv:=iplus1;  pivot:=a[piv,i];
                  for k:=i+2 step 1 until n do
                  begin aki:=a[k,i];
                        if abs(aki)>abs(pivot) then begin piv:=k;  pivot:=aki end
                  end
            end;
            iless1:=i
      end i
end hessenberg;
```

LE31

TRIDIAGONALIZE SYMMETRIC MATRIX GIVENS METHOD

procedure givens(a,s,n); **value** n; **integer** n; **array** a,s;

comment HUCC LIBRARY PROCEDURE LE31:

AUTHOR : J. BOOTHROYD:

Performs a similarity transformation of a full symmetric matrix a[1:n,1:n] to tridiagonal form by GIVENS Method. The orthogonal rotation matrices used have the typical form:

$$Pr = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & c & 0 & s \\ 0 & 0 & 1 & 0 \\ 0 & s & 0 & -c \end{matrix}$$

so that $P_r = P_r^{-1} = P_r^t$.

The continued product $P_1, P_2, P_3, \ldots$ of transformation matrices is finally available in array s[1:n,1:n];

LE31

```
procedure givens(a,s,n);  value n;  integer n;  array a,s;
begin integer i,p,q,nless1,k;
      real akp,akq,app,apq,aqq,aip,aiq,fac,cos,sin,cs;
      for i:=1 step 1 until n do
      begin s[i,i]:=1.0;
            for j:=i+1 step 1 until n do s[i,j]:=s[j,i]:=0.0
      end;
      nless1:=n-1;
      for p:=2 step 1 until nless1 do
      begin k:=p-1;  akp:=a[k,p];
            for q:=p+1 step 1 until n do
            begin akq:=a[k,q];
                  apq:=a[p,q];  app:=a[p,p];  aqq:=a[q,q];
                  fac:=sqrt(akp*akp+akq*akq);
                  if fac≠0.0 then
                  begin cos:=akp/fac;  sin:=akq/fac;
                        for i:=1 step 1 until n do
                        begin if i>p then
                              begin aip:=a[i,p];  aiq:=a[i,q];
                                    a[i,p]:=a[p,i]:=cos*aip+sin*aiq;
                                    a[i,q]:=a[q,i]:=sin*aip-cos*aiq
                              end;
                              aip:=s[i,p];  aiq:=s[i,q];
                              s[i,p]:=cos*aip+sin*aiq;
                              s[i,q]:=sin*aip-cos*aiq
                        end;
                        akp:=fac;  a[k,q]:=a[q,k]:=0.0;
                        cs:=cos*sin;  fac:=(cs+cs)*apq;
                        cos:=cos*cos;  sin:=sin*sin;
                        a[p,p]:=cos*app+sin*aqq+fac;
                        a[q,q]:=sin*app+cos*aqq-fac;
                        a[p,q]:=a[q,p]:=(app-aqq)*cs-apq*(cos-sin)
                  end
            end;
            a[k,p]:=a[p,k]:=akp
      end
end givens;
```

TRIDIAGONALIZE SYMMETRIC MATRIX GIVENS METHOD

<u>procedure</u> vecgivens(a,s,n); <u>value</u> n; <u>integer</u> n; <u>array</u> a,s;

<u>comment</u> HUCC LIBRARY PROCEDURE LE32:

AUTHOR : J. BOOTHROYD:

As for LE31 except that only the upper half of the symmetric matrix is stored, by columns, in the vector a[1:n*(n+1)<u>div</u>2]. The continued product $P_1, P_2, P_3 \ldots$ of orthogonal transformation matrices is available in s[1:n,1:n]. If this procedure is used as an alternative to LL09 and prior to LL10 and LL11 it will be necessary to transfer the diagonal and codiagonal elements of array a to vectors c[1:n] and b[1:n] respectively. This may be done by:-

```
jless1:=k:=1;  c[1]:=a[1];
for j:= 2 step 1 until n do
begin k:=k+j;  c[j]:=a[k];
     b[jless1]:=a[k-1];  jless1:=j
end;
b[n]:=0.0;
```

The eigenvectors of the tridiagonal system should be premultiplied by s[1:n,1:n] to yield the eigenvectors of a;

LE32

```
procedure vecgivens(a,s,n); value n; integer n; array a,s;
begin integer i,p,q,nless1,k,cp,cq,ci,ip,iq; integer array c[1:n];
    real akp,app,akq,apq,aqq,aip,aiq,fac,cos,sin,cs;
    for i:=1 step 1 until n do
    begin s[i,i]:=1.0
        for j:=i+1 step 1 until n do s[i,j]:=s[j,i]:=0.0
    end;
    p:=0; for i:=1 step 1 until n do begin c[i]:=p; p:=i+p end;
    nless1:=n-1;
    for p:=2 step 1 until nless1 do
    begin k:=p-1; cp:=c[p]; akp:=a[k+cp];
        for q:=p+1 step 1 until n do
        begin cq:=c[q]; akq:=a[k+cq];
            apq:=a[p+cq]; app:=a[p+cp]; aqq:=a[q+cq];
            fac:=sqrt(akp*akp+akq*akq);
            if fac≠0.0 then
            begin cos:=akp/fac; sin:=akq/fac;
                for i:=1 step 1 until n do
                begin if i>p then
                    begin ci:=c[i]; ip:=p+ci;
                        iq:= if i>q then q+ci else i+cq;
                        aip:=a[ip]; aiq:=a[iq];
                        a[ip]:=cos*aip+sin*aiq;
                        a[iq]:=sin*aip-cos*aiq
                    end;
                    aip:=s[i,p]; aiq:=s[i,q];
                    s[i,p]:=cos*aip+sin*aiq;
                    s[i,q]:=sin*aip-cos*aiq
                end i;
                akp:=fac; a[k+cq]:=0.0;
                cs:=cos*sin; fac:=(cs+cs)*apq;
                cos:=cos*cos; sin:=sin*sin;
                a[p+cp]:=cos*app+sin*aqq+fac;
                a[q+cq]:=sin*app+cos*aqq-fac;
                a[p+cq]:=(app-aqq)*cs-apq*(cos-sin)
            end
        end q;
        a[k+cp]:=akp
    end p
end vecgivens;
```

LL01

EIGENVALUES AND EIGENVECTORS OF SYMMETRIC MATRIX

```
procedure jacobi(a,s,n,rho);  value n,rho;  integer n;  real rho;
          array a,s;
comment   HUCC LIBRARY PROCEDURE LL01:
          AUTHOR      ACM 85          :

          The procedure finds all eigenvalues and eigenvectors of
          a real symmetric matrix by Jacobi's method as described
          in "Mathematical Methods for Digital Computers" edited
          by Ralston and Wilf.  The eigenvectors of a[1:n,1:n]
          are built up in s[1:n,1:n] the kth eigenvector
          occupying column k.  The corresponding eigenvalue
          occupies element a[k,k] of the original matrix.
          rho is the precision tolerance for the process which
          is terminated when, for every off diagonal element
          a[i,j], abs(a[i,j])< rho × norm1/n where norm1 is the
          square root of the sum of the squares of the off
          diagonal elements of a;
```

LL01

```
procedure jacobi(a,s,n,rho);
value n,rho;
integer n; real rho; array a,s;
      begin real norm1,norm2,thr,mu,omega,sint,cost,int1,v1,v2,v3;
            integer i,j,p,q,ind; switch ss:=main,main1;

            for i:=1 step 1 until n do
            for j:=1 step 1 until i do
            if i=j then s[i,j]:=1.0 else s[i,j]:=s[j,i]:=0;

            int1:=0;
            for i:=2 step 1 until n do
            for j:=1 step 1 until i-1 do int1:=int1+2*a[i,j]↑2;
            norm1:=sqrt(int1); norm2:=(rho/n)*norm1;
            thr:=norm1; ind:=0;

            main: thr:=thr/n;
            main1: for q:=2 step 1 until n do
            for p:=1 step 1 until q-1 do
            if abs(a[p,q]) ≥ thr then
                  begin ind:=1; v1:=a[p,p]; v2:=a[p,q]; v3:=a[q,q]; mu:=.5*(v1-v3);

                        omega:=if mu=0.0 then -1.0 else -sign(mu)*v2/sqrt(v2*v2+mu*mu);
                        sint:=omega/sqrt(2*(1+sqrt(1-omega*omega)));
                        cost:=sqrt(1-sint*sint);
                        for i:=1 step 1 until n do
                              begin int1:=a[i,p]*cost-a[i,q]*sint;
                                    a[i,q]:=a[i,p]*sint+a[i,q]*cost; a[i,p]:=int1;
                                    int1:=s[i,p]*cost-s[i,q]*sint;
                                    s[i,q]:=s[i,p]*sint+s[i,q]*cost; s[i,p]:=int1
                              end;
```

```
            for i:=1 step 1 until n do
                begin a[p,i]:=a[i,p]; a[q,i]:=a[i,q]
                end;

            a[p,p]:=v1*cost*cost+v3*sint*sint-2*v2*sint*cost;
            a[q,q]:=v1*sint*sint+v3*cost*cost+2*v2*sint*cost;
            a[p,q]:=a[q,p]:=(v1-v3)*sint*cost+v2*(cost*cost-sint*sint)
        end;

    if ind=1 then
        begin ind:=0; go to main1
        end
    else if thr > norm2 then go to main
end jacobi;
```

LL 02

SOLUTION OF THE EIGENPROBLEM $(A-\lambda B)x=0$

procedure eigensolve (A,B,n,eigen,eps,nonposdef); value n,eps;

integer n; real eps; array A,B,eigen; label nonposdef;

comment HUCC LIBRARY PROCEDURE LL 02

Author : J. Boothroyd.

Solves the equation $(A-\lambda B)x=0$ for symmetric A,B[1:n,1:n] provided one of these is positive definite. The equation is transformed into $(C-\lambda I)y=0$ where $C=L^{-1}A(L^T)^{-1}$ is symmetric and $y=L^Tx$. If B is non positive definite the Choleski decomposition fails in the attempted evaluation of a square root of a negative value. In this event the original equation is rearranged as $(B-\lambda A)x=0$ for which the eigenvalues are the reciprocals of $(A-\lambda B)x=0$ and the eigenvectors are unchanged. Failure of this second attempt causes exit to the label nonposdef.

At a successful exit the eigenvalues are in eigen[1:n] and the vectors occupy A. This procedure uses MS 01, the revised issues of LE 16, LE 17, together with LE 18 and LL 01. The parameter eps is a precision tolerance used by LL 01 for details of which see the appropriate commentary;

LL02

```
procedure eigensolve(A,B,n,eigen,eps,nonposdef); value n,eps; integer n;  real eps; array A,B,eigen; label nonposdef;
begin integer i,j; boolean recip; array C[1:n,1:n]; switch s:=newtry,swap,ok;

          comment temporary storage of the main diagonal of B;
          for i:= 1 step 1 until n do eigen[i]:=B[i,i];
          recip:=false; comment assumes B is positive definite;
 newtry:  choleski(B,n,swap);
          comment clear upper triangle of B;
          for i:= 1 step 1 until n do for j:= i+1 step 1 until n do B[i,j]:= 0.0;
          goto ok;
 swap:    if recip then goto nonposdef;  recip:=true;
          comment B was nonpositive definite. Swap A,B and try again;
          for i:= 1 step 1 until n do
          begin B[i,i]:=A[i,i];  A[i,i]:= eigen[i];
                for j:= i+1 step 1 until n do
                begin B[j,i]:=A[i,j];  A[i,j]:=A[j,i]:=B[i,j] end
          end;
          goto newtry;
 ok:      linv(B,n);  comment forms L-1 in B;
          mxmult(B,A,C,n,n,n);  comment  C:=L-1A;
          comment transpose L-1 and clear lower triangle of B;
          for i:= 1 step 1 until n do
          for j:= i+1 step 1 until n do begin B[i,j]:=B[j,i];  B[j,i]:=0.0 end;
          mxmult(C,B,A,n,n,n);  comment A:= C(Lt)-1 = L-1A(Lt)-1;
          jacobi(A,C,n,eps);   comment ACM85. Eigenvalues on A diagonal, y vectors in C;
          for i:= 1 step 1 until n do eigen[i]:= if recip then 1.0/A[i,i] else A[i,i];
          mxmult(B,C,A,n,n,n);  comment x vectors = (Lt)-1y now in A;
end eigensolve;
```

LL03 to LL07

SOLVE SYMMETRIC EIGENSYSTEM

These procedures are taken from the LINEAR ALGEBRA Handbook Series of Numerische Mathematique 4 pp 354 - 376 (1962). Their author, J.H. Wilkinson of the National Physical Laboratory, England, is the recognised authority on this subject and these procedures are included in the HUCC Library as examples of classical algorithms in this field.

The following program was used to test four of the five procedures and indicates how to use the various parameters.

```
 TEST HOUSEHOLDER;
begin comment procedure declarations;
      begin integer n;  read n;
            begin integer i,j,mr,ml,t,m1;  real e,gamma,norm;
                  array z,a[1:n,1:n],c,b,w[1:n];
                  for i:=1 step 1 until n do
                  for j:=1 step 1 until i do read a[i,j];
                  householder(a,n,c,b);
                  tridibisection2(c,b,n,-20,n,0,30,₁₀-16,w,norm,m1);
                  tridiinverseiteration(c,b,n,w,norm,m1,₁₀-8,z);
                  backtransformation(a,b,z,n,m1);
                  print ££l?EIGENVALUES£l2??;
                  sameline;  scaled(9);
                  for i:=1 step 1 until m1 do print w[i];
                  print ££l5?EIGENVECTORS£l??;
                  for i:=1 step 1 until n do
                        begin print ££l2??;
                              for j:=1 step 1 until n do print z[j,i],££s2??
                        end i
            end
      end
end;
```

LL03

HOUSEHOLDER TRIDIAGONALIZATION, FULL MATRIX

<u>procedure</u> householder tridiagonalization(a,n,c,b);

<u>value</u> n; <u>integer</u> n; <u>array</u> a,b,c;

<u>comment</u> HUCC LIBRARY PROCEDURE LL03:

AUTHOR : J.H. WILKINSON:

The symmetric matrix a[1:n,1:n] is reduced to a symmetric tridiagonal matrix using only the lower triangle of a. The diagonal elements of the tridiagonal matrix are stored in c[1:n], the subdiagonal elements in b[1:n] with b[n]=0. Sufficient details of the transformation are retained in a and b to enable the eigenvectors of a to be formed from the tridiagonal eigenvectors using <u>procedure</u> backtransformation;

LL04

EIGENVALUES OF SYMMETRIC TRIDIAGONAL MATRIX 1

<u>procedure</u> tridibisection 1(c,b,n,gu,go,t,gamma,w,norm,m1);

<u>value</u> n,gu,go,t,gamma; <u>integer</u> n,t,m1;

<u>real</u> gu,go,gamma,norm; <u>array</u> c,b,w;

<u>comment</u> HUCC LIBRARY PROCEDURE LL04:

AUTHOR : J.H. WILKINSON:

c[1:n] and b[1:n] are the diagonal and sub-diagonal of a symmetric tridiagonal matrix. The procedure determines the number m1 of eigenvalues lying between gu and go and computes these eigenvalues in w[1:n] in decreasing order of magnitude by the method of bisection. t is the number of bisection steps (30 to 35is appropriate on a 503), norm is the infinity norm of the matrix and gamma is the square of the relative machine precision (10^{-16} for a 503);

LL05

EIGENVALUES OF SYMMETRIC TRIDIAGONAL MATRIX 2

procedure tridibisection 2(c,b,n,e,mr,mℓ,t,gamma,w,norm,ml);

value n,e,mr,mℓ,t,gamma; real e,gamma,norm;

integer n,mr,mℓ,t,ml; array c,b,w;

comment HUCC LIBRARY PROCEDURE LL05:

AUTHOR : J.H. WILKINSON:

c[1:n] and b[1:n] are the diagonal and subdiagonal of a symmetric tridiagonal matrix. The mr eigenvalues to the left of e and the mℓ eigenvalues to the right of e are computed and stored in w[1:n]. t is the number of bisection steps(30to35 for a 503), norm is the infinity norm of the matrix and gamma is the square of the relative machine precision (10^{-16} for 503);

LL06

EIGENVECTORS OF TRIDIAGONAL MATRIX

procedure tridiinverseiteration(c,b,n,w,norm,wl,macheps,z);

value n,ml,norm,macheps; integer n,ml; real norm,macheps;

array c,b,w,z;

comment HUCC LIBRARY PROCEDURE LL06:

AUTHOR : J.H. WILKINSON:

c[1:n] and b[1:n] are the diagonal and subdiagonal of a symmetric tridiagonal matrix. w[1:n] contains the ml eigenvalues from w[1] to w[ml]. norm is the infinity norm of the matrix and macheps is the relative machine precision (10^{-8} on a 503). The ml eigenvectors are computed and stored in z[1:n,1:n], element z[i,j] being the j^{th} component of the eigenvector corresponding to the eigenvalue w[i];

LL07

EIGENVECTORS OF SYMMETRIC MATRIX

procedure backtransformation(a,b,z,n,ml); value n,ml; integer n,ml; array a,b,z;

comment HUCC LIBRARY PROCEDURE LL07:

AUTHOR : J.H. WILKINSON:

b[1:n] is the subdiagonal of a symmetric tridiagonal matrix whose eigenvectors occupy z[1:n,1:n],by rows. The eigenvectors of the original matrix a[1:n,1:n] are derived and overwritten on array z;

LL03

```
procedure householder tridiagonalisation(a,n)result:(c,b);
value n;  integer n;  array a,b,c;
begin integer j,i,k;  real ai,sigma,h,bj,bigk,bi;  array q[1:n-1];
      for i:=n step -1 until 3 do
      begin sigma:=0;
            for k:=1 step 1 until i-1 do
            sigma:=sigma+a[i,k]*a[i,k];
            ai:=a[i,i-1];
            if ai ≥ 0 then bi:=-sqrt(sigma) else bi:=sqrt(sigma);
            b[i-1]:=bi;
            if bi≠0 then
            begin h:=sigma-ai*bi;
            a[i,i-1]:=ai-bi;
                  for j:=i-1 step -1 until 1 do
                  begin bj:=0;
                        for k:=i-1 step -1 until j do
                        bj:=bj+a[k,j]*a[i,k];
                        for k:=j-1 step -1 until 1 do
                        bj:=bj+a[j,k]*a[i,k];
                        q[j]:=bj/h
                  end j;
            bigk:=0;
            for j:=i-1 step -1 until 1 do
            bigk:=bigk+a[i,j]*q[j];
            bigk:=bigk/(2*h);
            for j:=i-1 step -1 until 1 do
            q[j]:=q[j]-bigk*a[i,j];
            for j:=i-1 step -1 until 1 do
            begin for k:=j step -1 until 1 do
                  a[j,k]:=a[j,k]-a[i,j]*q[k]-a[i,k]*q[j];
            end j
            end
      end i;
      for i:=n step -1 until 1 do
      c[i]:=a[i,i];
      b[1]:=a[2,1];
      b[n]:=0
end;
```

```
procedure tridibisection1(c,b,n,gu,go,t,gamma)result:(w,norm,m1);
value n,gu,go,t,gamma;
integer n,t,m1;
real gu,go,gamma,norm;
array c,b,w;
      begin integer i,j,k,a1,a2,d;
                real l,g,h,lambda,p1,q1,y;  switch ss:=noeigenvalue;
                array p[1:n];
                procedure sturms sequence;
                begin   p1:=0;  q1:=1;  a1:=0;
                        for i:=1 step 1 until n do
                              begin y:=(c[i]-lambda)*q1-p[i]*p1;
                                    p1:=q1;  q1:=y;
                                    if p1 ≥ 0 ≡ q1 ≥ 0 then a1:=a1+1
                              end i;
                        if q1=0 and p1>0 then a1:=a1-1
                end;

                        if gu>go then
                              begin g:=gu;  gu:=go;  go:=g end;
                        norm:=abs(c[1])+abs(b[1]);
                        for i:=2 step 1 until n do
                              begin l:=abs(b[i-1])+abs(c[i])+abs(b[i]);
                                    if l>norm then norm:=l
                              end;
                        for i:=1 step 1 until n-1 do
                              begin if b[i]=0 then p[i+1]:=gamma*norm*norm
                                    else p[i+1]:=b[i]*b[i]
                              end;
                        p[1]:=0;
                        if gu>norm or go<-norm then
                              begin m1:=0;  goto noeigenvalue end;
                        lambda:=gu;

                        sturms sequence;
                        a2:=a1;
                        if q1=0 then a2:=a1+1;
                        lambda:=go;
                        sturms sequence;
                        m1:=a2-a1;
                        d:=a1;
                        if go>norm then go:=norm;
                        if gu<-norm then gu:=-norm;
                        for k:=1 step 1 until m1 do
                              begin d:=d+1;
                                    g:=go;  h:=gu;
                                    for j:=1 step 1 until t do
                                          begin lambda:=(g+h)/2;
                                                sturms sequence;
                                                if a1 ≥ d then h:=lambda else
                                                              g:=lambda
                                          end j;
                                    w[k]:=(g+h)/2
                              end k;
noeigenvalue: end;
```

```
procedure tridibisection 2 (c,b,n,e,mr,ml,t,gamma) result:(w,norm,m1);
value n,e,mr,ml,t,gamma;
integer n,mr,ml,t,m1;
real e,gamma,norm;
array c,b,w;
begin       integer d1,d2,i,j,k,a1,d;
            real l,g,h,lambda,p1,q1,y;
            array p[1:n];
            procedure sturms sequence;
            begin       p1:=0;  q1:=1;  a1:=0;
                        for i:=1 step 1 until n do
                        begin       y:=(c[i]-lambda)*q1-p[i]*p1;
                                    p1:=q1;  q1:=y;
                                    if p1 ≥ 0 ≡ q1 ≥ 0
                                    then a1:=a1+1;
                        end i;
            if q1=0 and p1>0 then a1:=a1-1
            end;
            norm:=abs(c[1])+abs(b[1]);
            for i:=2 step 1 until n do
            begin       l:=abs(b[i-1])+abs(c[i])+abs(b[i]);
                        if l>norm then norm:=l
            end;

            for i:=1 step 1 until n-1 do
            begin       if b[i]=0 then p[i+1]:=gamma*norm*norm
                        else p[i+1]:=b[i]*b[i]
            end;

            p[1]:=0;
            lambda:=e;
            sturms sequence;

            d1:=a1-mr;  d2:=a1+ml;
            if d1<0 then d1:=0;
            if d2 ≥ n+1 then d2:=n;
            d:=d1;  m1:=0;
            for k:=d1 step 1 until d2-1 do
            begin       d:=d+1;
                        g:=norm;  h:=-norm;
                        for j:=1 step 1 until t do
                        begin       lambda:=(g+h)/2;
                                    sturms sequence;
                                    if a1 ≥ d then h:=lambda else g:=lambda
                        end j;
                        m1:=m1+1;
                        w[m1]:=(g+h)/2
            end k
end;
```

```
procedure tridiinverse iteration(c,b,n,w,norm,m1,macheps)results:(z);
value n,m1,norm,macheps;  integer n,m1;  real norm,macheps;  array c,b,w,z;
begin integer i,j;  real bi,bi1,z1,lambda,u,s,v,h,eps,eta;
      array m,p,q,r,int[1:n],x[1:n+2];
      lambda:=norm;  eps:=macheps*norm;
      for j:=1 step 1 until m1 do
      begin lambda:=lambda-eps;
            if w[j]<lambda then lambda:=w[j];
            u:=c[1]-lambda;  v:=b[1];
            if v=0 then v:=eps;
            for i:=1 step 1 until n-1 do
            begin bi:=b[i];
                  if bi=0 then bi:=eps;
                  bi1:=b[i+1];
                  if bi1=0 then bi1:=eps;
                  if abs(bi) > abs(u) then
                  begin m[i+1]:=u/bi;
                        if m[i+1]=0 and  abs(bi) < eps then m[i+1]:=1;
                        p[i]:=bi;  q[i]:=c[i+1]-lambda;
                        r[i]:=bi1;
                        u:=v-m[i+1]*q[i];
                        v:=-m[i+1]*r[i];
                        int[i+1]:=1
                  end
                  else begin m[i+1]:=bi/u;
                        p[i]:=u;  q[i]:=v;
                        r[i]:=0;
                        u:=c[i+1]-lambda-m[i+1]*v;
                        v:=bi1;  int[i+1]:=-1
                        end
            end i;
            p[n]:=u;  q[n]:=r[n]:=0;
            x[n+1]:=x[n+2]:=0;  h:=0;  eta:=1/n;
            for i:=n step -1 until 1 do
                  begin u:=eta-q[i]*x[i+1]-r[i]*x[i+2];
                        if p[i]=0 then x[i]:=u/eps
                                  else x[i]:=u/p[i];
                        h:=h+abs(x[i])
                  end i;
            h:=1/h;
            for i:=1 step 1 until n do
            x[i]:=x[i]*h;
            for i:=2 step 1 until n do
                  begin if int[i]>0 then
                        begin u:=x[i-1];
                        x[i-1]:=x[i];
                        x[i]:=u-m[i]*x[i-1]
                        end
                  else x[i]:=x[i]-m[i]*x[i-1]
                  end i;
            h:=0;
            for i:=n step -1 until 1 do
                  begin u:=x[i]-q[i]*x[i+1]-r[i]*x[i+2];
                        if p[i]=0 then x[i]:=u/eps
                                  else x[i]:=u/p[i];
                        h:=h+x[i]*x[i]
                  end i;
            h:=1/sqrt(h);
            for i:=1 step 1 until n do
            z[j,i]:=x[i]*h
      end j
end;
```

LL07

```
procedure backtransformation(a,b,z,n,m1);
value n,m1;  integer n,m1;  array a,b,z;
begin integer i,j,k;  real s;  for j:=1 step 1 until m1 do
      for k:=3 step 1 until n do
      if b[k-1]≠0 then
      begin s:=0;
            for i:=1 step 1 until k-1 do
            s:=s+a[k,i]*z[j,i];
            s:=s/(b[k-1]*a[k,k-1]);
            for i:=1 step 1 until k-1 do
            z[j,i]:=z[j,i]+s*a[k,i]
      end
end;
```

LL08

EIGENVALUES AND VECTORS OF SYMMETRIC MATRIX

```
procedure vecjacobi(a,s,n,rho); value n,rho; real rho;
integer n; array a,s;
```

comment HUCC LIBRARY PROCEDURE LL08:

AUTHOR : J. BOOTHROYD:

A revised version of LL01, jacobi, such that only the upper half of the symmetric matrix is stored in the vector a[1:n*(n+1)div2]. Storage is by columns as described in LE24. On exit from the procedure the eigenvalues occupy elements a[1] through a[n] and the eigenvectors occupy the array s[1:n,1:n]. On a 20 x 20 matrix LL01 took 120 seconds, LL08 only 47 seconds. On the same matrix J.H. Wilkinson's procedures (LL03,04,06,07) took 27 seconds and their improved counterparts (LL09,10,11, 12) took 23 seconds;

LL08

```
procedure vecjacobi(a,s,n,rho);  value n,rho;  real rho;  integer n;  array a,s;
begin integer array c[1:n];  integer i,j,ci,cj,p,q,cp,cq,jless1,qless1,ip,iq;
      switch ss:=main,main1;
      real fac,aij,thr,norm1,norm2,apq,app,aqq,m,mu,lambda,cost,sint,aip,aiq,sip,siq,sincos;
      boolean ind;
      p:=0;  fac:=0.0;
      for i:=1 step 1 until n do
      begin s[i,i]:=1.0;  c[i]:=p;  p:=p+i;
            for j:=i+1 step 1 until n do s[i,j]:=s[j,i]:=0.0;
      end;
      for j:=2 step 1 until n do
      begin cj:=c[j];  jless1:=j-1;
            for i:=1 step 1 until jless1 do
            begin aij:=a[i+cj];  fac:=fac+2.0*aij*aij  end
      end;
      thr:=norm1:=sqrt(fac);  norm2:=rho*norm1/n;
main: thr:=thr/n;
main1: ind:=false;
      for q:=2 step 1 until n do
      begin cq:=c[q];  qless1:=q-1;
            for p:=1 step 1 until qless1 do
            begin apq:=a[p+cq];
                  if abs(apq) > thr then
                  begin cp:=c[p];  ind:= true;
                        app:=a[p+cp];  aqq:=a[q+cq];  m:=app-aqq;
                        lambda:=sign(m)*apq;  mu:=0.5*abs(m);
                        fac:=0.5/sqrt(lambda*lambda+mu*mu);
                        cost:=sqrt(0.5+mu*fac);  sint:=lambda*fac/cost;
                        for i:=1 step 1 until n do
                        begin ci:=c[i];
                              if i < p then begin ip:=i+cp;  iq:=i+cq end
                                       else begin ip:=p+ci;
                                                  iq:=if i>q then q+ci else i+cq
                                            end;
                              aip:=a[ip];  aiq:=a[iq];
                              sip:=s[i,p];  siq:=s[i,q];
                              s[i,p]:=cost*sip+sint*siq;
                              s[i,q]:=sint*sip-cost*siq;
                              a[ip]:=cost*aip+sint*aiq;
                              a[iq]:=sint*aip-cost*aiq
                        end i;
                        sincos:=sint*cost;  fac:=(apq+apq)*sincos;
                        sint:=sint*sint;  cost:=cost*cost;
                        a[p+cp]:=cost*app+sint*aqq+fac;
                        a[q+cq]:=sint*app+cost*aqq-fac;
                        a[p+cq]:=0.0
                  end
            end
      end;
      if ind then goto main1 else if thr>norm2 then goto main;
      for i:=2 step 1 until n do a[i]:=a[i+c[i]]
end vecjacobi;
```

LL09

```
procedure householder(a,n,c,b);  value n;  integer n;  real array a,b,c;
begin integer i,j,k,nless2,iplus1,cj,ck,kcj,ciplus1,nless1;  real s,aij,bi,h,rho,zj;
      real array z[1:n];
      nless2:=n-2;  nless1:=n-1;
      for i:=1 step 1 until nless2 do
      begin iplus1:=i+1;  s:=0.0;  k:=ciplus1:=col[iplus1];
            for j:=iplus1 step 1 until n do
            begin aij:=a[i+k];  k:=k+j;  s:=s+aij*aij end;
            aij:=a[i+ciplus1];  bi:=sqrt(s);
            if aij ≥ 0.0 then bi:=-bi;  b[i]:=bi;
            if bi ≠ 0.0 then
            begin h:=s-aij*bi;  a[i+ciplus1]:=aij-bi;
                  for j:=iplus1 step 1 until n do
                  begin s:=0.0;  cj:=col[j];
                        for k:=iplus1 step 1 until n do
                        begin ck:=col[k];
                              s:=s+a[i+ck]*a[if k<j then k+cj else j+ck]
                        end;
                        z[j]:=s/h
                  end;
                  s:=0.0;  k:=ciplus1;
                  for j:=iplus1 step 1 until n do begin s:=s+a[i+k]*z[j];  k:=k+j end;
                  rho:=s/(h+h);  k:=ciplus1;
                  for j:=iplus1 step 1 until n do begin z[j]:=z[j]-rho*a[i+k]; k:=k+j end;
                  for j:=iplus1 step 1 until n do
                  begin cj:=col[j];  aij:=a[i+cj];  zj:=z[j];  ck:=ciplus1;
                        for k:=iplus1 step 1 until j do
                        begin kcj:=k+cj;
                              a[kcj]:=a[kcj]-a[i+ck]*zj-aij*z[k];  ck:=ck+k
                        end
                  end
            end;
            c[i]:=a[i+col[i]]
      end;
      cj:=col[n];  c[nless1]:=a[nless1+col[nless1]];
      b[nless1]:=a[nless1+cj];  b[n]:=0.0;  c[n]:=a[n+cj]
end householder;
```

SYMMETRIC EIGENSYSTEM HALF MATRIX

These procedures are adapted from LL03 to LL07 to compute the eigenvalues and eigenvectors of an n^{th} order symmetric matrix of which only the upper half is stored by columns in a vector a[1:n*(n+1)div2]. In designing this revision the opportunity has been taken to improve, where possible, the efficiency of the original procedures and rationalise, to some extent, the parameter organisation.

The following program was used to test LL09 to LL12 and illustrates the use of the several parameters:-

```
TEST HOUSEHOLDER;
begin integer n,t;  real eps;  switch ss:=L;
  L:  read n,eps,t;
      begin integer i,j,k;  real norm;
            array z[1:n,1:n],a[1:n*(n+1) div 2],c,b,w[1:n];  integer array col[1:n];
            comment procedure declarations;
            k:=0;
            for j:=1 step 1 until n do
            begin col[j]:=k;
                  for i:=1 step 1 until j do read a[i+k];
                  k:=k+j
            end;
            householder(a,n,c,b);
            eigbisec(c,b,n,eps,t,w,norm);
            trivectors(c,b,n,w,norm,eps,z);
            eigvectors(a,b,z,n);
            freepoint(9);  sameline;
            for j:=1 step 1 until n do
            begin print ££1??,w[j],££1??;
                  for i:=1 step 1 until n do
                  begin if i-i div 10*10=1 then print ££1??; print z[i,j] end
            end
      end;
      goto L
end;
```

LL09

HOUSEHOLDER TRIDIAGONALISATION - HALF MATRIX

<u>procedure</u> householder(a,n,c,b); <u>value</u> n; <u>integer</u> n; <u>array</u> a,c,b;

<u>comment</u> HUCC LIBRARY PROCEDURE LL09;

AUTHOR : J. BOOTHROYD:

Transforms an n^{th} order symmetric matrix whose upper half is stored by columns in a[1:n*(n+1)<u>div</u>2] to tridiagonal form the diagonal and super-diagonal stored in c[1:n] and b[1:n] respectively (with b[n]=0). Sufficient information is retained in a, and b for subsequent use by <u>procedure</u> eigvectors (LL12);

LL10

EIGENVALUES OF TRIDIAGONAL MATRIX

<u>procedure</u> eigbisec(c,b,n,eps,t,w,norm); <u>value</u> n,eps,t;
<u>integer</u> n,t; <u>real</u> eps,norm; <u>array</u> c,b,w;

<u>comment</u> HUCC LIBRARY PROCEDURE LL10:

AUTHOR : J. BOOTHROYD:

Evaluates, in decreasing order of magnitude, using the method of bisection, the n eigenvectors of a tridiagonal matrix whose diagonal and co-diagonal occupy c[1:n] and b[1:n] respectively. The results occupy w[1:n]. eps is the relative machine precision (10^{-8} on a 503), t is the number of bisection steps (30 to 35 is appropriate) and norm is the computed infinity norm of the matrix;

LL11

EIGENVECTORS OF TRIDIAGONAL MATRIX

<u>procedure</u> trivector(c,b,n,w,norm,eps,z); <u>value</u> n,eps,norm;
<u>integer</u> n; <u>real</u> norm,eps; <u>array</u> c,b,w,z;

<u>comment</u> HUCC LIBRARY PROCEDURE LL11:

AUTHOR : J. BOOTHROYD:

Computes in the <u>columns</u> of z[1:n,1:n] the eigenvectors of a symmetric tridiagonal matrix whose eigenvalues occupy w[1:n] in decreasing order of magnitude. c[1:n] and b[1:n] are the diagonal and co-diagonal of the tridiagonal matrix. eps is the relative machine precision (10^{-8} on a 503), norm is the infinity norm of the matrix;

LL12

EIGENVECTORS OF SYMMETRIC MATRIX (UPPER HALF)

```
procedure eigvectors(a,b,z,n); value n;
integer n; array a,b,z;
comment   HUCC LIBRARY PROCEDURE LL12;
          AUTHOR : J. BOOTHROYD:
          From the information retained in a[1:n*(n+1)div2] and b[1:n]
          following the use of LL09 and the eigenvectors of the tridiagonal
          matrix in z[1:n,1:n], computes the eigenvectors of the original
          matrix a;
```

LL10

```
procedure eigbisec(c,b,n,eps,t,w,norm); value n,eps,t; integer n,t;
real eps,norm; real array b,c,w;
begin integer i,k,a,j; real g,h,p1,q1,bi,lim,lambda,y; array p[1:n];
      g:=abs(b[1]); p1:=abs(c[1])+g;
      for i:=2 step 1 until n do
      begin h:=abs(b[i]); q1:=g+h+abs(c[i]);
            if q1>p1 then p1:=q1; g:=h
      end;
      lambda:=norm:=lim:=p1; bi:=b[1];
      for i:=2 step 1 until n do
      begin if bi=0.0 then bi:=eps*lim;
            p[i]:=bi*bi; bi:=b[i]
      end;
      for k:=1 step 1 until n do
      begin g:=lambda; h:=-lim;
            for j:=1 step 1 until t do
            begin lambda:=(g+h)/2.0;
                  p1:=0.0; q1:=1.0; a:=0;
                  for i:=1 step 1 until n do
                  begin y:=(c[i]-lambda)*q1-p[i]*p1;
                        p1:=q1; q1:=y;
                        if p1≥0.0 = q1 ≥ 0.0 then a:=a+1
                  end i;

                  if q1=0.0 and p1>0 then a:=a-1;
                  if a>k then h:=lambda else g:=lambda;
                  if g=h then j:=t
            end j;
            w[k]:=(g+h)/2.0
      end
end eigbisec;
```

```
procedure trivectors(c,b,n,w,norm,eps,z);  value n,eps,norm;
integer n;  real norm,eps;  array c,b,w,z;
begin integer i,j,iplus1,iless1,nless1;  array m,p,q,r,int,x[1:n];
      real bi,bi1,lambda,u,v,xi,miplus1,h,eta;
      lambda:=norm;  eps:=eps*norm;  nless1:=n-1;
      q[n]:=r[n]:=0.0;  eta:=1.0/n;
      for j:=1 step 1 until n do
      begin lambda:=lambda-eps;
            if w[j]<lambda then lambda:=w[j];
            u:=c[1]-lambda;  bi:=v:=b[1];
            if bi=0.0 then bi:=v:=eps;
            for i:=1 step 1 until nless1 do
            begin iplus1:=i+1;
                  bi1:=b[iplus1];  if bi1=0.0 then bi1:=eps;
                  if abs(bi)≥abs(u) then
                  begin miplus1:=m[iplus1]:=if u=0.0 and abs(bi)≤eps then 1.0 else u/bi;
                        p[i]:=bi;  bi:=r[i]:=bi1;
                        bi1:=q[i]:=c[iplus1]-lambda;
                        u:=v-miplus1*bi1;  v:=-miplus1*bi;
                        int[iplus1]:=1.0
                  end
                  else
                  begin miplus1:=m[iplus1]:=bi/u;
                        p[i]:=u;  q[i]:=v;  r[i]:=0.0;
                        u:=c[iplus1]-lambda-miplus1*v;
                        bi:=v:=bi1;  int[iplus1]:=-1.0
                  end
            end i;
            p[n]:=if u≠0.0 then u else eps;
             v:=u:=h:=0.0;
             for i:=n step -1 until 1 do
             begin xi:=x[i]:=(eta-q[i]*u-r[i]*v)/p[i];
                   v:=u;   u:=xi;   h:=h+abs(xi)
             end;
           * for i:=1 step 1 until n do x[i]:=x[i]/h;
             u:=x[1];  iless1:=1;
             for i:=2 step 1 until n do
             begin if int[i]>0.0 then begin v:=x[iless1]:=x[i];
                                            u:=x[i]:=u-m[i]*v
                                      end
                                  else u:=x[i]:=x[i]-m[i]*u;
                   iless1:=i
             end i;
             v:=u:=h:=0.0;
             for i:=n step -1 until 1 do
             begin xi:=x[i]:=(x[i]-q[i]*u-r[i]*v)/p[i];
                   v:=u;  u:=xi;  h:=h+xi*xi;
             end;
             h:=1.0/sqrt(h);
             for i:=1 step 1 until n do z[i,j]:=x[i]*h
       end j
 end trivectors;
```

LL12

```
procedure eigvectors(a,b,z,n);  value n;  integer n;  array a,b,z;
begin integer i,j,k,iplus1,ck,ciplus1;  real s,bi;
      for j:=1 step 1 until n do
      begin for i:=n-2 step -1 until 1 do
            begin bi:=b[i];
                  if bi≠0.0 then
                  begin s:=0.0;  iplus1:=i+1;  ck:=ciplus1:=col[iplus1];
                        for k:=iplus1 step 1 until n do begin s:=s+z[k,j]*a[i+ck];  ck:=ck+k end;
                        s:=s/(bi*a[i+ciplus1]);  ck:=ciplus1;
                        for k:=iplus1 step 1 until n do begin z[k,j]:=z[k,j]+a[i+ck]*s;  ck:=ck+k end
                  end
            end
      end
end eigvectors;
```

LZ 01

GENERATE UNSYMMETRIC TEST MATRIX

<u>procedure</u> testmx(a,n); <u>value</u> n; <u>integer</u> n; <u>array</u> a;

<u>comment</u> HUCC LIBRARY PROCEDURE LZ 01:

AUTHOR J. BOOTHROYD:

The procedure generates matrices of the type described by T.J.Dekker, Report No. MR 63, Mathematical Centre, Amsterdam.
The matrices have the following properties :-

(a) elements a[i,j] are integers

(b) elements of the inverse, $(-1)^{i+j}$*a[i,j],are also integers

(c) the degree of ill condition increases rapidly with n

(d) the determinant of all matrices is 1.

Computation of a matrix order 15 is possible on the 503 with a 39 bit register. As <u>real</u> matrices however the order of the largest useable matrix should be restricted to that value of n(12) for which all elements of the matrix have an exact floating point representation;

```
procedure testmx(a,n);  value n;  integer n;  array a;
      begin integer i,j,k,fi,gi,d,q,r;boolean even;  integer array f,g[1:n];
       comment first we compute F = diag(fi);
       fi:=f[1]:=n;  j:=n*n;
       for i:=1 step 1 until n-1 do
             begin d:=i*i;  k:=j-d;
                    q:=fi div d;  r:=fi-q*d;
                    f[i+1]:=fi:=q*k+(r*k) div d
             end;
       comment and now,using a modified prime factors algorithm to obtain G = diag(gi) we compute
       FG-1,whose elements replace those of F;
       for i:=1 step 1 until n do
             begin d:=gi:=1;  q:=fi:=f[i];  j:=2;
         newj:      even:= false;
         next:      if q > j then
                           begin q:=fi  div j;
                                  if fi≠q*j then begin j:=j+d;  d:=2;  goto newj end;
                                  if even then gi:=gi*j; even:= not even;
                                  fi:=q;  goto next
                           end;
                    g[i]:=gi;  f[i]:=f[i] div gi
             end;
       comment finally,in one operation (FG-1)HG where H is a non-existent Hilbert
                matrix whose reciprocal elements ,i+j-1,are computed as we go;
       for i:=1 step 1 until n do
             begin fi:=f[i];
                    for j:=1 step 1 until n do
                           begin gi:=g[j];  k:=i+j-1;
                                  q:=fi div k;  r:=fi-q*k;
                                  a[i,j]:=q*gi+(r*gi) div k
                           end
             end
      end testmx;
```

LEAST SQUARES POLYNOMIAL FIT

procedure LSQFIT(x,y,N,a,n); **value** N,n; **integer** N,n;

real array x,y,a;

comment HUCC LIBRARY PROCEDURE MC01:

AUTHOR J. BOOTHROYD :

Given the N+1 sample pairs $(x_0,y_0),(x_1,y_1) \ldots\ldots (x_N,y_N)$ in arrays x,y[0:N] the procedure determines the coefficients $a_1,a_2,a_3 \ldots\ldots a_{n+1}$ of the nth order polynomial approximation

$$y = a_1 + a_2x + a_3x^2 + \ldots\ldots a_{n+1}x^n$$

obtained from applying the least squares principle to the given data. The coefficients $a_1,a_2 \ldots\ldots a_{n+1}$ occupy the corresponding element positions of a[1:n+1]. The procedure uses SOLVE (LE02) to obtain the solutions of the normal equations. The matrix of coefficients of the normal equations becomes increasingly ill-conditioned as n increases, and it is recommended that the use of this procedure be restricted to values of n < 6;

MC01

```
procedure LSQFIT(x,y,N,a,n);  value N,n;  integer N,n;  real array x,y,a;
begin      integer i,j,k,nplus1,nplus2;  real p,q,xk;  real array b[1:n+1,1:n+2];
           nplus1:=n+1;  nplus2:=n+2;
           for i:=1 step 1 until nplus1 do
                 begin b[i,nplus2]:=0.0;
                       for j:=1 step 1 until i do b[i,j]:=0.0
                 end;
           for k:=0 step 1 until N do
                 begin p:=1.0;  xk:=x[k];
                       for i:=1 step 1 until nplus1 do
                             begin q:=1.0;
                                   for j:=1 step 1 until i do
                                         begin b[i,j]:=b[i,j]+p*q;  q:=q*xk end;
                                               b[i,nplus2]:=b[i,nplus2]+p*y[k];
                                               p:=p*xk
                             end
                 end;
           for i:=2 step 1 until nplus1 do
              for j:=i-1 step -1 until 1 do b[j,i]:=b[i,j];
                 SOLVEQ(b,nplus1);
           for i:=1 step 1 until nplus1 do a[i]:=b[i,nplus2]
end LSQFIT;
```

MC02

Least Squares Linear Fit

```
procedure Linfit(N,x,y,a0,a1,xm,ym); value N; integer N;
real a0,a1,xm,ym; real array x,y;
comment   HUCC LIBRARY PROCEDURE MC02
          AUTHOR : J.N. BAXTER
```

Given the N+1 sample pairs $(x_0,y_0),(x_1,y_1)......(x_N,y_N)$ in arrays x,y[0:N] the procedure determines the coefficients a0 and a1 of the straight line approximation

y=a0+a1x

obtained from applying the least squares principle to the given data. This is simpler than using procedure LSQFIT(MC01) with n=1 for this purpose.

The procedure also evaluates the arithmetic means, sm and ym, of the contents of the arrays x and y respectively, for the reason that the straight line of best fit may be expressed conveniently as

(y-ym)=a1(x-xm)

Should this not be desired, and the values of xm and ym be of no interest, the procedure call can be such as

Linfit(N, independent variable, dependent variable, intercept, slope, slope, slope)

so that the values of xm and ym in turn are assigned to the variable slope and then overwritten by the value of a1;

```
procedure Linfit(N,x,y,a0,a1,xm,ym);  value N;  integer N;  real a0,a1,xm,ym;  real array x,y;
      begin real n,xi,yi,sx,sy,sxx,sxy;  integer i;
            n:=N+1.0;  sx:=sy:=sxx:=sxy:=0.0;
            for i:=0 step 1 until N do
                  begin xi:=x[i];  yi:=y[i];
                        sx:=sx+xi;  sy:=sy+yi;  sxx:=sxx+xi*xi;  sxy:=sxy+xi*yi
                  end;
            xm:=sx/n;  ym:=sy/n;
            a1:=(n*sxy-sx*sy)/(n*sxx-sx*sx);  a0:=(sy-a1*sx)/n
      end Linfit;
```

MC03

<u>Weighted Linear Least Squares Fit</u>

<u>procedure</u> wt linfit(N,xk,yk,k,wk,a0,a1,xm,ym); <u>value</u> N; <u>integer</u> N,k;

<u>real</u> xk,yk,wk,a0,a1,xm,ym;

<u>comment</u> HUCC LIBRARY PROCEDURE MC03

AUTHOR : J.N. BAXTER

In cases where a curve is to be fitted to sample pairs (x_i, y_i) by means of a transformation into $(f(x), f'(y))$ and a linear fit on the functions, it is necessary to weight the data so that the least squares approximation obtain is that for the original x and y. This is a Jensen procedure for the purpose, and k is the Jensen parameter which is utilised in the procedure call. For example, if $f(x) = {}^1/x$ and $f'(y) = \ell n\ y$ with a chosen weighting for each point of y*y the procedure call might be

wtlinfit(N,1.0/x[k],ℓn(y[k]),k,y[k]↑2)

to find:(cept,slope,meanfx,meanfy).

The procedure is also capable of an unweighted fit, if the formal parameter wk is replaced in the procedure call by 1.0, but an unweighted fit is more simply carried out by procedure Linfit(MC02);

MC03

```
procedure wtlinfit(N,xk,yk,k,wk,a0,a1,xm,ym);  value N;  integer N,k;  real xk,yk,wk,a0,a1,xm,ym;
    begin real w,x,yw,xw,sw,sxw,syw,sxxw,sxyw;
        sxw:=syw:=sxxw:=sxyw:=sw:=0.0;
        for k:=0 step 1 until N do
            begin x:=xk;  w:=wk;  yw:=yk*w;  xw:=x*w;
                sxw:=sxw+xw;  syw:=syw+yw;  sw:=sw+w;  sxxw:=sxxw+x*xw;  sxyw:=sxyw+yw*x
            end;
        xm:=sxw/sw;  ym:=syw/sw;
        a1:=(sw*sxyw-sxw*syw)/(sw*sxxw-sxw*sxw);  a0:= (syw-a1*sxw)/sw
    end wtlinfit;
```

Standard Deviation of Fitted Straight Line

```
real procedure Lindev(N,x,y,a0,a1); value N,a0,a1; integer N;
real a0,a1; real array x,y;
comment  HUCC LIBRARY PROCEDURE MC04 ,
         AUTHOR : J.N. BAXTER

         A procedure to evaluate the standard deviation of the N+1
         data points held in x,y[0:N] from the fitted straight line
         y=a0+a1x. (The values of a0 and a1 supplied to this procedure
         may be those calculated by Linfit(MC02) or may be other
         values, chosen at will, or derived from other work).
         To avoid the infinite value which would otherwise result
         from the use of this procedure with 2(N=1) data points
         the assignment Lindev:=5*10^75 occurs.  The procedure is
         not protected from use with N=0 which case fails on a
         sqrt error;
```

MC04

```
real procedure Lindev(N,x,y,a0,a1);  value N,a0,a1;  integer N;  real a ,a1;  real array x,y;
    begin integer i,n2;  real sum,dev;
        sum:=0.0;
        for i:= 0 step 1 until N do  begin dev:=a0+a1*x[i]-y[i];  sum:=sum+dev*dev  end;
        n2:=N-1;
        Lindev:= (if n2≠0 then sqrt(sum/n2) else 5.0⏨75)
    end Lindev;
```

LOCATE MINIMUM OF FUNCTION f(x)

```
real procedure MINX(a,b,eps,x,fx,fval);  value a,b,eps;
real a,b,eps,x,fx,fval;
comment  HUCC LIBRARY PROCEDURE MD01:
         AUTHOR   J. BOOTHROYD      :

         A procedure to locate the minimum of fx in the interval
         a ≤ x ≤ b by the method of trisection.  fx must be
         monotonic decreasing from x=a to the position of the
         minimum and thereafter monotonic increasing until x=b.
         eps is an absolute tolerance and the procedure aims to
         locate the minimum with an error less than eps.  Whether
         it succeeds will depend on how well the minimum is
         defined taking into account the specified eps and the
         precision to which  fx  may be evaluated for any given real
         number representation.  A relative tolerance delta may
         be specified using the call MINX(a,b,(b-a)*delta,x,fx,fval).
         On exit from the procedure fval yields the minimum value
         of the function;
```

MD01

```
real procedure MINX(a,b,eps,x,fx,fval);  value a,b,eps;  real a,b,eps,x,fx,fval;
      begin real sep,fx1,fx2;  integer d;    switch s:=L,movea;
        L:    sep:=(b-a)/3.0; x:=a+sep;  fx1:=fx; x:=b-sep;  fx2:=fx;
              if fx2=fx1 then begin d:=0; b:=b-sep; goto movea end;
              d:=1;
              if fx2>fx1 then b:=b-sep else movea: a:=a+sep;
              if sep≥eps then goto L;
              MINX:=x:= if d≠0 then a+sep else (a+b)/2.0; fval:= fx
end MINX;
```

MG 01

CONVERT SEXAGESIMAL ANGLES TO RADIANS

```
real procedure sexrad(deg,min,sec); value deg,min,sec;
integer deg,min; real sec;

comment   HUCC LIBRARY PROCEDURE MG 01:
          AUTHOR   J. BOOTHROYD:

          Converts angular measure in the sexagesimal (360,60,60) system
          to radians;
```

MG 02

CONVERT RADIAN ANGULAR MEASURE TO SEXAGESIMAL UNITS

```
procedure radtosex(rad,deg,min,sec); value rad;
integer deg,min; real rad,sec;

comment   HUCC LIBRARY PROCEDURE MG 02:
          AUTHOR   J. BOOTHROYD:

          Converts angular measure in radians to degrees, minutes and
          seconds in the sexagesimal system. The result is correctly
          rounded to two decimal places of seconds;
```

MG 03

CONVERT CENTESIMAL ANGLES TO RADIANS

```
real procedure centrad (deg,min,sec); value deg,min,sec;
integer deg,min; real sec;

comment   HUCC LIBRARY PROCEDURE MG 03:
          AUTHOR   J. BOOTHROYD:

          Converts angular measure from the centesimal (400,100,100) system
          to radians;
```

MG 04

CONVERT RADIAN ANGULAR MEASURE TO CENTESIMAL UNITS

<u>procedure</u> radtocent(rad,deg,min,sec); <u>value</u> rad;
<u>integer</u> deg, min; <u>real</u> rad,sec;

<u>comment</u> HUCC LIBRARY PROCEDURE MG 04:
AUTHOR J. BOOTHROYD:

Converts angular measure from radians to centesimal (400,100,100) units. The result is correctly rounded to two decimal places of seconds;

MG 05

COMPUTE DISTANCE AND GRID BEARING

<u>procedure</u> bearing(x,y,r,t); <u>value</u> x,y; <u>real</u> x,y,r,t;

<u>comment</u> HUCC LIBRARY PROCEDURE MG 05:
AUTHOR J. BOOTHROYD:

Computes r, the length, and t, the clockwise radian angular bearing from north of a line joining the origin (0,0) to the point (x,y). For the bearing and length of a line joining (x1,y1) to (x2,y2) use the call bearing (x2-x1,y2-y1,r,t);

MG 06

COMPUTE POLAR COORDINATES OF POINT X,Y

<u>procedure</u> polar (x,y,r,t); <u>value</u> x,y; <u>real</u> x,y,r,t;

<u>comment</u> HUCC LIBRARY PROCEDURE MG 06:
AUTHOR NATIONAL PHYSICAL LABORATORY:

Computes the polar coordinates r,t corresponding to given x,y Cartesian coordinates;

MG 07

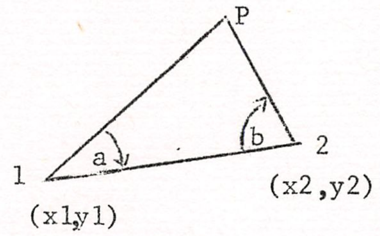
COMPUTE POINT OF INTERSECTION FROM ANGLES

```
procedure intang(x1,y1,a,x2,y2,b,x,y);   value x1,y1,a,x2,y2,b;
real x1,y1,a,x2,y2,b,x,y;
```

comment HUCC LIBRARY PROCEDURE MG 07:

AUTHOR J. BOOTHROYD:

Computes the coordinates of the point of intersection of the lines through (x1,y1),(x2,y2) which make angles, with respect to the line 12,of a and b radians respectively.



The following convention of signs must be observed.

Positive angles a are those swept out by closing the line 1P onto 12 clockwise.

Positive angles b are those swept out by closing the line 21 onto 2P clockwise.

The formulae used are:

$$x= \frac{((x2-x1)\cot(a)-(y2-y1))}{\cot(a)+\cot(b)} +x1$$

$$y= \frac{((y2-y1)\cot(a)+(x2-x1))}{\cot(a)+\cot(b)} +y1$$

```
real procedure sexrad(deg,min,sec);  value deg,min,sec;
integer deg,min;  real sec;
begin sec:= ((sec/60.0+min)/60.0+abs(deg))*0.0174532925;
      sexrad:=if deg<0 then -sec else sec
end sexrad;
```

MG 01

```
procedure radtosex(rad,deg,min,sec);  value rad;
integer deg,min;  real rad,sec;
begin real x;
      x:= abs(rad)/0.0174532925+0.00000139;
      deg:= entier(x);  x:= (x-deg)*60.0;
      if rad<0.0 then deg:=-deg;
      min:= entier(x);  sec:=(x-min)*60.0*100.0;
      sec:=entier(sec)/100.0
end radtosex;
```

MG 02

```
real procedure centrad(deg,min,sec);  value deg,min,sec;
integer deg,min;  real sec;
begin sec:=((sec/100.0+min)/100.0+abs(deg))*0.01570796327;
      centrad:= if deg<0 then -sec else sec
end centrad;
```

MG 03

```
procedure radtocent(rad,deg,min,sec);  value rad;
integer deg,min;  real rad,sec;
begin real x;
      x:= abs(rad)/0.01570796327+0.0000005;
      deg:= entier(x);  x:=(x-deg)*100.0;
      if rad<0.0 then deg:=-deg;
      min:= entier(x);  sec:=(x-min)*100.0*100.0;
      sec:=entier(sec)/100.0
end radtocent;
```

MG 04

MG 05

```
procedure bearing(x,y,r,t);  value x,y;  real x,y,r,t;
      begin real pi,piby2;  pi:=3.141592654;  piby2:=1.570796327;
            r:=sqrt(x*x+y*y);
            if r=0.0 then t:=0.0
                     else begin t:= if abs(x)<abs(y) then arctan(x/y)
                                                     else piby2-arctan(y/x);
                          y:=x+y;
                         if y<0.0 or x<0.0 then t:=t+pi;
                         if y>0.0 and x<0.0 then t:=t+pi
                     end
      end bearing;
```

MG 06

```
procedure polar(x,y,r,t);  value x,y;  real x,y,r,t;
      begin real pi,piby2;  pi:=3.141592654;  piby2:=1.570796327;
            r:=sqrt(x*x+y*y);
            if r=0.0 then t:=0.0
                     else begin t:= if abs(x)<abs(y) then piby2-arctan(x/y)
                                                     else arctan(y/x);
                          x:=x+y;
                         if x < 0.0 or y<0 then t:=t+pi;
                         if x > 0.0 and y<0 then t:=t+pi
                     end
      end polar;
```

MG 07

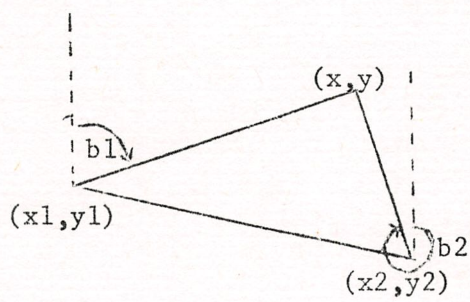
```
procedure intang(x1,y1,a,x2,y2,b,x,y); value x1,y1,a,x2,y2,b;
real x1,y1,x2,y2,a,b,x,y;
begin a:=1.0/tan(a);  b:=a+1.0/tan(b);
      x2:=x2-x1;  y2:=y2-y1;
      x:=(x2*a-y2)/b+x1;
      y:=(y2*a+x2)/b+y1
end intang;
```

COMPUTE POINT OF INTERSECTION FROM BEARINGS

```
procedure intbrg (x1,y1,b1,x2,y2,b2,x,y);   value x1,y1,b1,x2,y2,b2;
real x1,y1,b1,x2,y2,b2,x,y;

comment   HUCC LIBRARY PROCEDURE MG 08:
          AUTHOR   J. BOOTHROYD:
```

Computes the point of intersection of the line,bearing b1, through (x1,y1) with the line, bearing b2, through (x2,y2).



This procedure uses MG 05, procedure bearing, to compute the bearing of the line joining (x1,y1) to (x2,y2). The base angles of the triangle are then computed and the point of intersection evaluated by the method of MG 07;

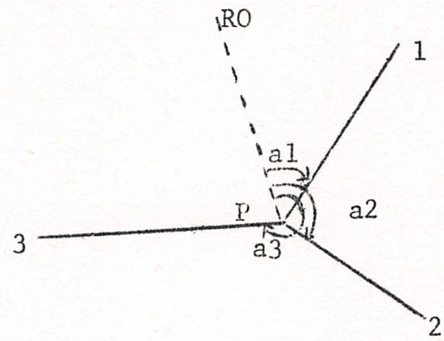
MG 09

THREE POINT RESECTION BY ANGLES

procedure resect(x1,y1,a1,x2,y2,a2,x3,y3,a3,x,y,scale,fail);
value x1,y1,a1,x2,y2,a2,x3,y3,a3,scale;
real x1,y1,a1,x2,y2,a2,x3,y3,a3,x,y,scale; **label** fail;

comment HUCC LIBRARY PROCEDURE MG 09:
AUTHOR J. BOOTHROYD:

Computes the coordinates x,y of the point of intersection P of the lines P1,P2,P3 from the angles a1,a2,a3 (in radians) where a1,a2,a3 are measured clockwise from a reference observation line to the lines P1, P2 and P3 respectively.



The RO line may coincide with the line P1, i.e. a1=0.
The points may be in any order with respect either to themselves or to the resected point P. The procedure rejects all geometrically impossible configurations by an exit to the label fail, and special steps are taken to preserve accuracy in the numerically difficult configurations.
The procedure is based on the formulae:-

x-x1=tan b1(y-y1)
x-x2=tan b2(y-y2)
x-x3=tan b3(y-y3)
b2=b1+(a2-a1), b3=b1+(a3-a1)

where b1,b2,b3 are the grid bearings of the lines P1,P2,P3.
From these may be derived the formula

$$\tan(b1)=\frac{((x2-x1)\cot(a2-a1)-(x3-x1)\cot(a3-a1)+(y3-y1))}{((y2-y1)\cot(a2-a1)-(y3-y1)\cot(a3-a1)-(x3-x1))}$$

As is well known to surveyors, the case in which P,1,2,3 together form a cyclic quadrilateral is indeterminate. Mathematically this case yields tan(b1)=0/0. Near cyclic cases also produce large inaccuracies and are detected by evaluating the numerator and denominator of the expression for tan(b1) separately. The parameter scale has been provided for this case. The value given to scale should be twice the distance from the approximate centre of the known points to the most distant known point. The procedure computes 5% of this value and if <u>both</u> numerator and denominator are less than 0.05*scale a cyclic case is assumed;

```
procedure resect(x1,y1,a1,x2,y2,a2,x3,y3,a3,x,y,scale,fail);
value x1,y1,a1,x2,y2,a2,x3,y3,a3,scale;
real x1,y1,a1,x2,y2,a2,x3,y3,a3,x,y,scale;  label fail;
begin real t,r,eps;  boolean a2zero,a3zero,absa2,absa3;  switch s:=L2,L3,done;
      procedure solve(x1,y1,x2,y2,a1,a2);
      value x1,y1,x2,y2,a1,a2;  real x1,y1,x2,y2,a1,a2;
      begin y2:=(y2*a2-x2)/(a2-a1);
            x:=y2*a1+x1;  y:=y2+y1
      end solve;
      eps:=0.05*scale;
      x3:=x3-x2;  y3:=y3-y2;  a3:=a3-a1;
      x2:=x2-x1;  y2:=y2-y1;  a2:=a2-a1;
      a3zero:=a3=0.0;  a2zero:=a2=0.0;
      if a2zero and a3zero then goto fail;
      if a3zero then
      begin a2:=1.0/tan(a2);  if abs(a2)>₁₀5 then goto fail;
            t:=y3+y2;
            if t=0.0 then begin y:=0.0;  x:=a2*y2+x2+x1;  goto done end
                  else begin a1:=(x3+x2)/(y3+y2);  goto L2 end
      end;
      if a2zero then
      begin a3:=1.0/tan(a3);  if abs(a3)>₁₀5 then goto fail;
            if y2=0.0 then begin y:=0.0;  x:=a3*y3+x3+x2+x1;  goto done end
                   else begin a1:=x2/y2;  goto L3 end
      end;
       a2:=1.0/tan(a2);  a3:=1.0/tan(a3);  absa2:=abs(a2)>₁₀5;  absa3:=abs(a3)>₁₀5;
      if absa2 and absa3 then goto fail;
      if absa3 then
      begin t:=y3+y2;
            if t=0.0 then begin y:=0.0;  x:=a2*y2+x2+x1;  goto done end
                  else begin a1:=(x3+x2)/(y3+y2);  goto L2 end
      end;
      if absa2 then
      begin if y2=0.0 then begin y:=0.0;  x:=a3*y3+x3+x2+x1;  goto done end
                   else begin a1:=x2/y2;  goto L3 end
      end;
      t:=(x2*a2-(x3+x2)*a3+y3);  r:=(y2*a2-(y3+y2)*a3-x3);
      if abs(t)<eps and abs(r)<eps then goto fail;
      a1:=t/r;
L2:   a2:=(1.0+a2*a1)/(a2-a1);  solve(x1,y1,x2,y2,a1,a2);  goto done;
L3:   a3:=(1.0+a3*a1)/(a3-a1);  solve(x1,y1,x3+x2,y3+y2,a1,a3);
done: end resect;
```

MULTIPLE POINT RESECTION FROM ANGLES

procedure avresect(obs,n,scale,x,y); **value** n,scale;
real scale,x,y; **integer** n; **array** obs;

comment HUCC LIBRARY PROCEDURE MG 10:
AUTHOR J. BOOTHROYD:

The minimum number of known points necessary for the evaluation of the coordinates of a resected point is 3. Where the number of known points exceeds 3 some method of obtaining the best estimate of x,y from all computed values of x,y is required. This procedure computes the weighted mean of all calculations based on the selection of all possible combinations of 3 points selected from n points, using the procedure resect (MG 09) ${}_nC_3$ times. The array obs[1:n,1:4] contains the observational data for the n observations. For the r^{th} observation x_r y_r a_r and weight$_r$ should be provided respectively in obs[r,1], obs[r,2], obs[r,3] and obs[r,4]. For a resection based on points i,j,k with respective weights wti,wtj,wtk a group weight wti+wtj+wtk is assumed. The parameter scale is used to reject near cyclic cases for details of which see the description of MG 09. For angular errors of the order of 1 second the procedure accuracy over eight (difficult) points is of the order of .0001 unit with a scale of 100 units. This means an accuracy of .1 yd for resections over distances of 100,000 yds. The errors are approximately linear with angular error and will increase if the number of observations decreases;

```
procedure intbrg(x1,y1,b1,x2,y2,b2,x,y);  value x1,y1,b1,x2,y2,b2;        MG 08
real x1,y1,b1,x2,y2,b2,x,y;
begin      real r,t;
           x2:=x2-x1;  y2:=y2-y1;
           bearing(x2,y2,r,t);
           b1:=1.0/tan(t-b1);
           t:=b1+1.0/tan(3.141592654+b2-t);
           x:=(x2*b1-y2)/t+x1;
           y:=(y2*b1+x2)/t+y1
end intbrg;


procedure avresect(obs,n,scale,x,y);  value n,scale;                     MG10
real scale,x,y;  integer n;  array obs;
begin integer i,j,k;
      real sn,sx,sy,x1,y1,a1,x2,y2,a2,xx,yy,wti,wtj,wt;
      switch s:=loop;
      sn:=sx:=sy:=0.0;
      for i:=1 step 1 until n do
      begin x1:=obs[i,1];  y1:=obs[i,2];  a1:=obs[i,3];  wti:=obs[i,4];
            for j:=i+1 step 1 until n do
            begin x2:=obs[j,1];  y2:=obs[j,2];  a2:=obs[j,3];  wtj:=obs[j,4];
                  for k:=j+1 step 1 until n do
                  begin resect(x1,y1,a1,x2,y2,a2,obs[k,1],obs[k,2],obs[k,3],xx,yy,scale,loop);

                        wt:=wti+wtj+obs[k,4];
                        sn:=sn+wt;
                        sx:=sx+wt*xx;
                        sy:=sy+wt*yy;
            loop: end k
            end j
      end i;
      if sn‡0 then begin x:=sx/sn;  y:=sy/sn  end
            else print punch(3),££l?RESECT ERROR?,wait
end avresect;
```

THREE POINT RESECTION FROM DISTANCES

```
procedure distresect(x1,y1,d1,x2,y2,d2,x3,y3,d3,x,y,fail);
value x1,y1,d1,x2,y2,d2,x3,y3,d3;
real x1,y1,d1,x2,y2,d2,x3,y3,d3,x,y;  label fail;

comment    HUCC LIBRARY PROCEDURE MG 11:
           AUTHOR   J. BOOTHROYD:
```

Computes the coordinates of a resected point P from the distances d1,d2,d3 from P to known points (x1,y1),(x2,y2),(x3,y3). This procedure has been written to provide assistance to those making use of telurometery.

The equations are

$$(x-x1)^2+(y-y1)^2 = d1^2$$
$$(x-x2)^2+(y-y2)^2 = d2^2$$
$$(x-x3)^2+(y-y3)^2 = d3^2$$

from which two linear equations in x and y may be obtained by subtraction in pairs. If the data provided results in a zero determinant of the two linear equations the procedure exits to the label fail;

MULTIPLE POINT RESECTION USING DISTANCES

procedure avdistresect(obs,n,x,y); **value** n;
integer n; **real** x,y; **array** obs;

comment HUCC LIBRARY PROCEDURE MG 12:
AUTHOR J. BOOTHROYD:

This procedure evaluates the coordinates x,y of a resected point as the weighted mean of all computations based on the use of all combinations of 3 points selected from n observations, using distances in all cases. The procedure disresect (MG 11) is called $^{n}C_{3}$ times. For the r^{th} observation the data $x_r, y_r, d_r, weight_r$ should occupy obs[r,1], obs[r,2], obs[r,3], obs[r,4] respectively of the array obs[1:n,1:4].
For a computation using observations i,j,k with respective weights wti,wtj,wtk, a group weight wti+wtj+wtk is assumed;

```
procedure distresect(x1,y1,d1,x2,y2,d2,x3,y3,d3,x,y,fail);
value x1,y1,d1,x2,y2,d2,x3,y3,d3;
real x1,y1,d1,x2,y2,d2,x3,y3,d3,x,y;  label fail;
begin real a11,a12,a21,a22,b1,b2,det;
      a11:=(x2-x1);  a12:=(y2-y1);
      a21:=(x3-x1);  a22:=(y3-y1);  d1:=d1*d1;
      b1:=(a11*(x2+x1)+a12*(y2+y1)-d2*d2+d1)/2.0;
      b2:=(a21*(x3+x1)+a22*(y3+y1)-d3*d3+d1)/2.0;

      det:=a11*a22-a21*a12;
      if det=0.0 then goto fail;
      x:=(a22*b1-a21*b2)/det;
      y:=(a11*b2-a21*b1)/det
end distresect;
```

MG 11

```
procedure avdistresect(obs,n,x,y);  value n;
real x,y;  integer n;  array obs;
begin integer i,j,k;
      real sn,sx,sy,x1,y1,d1,x2,y2,d2,xx,yy,wti,wtj,wt;
      switch s:=loop;
      sn:=sx:=sy:=0.0;
      for i:=1 step 1 until n do
      begin x1:=obs[i,1];  y1:=obs[i,2];  d1:=obs[i,3];  wti:=obs[i,4];
            for j:=i+1 step 1 until n do
            begin x2:=obs[j,1];  y2:=obs[j,2];  d2:=obs[j,3];  wtj:=obs[j,4];
                  for k:=j+1 step 1 until n do
                  begin distresect(x1,y1,d1,x2,y2,d2,obs[k,1],obs[k,2],obs[k,3],xx,yy,loop);
                        wt:=wti+wtj+obs[k,4];
                        sn:=sn+wt;
                        sx:=sx+wt*xx;
                        sy:=sy+wt*yy;
            loop: end k
            end j
      end i;
      if sn≠0 then begin x:=sx/sn;  y:=sy/sn  end
            else print punch(3),££l?DISTRESECT ERROR?,wait
end avdistresect;
```

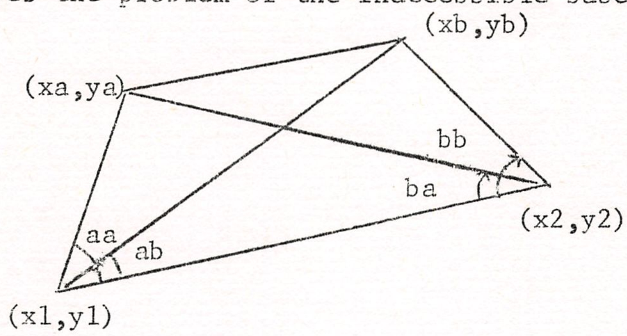
MG 12

THE INACCESSIBLE BASE PROBLEM

```
procedure farbase(xa,ya,xb,yb,aa,ab,ba,bb,x1,y1,x2,y2);
value xa,ya,xb,yb,aa,ab,ba,bb;
real xa,ya,xb,yb,aa,ab,ba,bb,x1,y1,x2,y2;
```

comment HUCC LIBRARY PROCEDURE MG 13:
AUTHOR J. BOOTHROYD:

Solves the problem of the inaccessible base.



(x1,y1) and (x2,y2) are stations whose coordinates are required. (xa,ya),(xb,yb) are stations whose coordinates are known. aa,ab,ba,bb are angular observations from (x1,y1) and(x2,y2) on (xa,ya) and (xb,yb) using the angular sign convention of procedure MG 07. Standard survey computational methods would compute the tangent of the angle of inclination of line a,b to line 1,2 and, from the computed bearing of 1,2 and the measured angles deduce the bearings of all other lines. This would provide sufficient information to evaluate x1,y1,x2,y2 using procedure MG 08. This is not the best method if automatic computing facilities are available. Using the formulae described in procedure MG 07 we obtain the four equations :-

$$xa = ((x2-x1)\cot aa-(y2-y1))/(\cot aa + \cot ba)+x1$$

$$xb = ((x2-21)\cot ab-(y2-y1))/(\cot ab+\cot bb)+x1$$

$$ya = ((y2-y1)\cot aa+(x2-x1))/(\cot aa+\cot ba)+y1$$

$$yb = ((y2-y1)\cot ab+(x2-21))/(\cot ab+\cot bb)+y1$$

which, rearranged yields four linear equations in the four unknowns x1,y1,x2,y2. This procedure computes the coefficients of the linear system and calls on LE 02 (SOLVEQ) to obtain the solution;

```
procedure farbase(xa,ya,xb,yb,aa,ab,ba,bb,x1,y1,x2,y2);
value xa,ya,xb,yb,aa,ab,ba,bb;
real xa,ya,xb,yb,aa,ab,ba,bb,x1,y1,x2,y2;
begin real aaba,abbb;  array A[1:4,1:5];
      aa:=1.0/tan(aa);  ab:=1.0/tan(ab);
      ba:=1.0/tan(ba);  bb:=1.0/tan(bb);
      aaba:=aa+ba;  abbb:=ab+bb;
      A[1,1]:=A[3,2]:=ba;  A[2,1]:=A[4,2]:=bb;
      A[1,3]:=A[3,4]:=aa;  A[2,3]:=A[4,4]:=ab;
      A[1,2]:=A[2,2]:=A[3,3]:=A[4,3]:=1.0;
      A[1,4]:=A[2,4]:=A[3,1]:=A[4,1]:=-1.0;
      A[1,5]:=xa*aaba;  A[2,5]:=xb*abbb;
      A[3,5]:=ya*aaba;  A[4,5]:=yb*abbb;
      SOLVEQ(A,4);
      x1:=A[1,5];  y1:=A[2,5];  x2:=A[3,5];  y2:=A[4,5]
end farbase;
```

MG14

3-DIMENSIONAL COORDINATE GEOMETRY AND VECTOR ARITHMETIC PACKAGE

The procedures of this package are collectively designated as MG14. No provision is made for copying individual procedures into user's tapes since these are brief enough to be re-written as required.

comment HUCC LIBRARY PROCEDURE MG14:

AUTHOR : B. ROBINSON:

The point P(x,y,z) in three dimensional space has one-to-one correspondence with the vector $\vec{V}$ which has components x,y,z parallel to the three mutually perpendicular coordinate axes respectively. In these procedures the vector $\vec{V}$ (which may be visualised as the directed line-segment OP from the origin to the point P) is represented by an ALGOL array of three elements, which is declared thus

real array V[1:3];

The elements of this array are the components of the vector,

V[1]=x

V[2]=y

V[3]=z

In procedure calls the vector is referred to as V[j]. The subscript j must be declared as an **integer** and the declaration must be valid for all blocks in which these procedures are employed. Inside the procedures j is assigned the values 1,2, or 3 as needed for operations on the components of the vector. Outside the procedures j should not be used. The procedures make extensive use of the ALGOL facility of calling by name. This is convenient because it makes possible the use of linear expressions of vectors as actual parameters in procedure calls, and the use of arrays of vectors or individual vectors. For example, a two-dimensional lattice of vectors with typical element L_{mn} would be declared as

real array L[ℓ1:u1, ℓ2:u2, 1:3]

and any individual vector of this lattice could be referred to in a procedure call as L[m,n,j].

The result of this is that the text of the program will be conveniently reminiscent of vector algebra.

VECTOR ASSIGNMENT:

<u>procedure</u> make(yj) equal to:(xj); <u>real</u> xj,yj;

Examples :	make(V2[j],V1[j]);	effect is	"V2:=V1"
	make(V3[j],V1[j] + V2[j]);		"V3:=V1+V2"
	make(V2[j],V1[j]*sc);		"V2:=V1*sc"
	make(M[j],(A[j]+B[j])/2.0);		

This finds the midpoint of the line-segment AB

effect is "M:=(A+B)/2"

make(G[j],(A[j]+B[j]+C[j])/3.0);

This finds the centroid of the triangle ABC,

effect is "G:=(A+B+C)/3"

MULTIPLE VECTOR ASSIGNMENT:

<u>procedure</u> make both(yj) and:(zj)equal to:(xj); <u>real</u> xj,yj,zj;

Example: makeboth(V[6,j],temp[j],V2[j]-V1[j];

effect is "V[6]:=temp:=V2-V1"

DOT PRODUCT OF TWO VECTORS:

<u>real</u> <u>procedure</u> dot(xj,yj); <u>real</u> xj,yj;

The procedure takes the value of the scalar product of the two vectors.

Example: r:=dot(V1[j],V2[j]);

If the dot product is zero, then the two vectors are perpendicular or else one is a zero vector.

LENGTH OF A VECTOR:

real procedure length(xj); **real** xj;

NOTE: uses **procedure** dot.

Examples : The length of the vector V from the origin is found by

r:=length(V[j]);

The distance between the ends of the vectors V_1 and V_2 is found by r:=length(V2[j]-V1[j]);

COSINE OF THE ANGLE BETWEEN TWO VECTORS:

real procedure cosang(xj,yj) escape label : (fail);
real xj,yj; **label** fail;

The procedure takes the value of the cosine of the angle subtended at the origin by the two vectors. If either vector is of zero length then the angle is indeterminate and the procedure exits to a label.

Example: cstheta:=cosang(V1[j],V2[j],fail);

To find the cosine of an angle not at the origin the call by name is so arranged as to simulate a change of origin to the apex of the angle. Thus where the angle $\angle ABC$ is represented by the three vectors a[j], b[j] and c[j],

cstheta:=cosang(a[j]-b[j], c[j]-b[j], fail);

NOTE: uses procedures dot and length.

VECTOR OF UNIT LENGTH PARALLEL TO A GIVEN VECTOR:

procedure unit(nj)parallel to:(xj)escape label: (fail);
real nj,xj; **label** fail;

Example: unit(V[j],V[j],fail); converts the vector V into a unit vector in the same direction. The procedure will exit by the label if the vector given to it has zero length.

NOTE: uses procedures dot and length

CROSS PRODUCT OF TWO VECTORS:

<u>procedure</u> cross(xj,yj) is written into:(zj); <u>real</u> xj,yj,zj;

Example: cross(V1[j], V2[j], V3[j]);   effect is "V3:=V1xV2"

The direction of V3 is perpendicular to the plane in which V1 and V2 lie. If V1 and V2 are parallel vectors then V3 will be a zero vector. If V1 lies along the x-axis and V2 lies along the y-axis, then V3 will lie along the z-axis in accordance with the convention, this may be expressed by the statement that right-handed coordinates are used (remember that V1xV2=-V2xV1).

VECTOR OF UNIT LENGTH PERPENDICULAR TO TWO GIVEN VECTORS:

<u>procedure</u> perpunit(nj) perpendicular to:(xj,yj) escape label:(fail);
<u>real</u> nj,xj,yj; <u>label</u> fail;

Example: perpunit(n[j],V1[j],V2[j]);

NOTE: uses procedures dot, length, unit and cross.

BOX PRODUCT OF THREE VECTORS:

<u>real</u> <u>procedure</u> box(xj,yj,zj); <u>real</u> xj,yj,zj;

The procedure takes the value of the scalar triple product of the three vectors. If the three vectors are coplanar with the origin then the product is zero. It can therefore be used to test whether any four points are coplanar, by evaluation of

q:=box(a[j]-b[j], c[j]-b[j], d[j]-b[j]);

NOTE: uses procedures dot and cross.

MG14

```
comment vector procedures;

    procedure make(yj,xj); real xj,yj; for j:=1,2,3 do yj:=xj;

    procedure makeboth(yj,zj,xj); real xj,yj,zj; for j:=1,2,3 do yj:=zj:=xj;

    real procedure dot(xj,yj); real xj,yj;
        begin real t; t:=0.0; for j:=1,2,3 do t:=t+xj*yj; dot:=t
        end dot product;

    real procedure length(xj); real xj; length:=sqrt(dot(xj,xj));

    real procedure cosang(xj,yj,fail); real xj,yj; label fail;
        begin real d; d:=length(xj)*length(yj); if d=0.0 then goto fail;
            cosang:= dot(xj,yj)/d
        end cos angle;

    procedure unit(nj,xj,fail); real nj,xj; label fail;
        begin real d; d:=length(xj); if d=0.0 then goto fail;
            for j:=1,2,3 do nj:=xj/d
        end unit;

    procedure cross(xj,yj,zj); real xj,yj,zj;
        begin real x1,x2,x3,y1,y2,y3;
            j:=1; x1:=xj; y1:=yj;
            j:=2; x2:=xj; y2:=yj;
            j:=3; x3:=xj; y3:=yj;
            zj:=x1*y2-x2*y1; j:=2; zj:=x3*y1-x1*y3;
            j:=1; zj:=x2*y3-x3*y2
        end cross product;

    procedure perpunit(nj,xj,yj,fail); real nj,xj,yj; label fail;
        begin array a[1:3]; cross(xj,yj,a[j]); unit(nj,a[j],fail) end;

    real procedure box(xj,yj,zj); real xj,yj,zj;
        begin array a[1:3]; cross(xj,yj,a[j]); box:= dot(a[j],zj)
        end box product;
```

COMPUTE FOURIER COEFFICIENTS

```
procedure fourier(f,a,b,n); value n; integer n; real array f,a,b;
comment   HUCC LIBRARY PROCEDURE MH01:
          AUTHOR:  J. BOOTHROYD

          Evaluates the coefficients of the Fourier expansion of
          a function f(x) specified at the  n  sample points
          f[0],f[1], ...., f[n-1].  n  may be odd or even and
          on exit from the procedure the arrays a,b[0:n div 2]
          will contain the coefficients appropriate to either case
          as follows:-
          n even (n = 2N)

                         N-1
          f(x)=a[0] + Σ(a[k]*cos(kπx/N)+b[k]*sin[kπx/N))+a[N]cosπx
                         k=1

               with b[0] = b[N] = 0.
          n odd (n = 2N+1)

                          N
          f(x)=a[0] + Σ(a[k]*cos(2πkx/n)+b[k]*sin(2πkx/n)),
                         k=1

               with b[0] = 0.
          The algorithm uses the recurrence relations described
          in R. W. Hamming - Numerical Methods for Scientists and
          Engineers, page 72.
```

MH01

```
procedure fourier(f,a,b,n); value n; integer n; real array f,a,b;
begin integer ndiv2,k,m; real c,s,vk,v1,u0,u1,um,t,ck,nby2;
      ndiv2:= n div 2;  nby2:= n/2;  t:= 6.2831853/n;
      c:= cos(t); s:= sin(t);  vk:= 0.0;  v1:= -1;
      for k:= 0 step 1 until ndiv2 do
            begin t:= c*vk; ck:= t-v1;  v1:= vk;  vk:= ck+t;
                  t:= ck+ck;  u1:=0.0; um:= f[n-1];
                  for m:= n-2 step -1 until 1 do
                        begin u0:= u1; u1:= um; um:= t*u1-u0+f[m] end;
                  a[k]:= (ck*um-u1+f[0])/nby2;
                  b[k]:= s*v1*um/nby2
            end;
      a[0]:= a[0]/2.0;  if ndiv2*2=n then
            begin a[ndiv2]:= a[ndiv2]/2.0;
                  b[ndiv2]:=0.0
            end
end fourier;
```

SUM FOURIER SERIES

```
real procedure sumfourier(a,b,n,x);  value n,x;  real x;
     integer n;  real array a,b;
comment  HUCC LIBRARY PROCEDURE MH02:
         AUTHOR:  J. BOOTHROYD

         Evaluates the function f(x) for given argument x
         by summing the Fourier series of the function, the
         coefficients of which occupy arrays a,b[0:n div 2]
         as a result of using procedure MH01.
```

```
real procedure sumfourier(a,b,n,x);  value n,x;  real x;  integer n;  real array a,b;
comment evaluates the function f(x) for given argument x by summing the Fourier series whose
coefficients are the elements of a,b[0:ndiv2] derived by the use of procedure fourier, i.e.
n even (n=2N)
f(x)= a[0]+sigma(a[k]*cos(pi*k*x/N)+b[k]*sin(pi*k*x/N),k,1,N-1) + a[N]*cos(pi*x), b[0]=b[N]=0
n odd (n=2N+1)
f(x)=a[0]+sigma(a[k]*cos(2*pi*k*x/n)+b[k]*sin(2*pi*k*x/n),k,1,N),   b[0] = 0;
begin real vk,v1,sum,c,s,ck;  integer k,ndiv2;
      ndiv2:=n div 2;  x:=6.2831853*x/n;  c:=cos(x);  s:=sin(x);
      sum:=a[0];  vk:=1.0;  v1:=0.0;
      for k:=1 step 1 until ndiv2 do
            begin x:=c*vk;  ck:=x-v1;
                  v1:=vk;  vk:=ck+x;
                  sum:=sum+a[k]*ck+b[k]*s*v1
            end;
      sumfourier:=sum
end sumfourier;
```

LOCATE ROOT OF f(x)=0

real **procedure** bisec(f,x,a,b,eps,error); **value** a,b,eps;

real f,x,a,b,eps; **label** error;

comment HUCC LIBRARY PROCEDURE MR01:

AUTHOR: NOT KNOWN

A procedure to find a root of f(x)=0 in the interval a to b by continued bisection. If f(a) and f(b) have the same sign the procedure exits to the label error. The error escape facility may be used to determine the interval (a,b) within which some root of interest lies. Suppose that f(x) has several positive roots x_1 x_2 x_3 .... and it is known that the root seperation $(x_2 - x_1)$, $(x_3 - x_2)$ etc. is at least z but the location of x_1 is in some doubt. The routine

```
        xa := -z;
strobe: xa := xa + z:
        y := bisec(f(x),x,xa,xa+z,10⁻⁴,strobe)
```

will increment xa until f(xa) and f(xa+z) have opposite signs when the bisec routine will evaluate the required root to an accuracy of 10^{-4}.

MR01

```
real procedure bisec(f,x,a,b,eps,error);  value a,b,eps;  real f,x,a,b,eps;  label error;
comment the procedure finds a root of f(x)=0 in the interval a to b by
continued bisection.  If f(a) and f(b) have the same sign the procedure exits
through the label error;
      begin real q;  x:=a;  q:=f;  x:=b;
            if f*q>0 then goto error;
            q:=(b-a)*sign(q)/2.0;  x:=(a+b)/2.0;  eps:=eps/2.0;
            for q:=q/2.0 while abs(q)>eps do x:=x+(if f>0 then q else -q);
            bisec:=x
      end bisec;
```

MS01

GENERAL SUM SERIES

real procedure sigma(tk,k,a,b); value a,b; integer k,a,b; real tk;

comment HUCC LIBRARY PROCEDURE MS01:

AUTHOR: J. BOOTHROYD

A Jensen procedure to evaluate the sum over k from a to b of tk with positive unit increments in k. The parameters tk and k are called by name. As examples of the flexibility of this procedure the calls,

1. sigma(a+i*d,i,0,3)
2. sigma(a*r↑i,i,0,n-1)
3. sigma(a[i]*b[i],i,1,N)

will evaluate respectively

1. The sum of 4 terms of an arithmetic progression with first term a and common difference d.
2. The sum of n terms of a geometric progression.
3. The inner product of the vectors a,b[1:N].

These examples illustrate the dependence of the actual first parameter on the variable actually used as the second parameter.

MS01

```
real procedure sigma(tk,k,a,b); value a,b;  real tk;  integer k,a,b;
comment the sum over k from a to b of tk, with positive unit increments in k;
      begin real  sum;
            sum:=0.0;
            for k:=a step 1 until b do sum:=sum+tk;
            sigma:=sum
      end sigma;
```

20 ROMBERG INTEGRATION

```
real procedure romberg1(x,fx,a,b,order,eps);  value a,b,eps,order;
                real x,fx,a,b,eps;  integer order;
begin integer i,j,mark;  real ti,k,h,n,m,t,fac,del,abstm,abstol,absti;
      array aa[1:order];  switch s:=test,newi,done;
      x:=a;  ti:=fx;  x:=b;  ti:=aa[1]:=(ti+fx)/2.0;
      k:=h:=b-a;
      if k=0.0 then begin i:=0;  goto done end;
      n:=1.0;  abstol:=₁₀-8/abs(k);
      for i:=2 step 1 until order do
      begin h:=h/2.0;  m:=0.0;
            for x:=a+h step k until b do :=m+fx;
            m:=m/n;  t:=ti;  j:=i;
            absti:=abs(ti);
  test:     abstm:=abs(m-t);
            if ti≠0.0 then
            begin if abstm/absti≤eps then goto done end
            else
            begin if abstm≤abstol then goto done end;
            if i ≠ j then
            begin if mark=0 then
                  begin t:=aa[j]:=m+(m-t)/fac;
                        mark:=1;  del:=3.0*(fac+1.0);
                        fac:=fac+del
                  end
                  else
                  begin j:=j-1;  if j=0 then goto newi;
                        m:=t;  t:=aa[j];  mark:=0
                  end;
            end
            else
            begin mark:=0;  fac:=3.0;  del:=0.0;
                  ti:=m:=aa[j]:=(m+t)/2.0;  j:=j-1
            end;
            goto test;
   newi:    k:=h;  n:=n+n
      end i;
      i:=0;
done: romberg:=(b-a)*(if i>0 then (t+m)/2.0 else aa[1])
end romberg;
```

MS02

ADAPTIVE SIMPSON INTEGRATION

real **procedure** simps (f, x, a, b, eps); **value** a, b, eps; **real** f, x, a, b, eps;

comment HUCC LIBRARY PROCEDURE MS02 :

AUTHOR J. BOOTHROYD :

A modification of Algorithm 233 COMM. A.C.M. VOL 7 NO. 6 JUNE 1964 p 348. The procedure evaluates the integral of f with respect to x from lower bound a to upper bound b.

$$simps := \int_a^b f \, dx$$

The parameter eps determines the acceptable relative error between successive evaluations of the integral I. If I_r and I_{r-1} are successive approximations, the process will terminate when

$$abs((I_r - I_{r-1}) / I_r) < eps$$

unless $I_r = 0$ in which case, to avoid division by zero, the process terminates when the absolute difference between successive values of I differ by less than 10^{-8}. As an example of the use of simps, the call

y : = simps (1.0/x, x, 1.0, B, .0001)

will assign to y the value $\int_1^B (1/x) \, dx = \ln [B]$

with a tolerance of .0001.

The procedure operates correctly for the cases a < b, a = b and a > b.

```
real procedure simps(f,x,a,b,eps);  value a,b,eps;  real f,x,a,b,eps;
      begin real Z1,Z2,Z3,h,k;  switch s:=again,done;
            if a=b then begin h:=0.0;  goto done  end;
            x:=a;  Z1:=f;  x:=b;  Z1:=Z1+f;  k:=(b-a)/2.0;
            x:=a+k;  Z2:=f;  Z3:=Z1+4.0*Z2;  Z1:=Z1+2.0*Z2;
again:      Z2:=0;  h:=k/2;
            for x:=a+h step k until b do Z2:=Z2+f;
            Z1:=Z1+4.0*Z2;
            if Z1≠0 then begin if abs((Z1-2.0*Z3)/Z1)<eps then goto done end
                    else if abs(Z1-2.0*Z3)*h/3.0<₁₀-8 then goto done ;
            Z3:=Z1;  Z1:=Z1-2.0*Z2;  k:=h;  goto again;
done:       simps:=h*Z1/3.0
      end simps;
```

MS 03

EVALUATE DEFINITE INTEGRAL

real procedure havie (x,a,b,eps,integrand,m,mask);

value a,b,eps,m,mask; integer m; real a,b,eps,integrand,x,mask;

comment HUCC LIBRARY PROCEDURE MS 03

Author : ACM 257

Performs numerical integration of integrand with respect to x over the interval a to b

$$\text{havie} := \int_a^b \text{integrand}\, dx$$

eps is the convergence criterion, m is the maximum order of approximation to be considered in attempting to satisfy the eps criterion and mask is a value supplied by the user which will be returned by havie if convergence is not obtained. The method is basically the trapezium formula with higher order corrections (m specifies an upper limit to the number of these performed).

Examples of calls are :-

(a) To evaluate $y = \int_0^{\pi/2} \cos x\, dx$

y:= havie (x,0.0,1.5707963,10^{-6},cos(x),10,9.99999);

(b) To evaluate $z = \int_0^{4.3} e^{-y^2} dy$

z:= havie(y,0.0,4.3,10^{-6},exp(y*y),10,9.99999)

```
          real procedure havie(x,a,b,eps,integrand,m,mask);  value a,b,eps,m,mask;  integer m;   real a,b,x,eps,integrand,mask;
              begin real h,endpts,sumt,sumu,d;  integer i,j,k,n;  switch s:=estimate,test,exit;
                  real array t,u,tprev,uprev[1:m];
                  x:=a;  endpts:=integrand;  x:=b;  endpts:=0.5*(integrand+endpts);
                  sumt:=0.0;  i:=n:=1;  h:=b-a;
estimate:         t[1]:=h*(endpts+sumt);  sumu:=0.0;  x:=a-h/2.0;
                  for j:=1 step 1 until n do begin x:=x+h;  sumu:=sumu+integrand  end;
                  u[1]:=h*sumu;  k:=1;
test:             if abs(t[k]-u[k]) ≤ eps then begin havie:=0.5*(t[k]+u[k]);  goto exit  end;
                  if k ≠ i then
                      begin d:=2.0↑(k+k);  t[k+1]:=(d*t[k]-tprev[k])/(d-1.0);  tprev[k]:=t[k];
                          u[k+1]:=(d*u[k]-uprev[k])/(d-1.0);  uprev[k]:=u[k];  k:=k+1;
                          if k=m then begin havie:=mask;  goto exit  end;  goto test
                      end;
                  h:=h/2.0;  sumt:=sumt+sumu;  tprev[k]:=t[k];  uprev[k]:=u[k];
                  i:=i+1;  n:=n+n;  goto estimate;
exit:         end havie;
```

LAGRANGE INTERPOLATION

```
real procedure Lagrange(x,y,arg,n,m);  value arg,m,n;
    array x,y;  real arg;  integer m,n;
    comment   HUCC LIBRARY PROCEDURE MT01:
              AUTHORS:  J. BOOTHROYD, G. WALKER.

              The array y[0:n] contains function values y(x)
              at the sample points of x[0:n] which is sorted
              in ascending order.  The procedure finds an
              approximation to the function at the specified
              point arg by evaluating an mth order polynomial
              (m≤n), choosing the appropriate subset of m+1
              sample points so that these are evenly distributed
              about arg.  If the requested m exceeds n
              the assignment m:=n occurs.
              For  arg < x[0] the procedure extrapolates using
                 x[0] ...... x[m]
              For  arg > x[n] the procedure extrapolates using
                 x[n-m] .... x[n]
```

```
real procedure Lagrange(x,y,arg,n,m);
value arg,m,n; array x,y;  real arg;  integer n,m;
      begin integer i,j,min;
            real term,fac,yest;  switch SS:=out;
            integer procedure setmin(L); label L;
                  begin integer i;
                        switch S:=found;
                        for i:=0 step 1 until n do
                        if arg<x[i] then goto found;
                        i:=n;
            found:      if arg=x[i] then begin yest:=y[i]; goto L end;
                        i:=i-m div 2-1;
                        setmin:= if i<0 then 0 else if i+m>n then n-m else i
                  end setmin;
            if m>n then m:=n;
            min:= setmin(out);
            fac:=1.0;
            for j:=min+m step -1 until min do
            fac:=fac*(arg-x[j]);
            yest:=0;
            for i:=min+m step -1 until min do
                  begin term:=y[i]*fac/(arg-x[i]);
                        for j:=min+m step -1 until i+1,i-1 step -1 until min do
                        term:=term/(x[i]-x[j]);
                        yest:=yest+term
                  end;
out:        Lagrange:=yest
      end Lagrange;
```

MT 02

AITKEN UNEQUAL INTERVAL INTERPOLATION

real **procedure** aitken (x,y,arg,n,m); **value** arg,n,m;

array x,y; **real** arg; **integer** m, n;

comment HUCC LIBRARY PROCEDURE MT 02;

Author : J. Boothroyd.

This procedure performs the same function as MT 01 but is more efficient and economical. The array y[0:n] contains function values y(x) corresponding to the sample points of x[0:n] which is assumed sorted in ascending order. The procedure estimates the value of the function corresponding to a specified point arg by evaluating an m^{th} order polynomial ($m \leq n$), choosing the appropriate subset of m+1 sample points evenly distributed about the value of arg. If the requested value m exceeds n the assignment m:=n occurs. For arg < x[0] the procedure extrapolates using x[0].....x[m]. For arg > x[n] the procedure extrapolates using x[n-m]...x[n];

MT02

```
real procedure aitken(x,y,arg,n,m);  value arg,m,n;  array x,y;  real arg;  integer m,n;
begin       integer i,j,mless1;  real fi,zi;  real array z,f[0:m];  switch s:=cut;
            integer procedure setmin(L);  label L;
                  begin integer i;  switch s:=found;
                        for i:=0 step 1 until n do if arg≤x[i] then goto found;
                        i:=n;
found:                  if arg=x[i] then begin f[m]:=y[i];  goto L end;
                        i:=i-m div 2-1;
                        setmin:= if i<0 then 0 else if i+m>n then n-m else i
                  end setmin;
            if m>n then m:=n;  j:=setmin(out);
            for i:=0 step 1 until m do begin z[i]:=arg-x[j];  f[i]:=y[j];  j:=j+1 end;

            mless1:=m-1;
            for i:= 0 step 1 until mless1 do
            begin fi:=f[i];  zi:=z[i];
                  for j:= i+1 step 1 until m do
                        f[j]:= fi+zi*(f[j]-fi)/(zi-z[j])
            end;
out:        aitken:=f[m]
end;
```

MT 03

AITKEN EQUAL INTERVAL INTERPOLATION

real procedure equipol (xbase,y,arg,n,m,h); **value** xbase,arg,m,n,h;

real xbase,arg,h; **array** y; **integer** m,n;

comment HUCC LIBRARY PROCEDURE MT 03

Author : J. Boothroyd.

An equal interval version of MT 02. Array y[0:n] contains values of a function y(x) corresponding to the equal interval values of the argument xbase,xbase + h,xbase + 2h... .....,xbase + nh. The procedure estimates the value of the function for some specified value arg by evaluating an m^{th} order polynomial ($m \leq n$), choosing the appropriate subset of m+1 sample points evenly distributed about the value arg.

If m exceeds n the assignment m:=n occurs.

For arg < xbase the procedure extrapolates using y[0]....y[m].

For arg > xbase+nh the procedure extrapolates using y[n-m]......y[n];

MT 04

AITKEN N^{th} ORDER INTERPOLATION

real procedure ait(z,f,arg,n); **value** arg,n; **integer** n;

real arg; **array** z,f;

comment HUCC LIBRARY PROCEDURE MT 04

Author : J. Boothroyd

An abridged version of MT 02. Array f[0:n] contains values of a function corresponding to the values of the argument in z[0:n] which may be in any order. The procedure estimates the value of f(z) corresponding to some specified value of z=arg by evaluating an n^{th} order polynomial, using all the values of z,f[0:n];

MT03

```
real procedure equipol(xbase,y,arg,n,m,h);  value xbase,arg,m,n,h;  real xbase,arg,h;  array y;  integer m,n;
      begin integer i,j,mless1;  real jh,fi;  array f[0:m];
            if m>n then m:=n;  i:=entier((arg-xbase)/h)-m div 2;
            j:= if i<0 then 0 else if i+m>n then n-m else i;
            for i:= 0 step 1 until m do f[i]:=y[i+j];
     C-     arg:=arg-j*h;
            mless1:=m-1;
            for i:=0 step 1 until mless1 do
                  begin fi:=f[i];  jh:=h;
                        for j:=i+1 step 1 until m do
                              begin f[j]:=fi+arg*(f[j]-fi)/jh;  jh:=jh+h  end ;
                        arg:=arg-h
                  end;
            equipol:=f[m]
      end equipol;
```

MT04

```
real procedure ait(z,f,arg,n);  value arg,n;  integer n;  real arg;  array z,f;
      begin integer i,j,nless1;  real fi,zi,u;  nless1:=n-1;
            for i:=0 step 1 until nless1 do
                  begin fi:=f[i];  zi:=z[i];  u:=arg-zi;
                        for j:=i+1 step 1 until n do
                        f[j]:=fi+u*(f[j]-fi)/(z[j]-zi)
                  end;
            ait:=f[n]
      end;
```

MT 05

ESTIMATION OF DERIVATIVE OF GIVEN FUNCTION - UNEQUAL INTERVALS

real procedure dydx(x,y,arg,n,m,est); value arg,m,n;

array x,y; real est,arg; integer m,n;

comment HUCC LIBRARY PROCEDURE MT 05

Author : J. Boothroyd.

Array y[0:n] contains values of a function y(x) corresponding to the sample values of the argument in x[0:n] which is assumed sorted in ascending order. The procedure estimates the value of the derivative of y(x) at some specified value of x=arg by evaluating the derivative of an m^{th} order polynomial ($m \leq n$), choosing the appropriate subset of m+1 sample points evenly distributed about the value of arg.

If m>n the assignment m:=n occurs.

For arg <x[0] the procedure extrapolates using x[0]....x[m]

For arg >x[m] the procedure extrapolates using x[n-m]....x[n].

The output parameter est delivers an estimation of y(arg) yielding a value which is the same as would be obtained from the use of MT 02.

In so far as derivative estimation is a numerical process susceptible to large errors this procedure should be used with caution using values of m not exceeding 5;

MT05

```
real procedure dydx(x,y,arg,n,m,est);  value arg,m,n;  array x,y;  real arg,est;  integer m,n;
begin      integer i,j,mless1;  real fi,zi,diffi,zizj,fjfi;  real array z,f,diff[0:m];
           integer procedure setmin;
                begin integer i;  switch s:=found;
                      for i:=0 step 1 until n do if arg<x[i] then goto found;
                      i:=n;
found:                i:=i-m div 2-1;
                      setmin:= if i<0 then 0 else if i+m>n then n-m else i
                end setmin;
           if m>n then m:=n;  j:=setmin;
           for i:=0 step 1 until m do begin z[i]:=arg-x[j];  f[i]:=y[j];  diff[i]:=0.0;  j:=j+1 end;

           mless1:=m-1;
           for i:= 0 step 1 until mless1 do
           begin fi:=f[i];  zi:=z[i];  diffi:=diff[i];
                 for j:= i+1 step 1 until m do
                 begin zizj:=zi-z[j];  fjfi:= f[j]-fi;
                       diff[j]:=diffi+(fjfi+zi*(diff[j]-diffi))/zizj;
                       f[j]:= fi+zi*fjfi/zizj
                 end
           end;
           est:=f[m];
           dydx:=diff[m]
end dydx;
```

MT 06

ESTIMATION OF DERIVATIVE OF GIVEN FUNCTION EQUAL INTERVALS

<u>real</u> <u>procedure</u> equidydx (xbase,y,arg,n,m,h,est);<u>value</u> xbase,arg,m,n,h;

<u>real</u> xbase,arg,est,h; <u>array</u> y; <u>integer</u> m,n;

<u>comment</u> HUCC LIBRARY PROCEDURE MT 06

Author : J. Boothroyd.

An equal interval version of MT 05. Array y[0:n] contains values of a function y(x) corresponding to the sample values of the argument xbase,xbase+h,.....,xbase+nh. The procedure estimates the value of the derivative of the function for some specified value x=arg by evaluating the derivative of an m^{th} order polynomial ($m \leq n$) choosing the appropriate subset of m+1 sample points evenly distributed about arg. If m>n the assignment m:=n occurs.

If arg < xbase the procedure extrapolates using y[0]....y[m]

If arg > xbase+nh the procedure extrapolates using y[n-m]..y[n].

The output parameter est delivers an estimation of the function value y(arg) and yields a value which is the same as would be obtained from MT 03.

This procedure should be used with discretion and for values of m not exceeding 5;

```
real procedure equidydx(xbase,y,arg,n,m,h,est);  value xbase,arg,m,n,h;  real xbase,arg,est,h;  array y;  integer m,n;
      begin integer i,j,mless1;  real jh,fi,diffi,fjfi;  array f,diff[0:m];
            if m>n then m:=n;  i:=entier((arg-xbase)/h)-m div 2;
            j:= if i<0 then 0 else if i+m>n then n-m else i;
            for i:= 0 step 1 until m do begin f[i]:=y[i+j];  diff[i]:=0.0 end;
            arg:=arg-j*h;
            mless1:=m-1;
            for i:=0 step 1 until mless1 do
                  begin fi:=f[i];  jh:=h;  diffi:=diff[i];
                        for j:=i+1 step 1 until m do
                              begin fjfi:=f[j]-fi;
                                    diff[j]:=diffi+(fjfi+arg*(diff[j]-diffi))/jh;
                                    f[j]:=fi+arg*fjfi/jh;  jh:=jh+h
                              end ;
                        arg:=arg-h
                  end;
            est:=f[m];
            equidydx:=diff[m]
      end equidydx;
```

SECOND ISSUE

REARRANGE ELEMENTS OF VECTOR

procedure PERMB(b,r,n); **value** n; **real** **array** b; **integer** **array** r;

integer n;

comment HUCC LIBRARY PROCEDURE NC01 :

AUTHOR J. BOOTHROYD :

A procedure which rearranges the elements of b[1:n] so that b[i] := b[r[i]] i = 1,2,....,n. The procedure should be used after solving Ax = b by the use of LE03 and LE04 in those cases where further processing of the solutions x_i requires x_i to occupy b[i] i = 1,2,.......n;

NC01

```
procedure PERMB(b,r,n); value n; real array b; integer array r;  integer n;
comment rearranges the elements of b[1:n] so that b[i]:=b[r[i]],i=1,2,.....,n;
begin integer i,k; real w; switch S:= L;
      for i:= n step -1 until 2 do
            begin k:= r[i];
              L: if k≠i then
                         begin if k>i then begin k:=r[k]; goto L end;
                               w:= b[i]; b[i]:= b[k]; b[k]:= w
                         end
            end
end PERMB;
```

NC02

PERMUTE ROWS OR COLUMNS OF MATRIX

comment mxperm(a,b,j,k,s,d,n,p); **value** n; **real** a,b; **integer array** s,d;

integer j,k,n,p;

comment HUCC LIBRARY PROCEDURE NC02:

AUTHOR: J. BOOTHROYD

A procedure using Jensen's device which exchanges rows or columns of a matrix to achieve a rearrangement specified by the permutation vectors s,d[1:n]. Elements of s specify the original source locations while elements of d specify the destination locations. Normally a and b will be called as subscripted variables of the same array. The parameters j,k nominate the subscripts of the dimension affected by the permutation, p is Jensen's parameter. As an example of the use of this procedure suppose r,c[1:n] to contain the row and column subscripts of the successive matrix pivots following an in-situ matrix inversion. The two calls

mxperm(a[j,p],a[k,p],j,k,r,c,n,p)

and mxperm(a[p,j],a[p,k],j,k,c,r,n,p)

will respectively perform the required rearrangement of rows and columns.

```
procedure mxperm(a,b,j,k,s,d,n,p);  value n;  real a,b;  integer array s,d;  integer j,k,n,p;
begin integer array tag,loc[1:n];  integer i,t,tagj;  real w;
          comment set up initial vector tag number and address arrays;
          for i:=1 step 1 until n do tag[i]:=loc[i]:=i;
          comment start permutation;
          for i:=1 step 1 until n do
               begin t:=s[i];  j:=loc[t];  k:=d[i];
                    if j≠ k then
                         begin for p:=1 step 1 until n do
                                   begin w:=a;  a:=b;  b:=w end;
                              tag[j]:=tag[k];  tag[k]:=t;  tagj:=tag[j];
                              loc[t]:=loc[tagj];  loc[tagj]:=j
                         end jk conditional
               end i loop
end mxperm;
```

PERMUTATION OF ELEMENTS OF A VECTOR

procedure vectorperm(m,d,n,mode,endperm); **value** n,mode; **integer** n,mode

array m; **integer array** d;

comment HUCC LIBRARY PROCEDURE NC03:

AUTHOR: J. BOOTHROYD

A procedure for generating, at each entry, a permutation of the elements m[1],m[2] .... m[n] of m[1:n]. (n-1)! successive entries generate all n! permutations.
The permutation is controlled by a variable radix counter, the array d[2:n], with digit positions d[2],d[3] ... d[n] where the subscript indicates the radix value. Starting with d=(0,0,0 .... 0) one is added to the counter at each entry to the procedure. One and only one element of d increases each time and all element positions below this are reset to zero. Denoting by k the subscript position of the counter element which increases the permutation rules are:-

(k odd) or (k even and d[k]<2)	interchange m[k],m[k-1]
k even for 2<d[k]<k	interchange m[k],m[k-d[k]]

A call of vectorperm initialises d to (0,0,0,0,...0) with mode=1 preparatory to further calls with mode=2. At the n!th call the counter d resets to zero and the procedure exits to the label endperm;

NC03

```
procedure vectorperm(m,d,n,mode,endperm);  value n,mode;  integer n,mode;
integer array d;  array m;  label endperm;
      begin integer k,j;  real temp;  switch s:=set,run,swap,exit;
            goto s[mode];
set:        for k:=2 step 1 until n do  d[k]:=0;  goto exit;
run:        j:=-1;
            for k:=2 step 1 until n do
                  begin if d[k]≠k-1 then goto swap;   d[k]:=0;  j:=-j end;
            goto endperm;
swap:       d[k]:=d[k]+1;
            if j≠1 then j:= if 2>d[k] then 1 else d[k];
            temp:=m[k];  m[k]:=m[k-j];  m[k-j]:=temp;
exit: end vectorperm;
```

NC04

PERMUTE ROWS OR COLUMNS OF MATRIX

```
procedure mxperml(a,b,j,k,s,d,n,p); value n; real a,b; integer array s,d;
        integer j,k,n,p;

comment HUCC LIBRARY PROCEDURE NC04:
        AUTHOR: J. BOOTHROYD

        A procedure which is the same as, but avoids the use of the
        local arrays of, NC02. It is, therefore, more economical of
        space but slower in operation. The identifiers of NC02 and
        NC04 agree over the first six characters, the parameter specifications
        are identical and NC04 is thus a direct replacement for NC02.
```

NC04

```
procedure mxperm1(a,b,j,k,s,d,n,p);  value n;  real a,b;  integer array s,d;  integer j,k,n,p;
        begin integer i; switch ss:= scan; real w;
            for i:= n step -1 until 2 do
               begin j:= s[i]; k:= d[i];
                 scan: if j≠k then
                       begin for p:= i+1 step 1 until n do
                             if j=d[p] then begin j:= s[p]; goto scan end;
                             for p:= 1 step 1 until n do begin w:= a; a:= b; b:= w end
                       end
              end
        endmxperm;
```

NC 05

PRE OR POST MULTIPLY MATRIX BY PERMUTED IDENTITY MATRIX

procedure permx (a,b,j,k,r,n,p,inv); **value** n,inv;

real a,b; **integer** j,k,n,p,inv; **integer** **array** r;

comment HUCC LIBRARY PROCEDURE NC 05

Author : J. Boothroyd

A procedure using Jensen's device whereby a matrix A[1:n,1:n] may be either pre or post multiplied by either a permuted identity matrix Ir or by the inverse of Ir. All row (or column) exchanges are performed on A without the use of an array of the same size. The inverse of Ir is its transpose and both Ir and Ir^{-1} may be defined by a permutation vector r[1:n] as follows :-

$$Ir[r[i],j] = \begin{matrix} 0 & i \neq j \\ 1 & i=j \end{matrix} \qquad Ir^{-1}[i,r[j]] = \begin{matrix} 0 & i \neq j \\ 1 & i=j \end{matrix}$$

for i=1,2,......n, j=1,2,......,n.

Values of the parameter inv $\geq$ 0 accomplish multiplication by Ir. Otherwise, for inv < 0 (inv = -1 is appropriate) multiplication is by Ir^{-1}.

The following calls illustrate how to achieve pre or post multiplication

Premultiplication by Ir permx(A[j,p],A[k,p],j,k,r,n,p,0)

Post multiplication by Ir^{-1} permx(A[p,j],A[p,k],j,k,r,n,p,-1);

```
procedure permx(a,b,j,k,r,n,p,inv);  value n,inv;  real a,b;  integer j,k,n,p,inv;  integer array r;
begin integer i;  real w;
      procedure itori;
      begin switch s:=L;
            for i:=n step -1 until 2 do
                  begin k:=r[i];  j:=i;
                   L:    if j≠k then
                                    begin if k>j then begin k:=r[k];  goto L end end;
                         for p:=1 step 1 until n do
                               begin w:=a;  a:=b;  b:=w end
                  end
      end itori;
      procedure ritoi;
      begin switch s:=scan;
            for i:=n step -1 until 2 do
                  begin j:=i;  k:=r[i];
                   scan:  if j≠k then
                               begin for p:=i+1 step 1 until n do
                                     if j=r[p] then begin j:=p;  goto scan  end;
                                     for p:=1 step 1 until n do
                                           begin w:=a;  a:=b;  b:=w  end
                               end
                  end
      end ritoi;
      if inv>0 then itori else ritoi
end permx;
```

INVERSEPERMUTATION OF INTEGER VECTOR

```
procedure inversepermb(P,n); value n; integer n; integer array P;
```

comment HUCC LIBRARY PROCEDURE NC 06

Author : J. Boothroyd

Rearranges the natural number integer elements of P[1:n] to effect an inversepermutation

i.e. Given

	1	2	3	4	5	6	7
P	6	2	1	3	7	5	4

the procedure produces

	1	2	3	4	5	6	7
P	3	2	4	7	6	1	5

This process is best described by :-

```
for i:=1 step 1 until n do Q[P[i]]:=i;
for i:=1 step 1 until n do P[i]:=Q[i]
```

though NC 06 accomplishes the transformation without the use of the auxiliary array Q;

NC 06

```
procedure inversepermb(P,n); value n; integer n; integer array P;
    begin             integer i,j;  switch s:=next;
        for i:=1 step 1 until n do P[i]:=-P[i];
        for n:=n step -1 until 1 do
            begin j:=n;
           next:  i:=P[j];  if i>0 then begin j:=i;  goto next end;
                  P[j]:=P[-i];  P[-i]:=n
            end
end inversepermb;
```

CRITICAL PATH SCHEDULING

procedure CRITICALPATH(n,I,J,DIJ,ES,LS,EF,LF,TF,FF,i,out);

integer i,n; **integer array** I,J,DIJ,ES,LS,EF,LF,TF,FF; **label** out;

comment HUCC LIBRARY PROCEDURE OP01:

AUTHOR ACM 74 (improved):

Given the total number of jobs n of a project, the pair I[k],J[k] representing the kth job as a directed line joining event I[k] to event J[k] (I[k]<J[k],k=1,2,...,n) and a duration vector DIJ, the procedure determines the earliest starting time ES[k], latest starting time LS[k], earliest completion time EF[k], latest completion time LF[k] total float TF[k] and free float FF[k]. I[1] must be 1 and the I[k],J[k] must be in ascending order with no values in the sequence 1 to n-1 omitted from the I[k] sequence. The critical path is the chain of all jobs with zero total float. The following tests are included:

(a) That I[k]<J[k], (b) All I[k] are in ascending sequence

(c) No I[k] is missing.

Failure to meet these requirements causes exit to the label parameter out. This exit is so arranged that by the use of an actual parameter OUT[i], for example, the respective failure paths are (a) OUT[1], (b) OUT[2], (c) OUT[3];

OP01

```
procedure CRITICALPATH(n,I,J,DIJ,ES,LS,EF,LF,TF,FF,i,out);
          integer i,n;  integer array I,J,DIJ,ES,LS,EF,LF,TF,FF;  label out;
          begin integer k,index,m,tiik,tejk,dijk;  integer array ti,te[1:n];
               index:=1;
               for k:=1 step 1 until n do
                    begin if I[k]>J[k] then begin i:=1;  goto out end;
                         if I[k]<index then begin i:=2;  goto out end;
                         if I[k]>index and I[k]≠ index+1 then begin i:=3;  goto out end;
                         if I[k]=index+1 then index:=I[k]
                    end;
               for k:=1 step 1 until n do begin ti[k]:=0;  te[k]:=9999 end;
               for k:=1 step 1 until n do
                    begin m:=ti[I[k]]+DIJ[k];
                         if ti[J[k]]<m then ti[J[k]]:=m
                    end ti;
               te[J[n]]:=ti[J[n]];
               for k:=n step -1 until 1 do
                    begin m:=te[J[k]]-DIJ[k];
                         if te[I[k]]>m then te[I[k]]:=m
                    end te;
               for k:=1 step 1 until n do
                    begin tiik:=ti[I[k]];  tejk:=te[J[k]];  dijk:=DIJ[k];
                         ES[k]:=tiik;  EF[k]:=tiik+dijk;
                         LS[k]:=tejk-dijk;  LF[k]:=tejk;
                         TF[k]:=tejk-tiik-dijk;  FF[k]:=ti[J[k]]-tiik-dijk
                    end
          end CRITICALPATH;
```

SS01

REAL RANDOM NUMBER GENERATOR

real procedure random;

Comment HUCC LIBRARY PROCEDURE SS01:

AUTHOR J. BOOTHROYD :

A procedure to generate a stream of uniformly distributed real random numbers in the interval $0 \leq$ random ≤ 1. Random integers xr are formed by the operation.

$$xr := ((2^{20} - 3)\ xr - 1) \text{ modulo } 2^{38}$$

and each real value random is formed from a corresponding value of xr by discarding the least significant nine bits. The resulting 30 bit truncated integer is then treated as a fraction f in the interval $0 \leq f < 1$ and an appropriate adjustment made in the integer to real conversion. The period of the sequence is thus 2^{38} though, by virtue of the truncation, each value of random will repeat 512 times in a complete cycle.

To obtain a uniform distribution in the interval [a,b] use the expression a + (b-a) * random. To obtain a symmetrical triangular distribution with mean =1 in the range $0 \leq y < 2$ use the statement y:= random + random.

Note that random is a procedure without parameters. The initial value of random is thus indeterminate. In circumstances where this might be inconvenient, as for instance in successive program testing runs, it would be preferable to use LIBRARY SS02;

SS02

Real Random Number Generator

real procedure randG2 (x, start); **value** x, start; **integer** x, start;

comment HUCC LIBRARY PROCEDURE SS02:

AUTHOR J. BOOTHROYD :

A procedure having the same function as LG1 but with facilities for determining the starting value of the stream of random numbers. A call of the procedure with start = 0 and x = X will cause the first number to be generated from

$$((2^{20} - 3) X - 1) \text{ modulo } 2^{38}$$

Subsequent calls of the form randG2 (0, 1) will generate numbers from the value of randG2 at the immediately previous call;

```
comment LG1;                                                              SS01
        real procedure random;
            begin own integer xr;  integer three8,cc;  real f;  switch S:=out;
                cc:=274877906432;  three8:=38;
                elliott(0,2,xr,0,0,1,0);
                elliott(5,0,1,0,0,5,xr);
                elliott(5,0,19,0,0,4,xr);
                elliott(5,0,18,0,5,7,0);
                elliott(2,0,xr,0,0,3,cc);
                elliott(6,5,4096,0,0,5,three8);
                elliott(2,0,f,0,4,3,out);
out:            random :=f
            end random;
```

```
comment LG2;  real procedure rand G2(x,start);  value x,start;  integer x,start;     SS02
            begin own integer xr,cc, three8;  real f;  switch S:=out;
                if start =0 then begin xr:=x;  three8:=38;  cc:=274877906432 end;
                elliott(0,2,xr,0,0,1,0);
                elliott(5,0,1,0,0,5,xr);
                elliott(5,0,19,0,0,4,xr);
                elliott(5,0,18,0,5,7,0);
                elliott(2,0,xr,0,0,3,cc);
                elliott(6,5,4096,0,0,5,three8);
                elliott(2,0,f,0,4,3,out);
out:            rand G2:=f
            end randG2;
```

RANDOM NUMBER GENERATOR - POISSON DISTRIBUTION

integer procedure poisson(lambda); value lambda; real lambda;

comment HUCC LIBRARY PROCEDURE SS03:

AUTHOR J. BOOTHROYD :

A procedure which generates integers x having the Poisson probability distribution

$$P(x) = \frac{\varepsilon^{-\lambda} \lambda^{x}}{x!}$$

The numbers are generated from a distribution uniform in $0 \leq y < 1$ by forming x terms of a continued product satisfying

$$y_1 \; y_2 \; y_3 \ldots\ldots y_x < e^{-\lambda}$$

For large values of λ the method is therefore not particularly efficient, and it may be preferable to use the fact that for large λ the Poisson distribution tends to a normal distribution and employ SS04;

SS03

```
integer procedure poisson(lambda); value lambda; real lambda;
begin own integer xr; integer three8,n; real elambda,try; switch s:= L;
      three8 := 38; n:=-1; try:= 1.0; elambda:=exp(-lambda);
  L: elliott(0,2,xr,0,0,1,0);
      elliott(5,0,1,0,0,5,xr);
      elliott(5,0,19,0,0,4,xr);
      elliott(5,0,18,0,5,7,0);
      elliott(1,6,xr,0,5,4,29);
      elliott(5,5,9,0,6,5,4096);
      elliott(0,5,three8,0,6,3,try);
      elliott(2,0,try,0,6,2,elambda);
      elliott(2,2,n,0,4,1,L);
      elliott(7,3,4,1,4,3,1);
            poisson:= n
endpoisson;
```

SS04

RANDOM NUMBER GENERATOR - STANDARD NORMAL DISTRIBUTION.

real **procedure** normal;

comment HUCC LIBRARY PROCEDURE SS04:

AUTHOR J. BOOTHROYD :

Generates random numbers with a standard normal distribution (mean zero and unit standard deviation). The numbers are generated from a distribution uniform in $0 \leq y < 1$ by summing twelve values of y and adjusting the mean by subtracting 6.0

$$\text{normal} := \sum_{1}^{12} y_i - 6.0$$

To generate numbers having a mean u and standard deviation σ use the transformation

```
musigma := sigma*normal + mu ;
```

SS04

```
real procedure normal;
begin own integer xr; integer three8,count; real sum; switch s:= L;
      three8:=38; count:= -11; sum:= -6.0;
   L: elliott(0,2,xr,0,0,1,0);
      elliott(5,0,1,0,0,5,xr);
      elliott(5,0,19,0,0,4,xr);
      elliott(5,0,18,0,5,7,0);
      elliott(1,6,xr,0,5,4,29);
      elliott(5,5,9,0,6,5,4096);
      elliott(0,5,three8,0,6,0,sum);
      elliott(2,0,sum,0,3,2,count);
      elliott(4,1,L,0,0,0,0);
      elliott(7,3,4,1,4,3,1);
            normal:= sum
endnormal;
```

SS05

GENERATE LARGE INTEGER RANDOM NUMBERS

<u>integer</u> <u>procedure</u> bigrn;

<u>comment</u> HUCC LIBRARY PROCEDURE SS05:

AUTHOR J. BOOTHROYD :

A procedure to generate, equiprobably, integers in the range $0 \leq$ bigrn $< 2^{38}-1$. There is thus a 0.5 probability that bigrn $> 2^{37}$. The numbers are formed by the recurrence

$$xr := ((2^{20}-3)xr-1) \text{modulo } 2^{38}.$$

For repeatable program testing purposes it may be preferable to use SS06;

SS06

GENERATE LARGE INTEGER RANDOM NUMBERS

<u>integer</u> <u>procedure</u> bigrnx(x,start); <u>value</u> x,start; <u>integer</u> x,start;

<u>comment</u> HUCC LIBRARY PROCEDURE SS06:

AUTHOR J. BOOTHROYD :

Performs the same function as SS05 but with facilities for initialising the starting value of the stream of random numbers. With start=0 the value of bigrnx is generated from the value supplied by the parameter x. For start=1 the value of x is ignored and operation of SS06 is the same as SS05;

SS07

GENERATE SMALL INTEGER RANDOM NUMBERS

<u>integer</u> <u>procedure</u> rn(n); <u>value</u> n; <u>integer</u> n;

<u>comment</u> HUCC LIBRARY PROCEDURE SS07:

AUTHOR J. BOOTHROYD :

A procedure to generate, equiprobably, integers in the range $0 \leq rn < 2^n$. The basic generation scheme is the recurrence

$$xr := ((2^{20}-3)xr-1) \text{ modulo } 2^{38}$$

The value given by the top n bits of each generated xr is assigned to rn. Where repeatable program testing results are required, it may be preferable to use SS08;

SS08

GENERATE SMALL INTEGER RANDOM NUMBERS

<u>integer</u> <u>procedure</u> rn(x,start,n); <u>value</u> x,start,n; <u>integer</u> x,start,n;

<u>comment</u> HUCC LIBRARY PROCEDURE SS08:

AUTHOR J. BOOTHROYD :

A procedure performing the same function as SS07 but with facilities for determining the starting value of the stream of random numbers. With start=0 the value of the first generated number is determined by the value of x. For start=1, x is ignored, and the procedure is identical with SS07;

SS05

```
integer procedure bigrn;
begin own integer xr;
      elliott(0,2,xr,0,0,1,0);
      elliott(5,0,1,0,0,5,xr);
      elliott(5,0,19,0,0,4,xr);
      elliott(5,0,18,0,5,7,0);
      elliott(2,0,xr,0,0,0,0);
      elliott(7,3,4,1,4,3,1);
            bigrn:= xr
endbigrn;
```

SS06

```
integer procedure bigrnx(x,start); value x,start; integer x,start;
begin own integer xr;
      if start=0 then xr:= x;
      elliott(0,2,xr,0,0,1,0);
      elliott(5,0,1,0,0,5,xr);
      elliott(5,0,19,0,0,4,xr);
      elliott(5,0,18,0,5,7,0);
      elliott(2,0,xr,0,0,0,0);
      elliott(7,3,4,1,4,3,1);
            bigrnx:= xr
endbigrnx;
```

SS07

```
integer procedure rn(n); value n; integer n;
begin own integer xr;
      elliott(0,2,xr,0,0,1,0);
      elliott(5,0,1,0,0,5,xr);
      elliott(5,0,19,0,0,4,xr);
      elliott(5,0,18,0,5,7,0);
      elliott(1,6,xr,0,6,7,n);
      elliott(5,4,0,0,2,0,n);
      elliott(7,3,4,1,4,3,1);
            rn:=n
endrn;
```

SS08

```
integer procedure rnx(x,start,n); value x,start,n; integer x,start,n;
begin own integer xr;
      if start=0 then xr:= x;
      elliott(0,2,xr,0,0,1,0);
      elliott(5,0,1,0,0,5,xr);
      elliott(5,0,19,0,0,4,xr);
      elliott(5,0,18,0,5,7,0);
      elliott(1,6,xr,0,6,7,n);
      elliott(5,4,0,0,2,0,n);
      elliott(7,3,4,1,4,3,1);
            rnx:=n
endrnx;
```

ZM01

OUTPUT BINARY VALUE

procedure binary(i,g);

value i,g; **integer** i,g;

comment HUCC LIBRARY PROCEDURE ZM01:

AUTHOR: W. G. WARNE;

Prints on current device the binary value of the integer i in groups of g digits separated by a space starting from the most significant binary digit. No character is output before the first digit. If g = 0 no spaces are output to cause grouping.

ZM02

OUTPUT OCTAL VALUE

procedure octal(i,g);

value i,g; **integer** i,g;

comment HUCC LIBRARY PROCEDURE ZM02:

AUTHOR: W. G. WARNE:

Outputs on current device the octal value of the integer i in groups of g octal digits, separated by a space, starting from the most significant octal digit. No character is output before the first octal digit.
If g = 0 the integer is output as a single group.

ZM01

```
procedure binary(i)in groups of:(g);
value i,g;  integer i,g;
begin       integer n,gc;
            switch sw := neg, cont;
            gc:=0;
            for n:=1 step 1 until 39 do
                begin elliott(3,0,i,0,4,1,neg);
                      print £0?;
                      goto cont;
neg:                  print £1?;
cont:                 gc:=gc+1;
                      if gc=g then begin print ££s??;  gc:=0 end;
                      elliott(3,0,i,0,5,5,1);
                      elliott(7,3,4,1,4,3,1);
                      elliott(2,0,i,0,0,0,0);
                end
end binary;
```

ZM02

```
procedure octal(i)in groups of:(g);
value i,g;  integer i,g;
begin       integer n,gc,temp,seven;
            gc:=0;
            seven:=7;
            for n:=36 step -3 until 0 do
                  begin elliott(3,0,i,0,6,7,n);
                        elliott(5,1,0,0,0,3,seven);
                        elliott(2,0,temp,0,2,2,gc);
                        print sameline,special(1),digits(1),temp;
                        if gc=g then begin print ££s??;  gc:=0 end
                  end
end octal;
```

SENSE NUMBER GENERATOR KEY

boolean **procedure** ng(i); **value** i; **integer** i;

comment HUCC LIBRARY PROCEDURE ZM03:

AUTHOR ELLIOTT AUTOMATION:

This procedure, when called as ng(x) where x is an integer, $1 \leq x \leq 39$, takes the value **true** if the key numbered x on the word generator is depressed at the time of call. If key x is not depressed, the value of ng is **false**. A typical call would then be

if ng(x) **then** .... ;

INPUT NEWLINE STRINGS

procedure charead(array); **integer array** array;

comment HUCC LIBRARY PROCEDURE ZM04:

AUTHOR P. W. FORD :

A call of charead causes the input tape to be searched for the first occurrence of a non newline non blank character immediately following a newline ℓ, i.e. blank tape and a succession of newlines is ignored. This first character and all subsequent non blank characters up to but not including the next newline are stored in the integer array array in a form consistent with subsequent use of the procedure outstring. The character string is stored starting in array[1] and adequate bounds for this array must be declared (see 503 TECH MANUAL 2.1.3.2 section 2.6). There is no protection on the bounds of array and the input of a string longer than can be accommodated is likely to be discovered by a subscript overflow failure when outstring is called;

ZM03

```
boolean procedure ng(i); value i; integer i;
              begin switch s:=1;
                    elliott(0,2,0,0,0,0,0);
                    elliott(0,0,i,1,5,4,8191);
                    elliott(1,6,i,0,7,0,0);
                    elliott(2,3,i,0,4,3,1);
                1:  ng:=i ≠ 0
              end ng(i);
```

ZM04

```
procedure charead(array);
integer array array;
      begin integer i,j,k,l,m;
              switch ss:=search,next,read,store,shift,test,out,back,on;
              m:=2;
              l:=64;
              elliott(0,2,0,0,5,5,38);
              elliott(2,0,k,0,4,3,search);
search:       elliott(0,6,0,0,7,1,0);
              elliott(0,5,m,0,4,2,next);
              elliott(4,0,search,0,0,0,0);
next:         elliott(0,2,0,0,0,7,array);
              elliott(1,6,j,0,0,0,0);
read:         elliott(7,1,0,0,4,2,read);
              elliott(0,5,m,0,4,2,read);
              elliott(0,4,m,0,5,0,7);
back:         elliott(3,0,l,0,0,0,0);
shift:        elliott(5,4,7,0,1,6,i);
              elliott(7,1,0,0,0,5,m);
              elliott(4,2,out,0,0,4,m);
              elliott(5,0,7,0,3,0,i);
              elliott(4,3,store,0,4,0,shift);
store:        elliott(2,2,j,1,1,6,0);
              elliott(4,0,back,0,0,0,0);
out:          elliott(3,0,i,0,0,0,0);
test:         elliott(4,3,on,0,5,4,7);
              elliott(4,0,test,0,0,0,0);
on:           elliott(0,4,k,0,0,0,0);
              elliott(2,2,j,1,1,6,0)
      end charead;
```

Algol A.D.T. Read and Punch Binary

Two procedures designed to meet the requirements of programs using paper tape as temporary storage so that partial results output by one program are used solely as subsequent input data for the same or another program. In these cases the most appropriate information format is pure binary and further efficiency has been obtained by taking advantage of the interrupt facilities of the 503 so that input and output takes place simultaneously with computation or the use of peripheral devices other than those used by ZM05 or ZM06, which may operate together.

Both procedures use a boolean parameter flag to indicate when a transfer initiated by a call of ZM05 or ZM06 is complete. Programs using these procedures should not access the data transfer locations unless flag ≡ true.

Realignment of a tape, output by ZM05, at subsequent use by ZM06 may be accomplished in several ways:-

(a) By arranging to output a correctly terminated number which is read in and checked before a call of ZM06

(b) By arranging to output an identifiable character and, on reinput, to execute a buffer search for the same character, or more simply but without any check, execute a call of the procedure advance which ignores blank tape and erase characters.

ZM05

procedure out(a,n,flag); value n; integer n; array a; boolean flag;

comment HUCC LIBRARY PROCEDURE ZM05
AUTHOR : W.G. WARNE

A procedure which outputs, in binary, on punch 1, the first $n(\geq 1)$ locations of array a, i.e. by rows. The parameter flag is assigned the value true when the transfer is complete. Total time is approximately 60n milliseconds of which 99% is available for simultaneous computation;

ZM06

<u>procedure</u> in(a,n,flag); <u>value</u> n; <u>integer</u> n; <u>array</u> a; <u>boolean</u> flag;

<u>comment</u> HUCC LIBRARY PROCEDURE ZM06

AUTHOR : W.G. WARNE

A procedure which reads, from reader 1, tapes previously output by ZM05. (see the notes above concerning realignment of tape). Data from the tape is input to the first n($\geq$1) locations of array a. Total time is approximately 6n milliseconds of which 90% is available for simultaneous computation;

ZM05

```
procedure out(a,n,flag);
value n;
array a;
boolean flag;
integer n;
begin       own integer aa,i,ab,temp,hold,x,mask,imask;
            switch s:= fin,L1,L2,L5;
            mask := 127;
            x := 32;
            imask := 223;
            flag := false;
            aa := address(a);
            ab := aa + n - 1;
            goto L1;
L2:         i := i - 262144;
            elliott(3,0,7886,0,0,3,imask);
            elliott(0,4,x,0,2,0,7886);
            elliott(6,7,7886,0,7,2,0);
            elliott(0,2,i,0,5,5,20);
            elliott(2,6,7895,0,1,0,7892);
            elliott(2,0,hold,0,2,6,7893);
            elliott(2,6,7894,0,3,0,fin);
            elliott(1,6,4,0,0,2,0);
            elliott(5,5,3,0,2,0,7896);
            for aa := aa step 1 until ab do
            begin       elliott(0,6,aa,1,0,4,0);
                        elliott(2,0,temp,0,0,0,0);
                        for i := 35 step -7 until 0 do
                        begin       elliott(3,0,temp,0,6,7,i);
                                    elliott(5,1,0,0,0,3,mask);
                                    elliott(2,0,x,1,7,4,0);
                                    elliott(6,6,7893,0,0,0,0);
L1:                                 elliott(7,3,i,0,4,0,L2);
                                    elliott(7,5,7168,0,5,5,2);
                                    elliott(7,3,x,1,4,3,1);
                                    elliott(4,1,L5,0,0,0,0);
                        end;
            end;
            flag := true;
            elliott(3,0,hold,0,2,0,7892);
            elliott(3,0,imask,0,0,3,7886);
            elliott(2,0,7886,1,7,2,0);
L5:         elliott(6,6,7893,0,0,0,0);
fin: end;
```

ZM06

```
procedure in (a,n,flag);
value n;  integer n;
array a;
boolean flag;
      begin own integer aa,i,ab,temp,hold,five,imask;
            switch s:= fin,L1,L2,L3,L4,L5;
            flag:= false;
            aa:=address(a);
            ab:=aa+n-1;
            five := 5;
            temp := 64;
            imask := 191;
            goto L1;
L2:         i:=i-262144;
            elliott(3,0,7886,0,0,3,imask);
            elliott(0,4,temp,0,2,0,7886);
            elliott(6,7,7886,0,7,2,0);
            elliott(0,2,i,0,5,5,20);
            elliott(2,6,7888,0,1,0,7887);
            elliott(2,0,hold,0,2,6,7889);
            elliott(3,0,five,0,2,1,i);
            elliott(2,6,7890,0,3,0,fin);
            elliott(1,6,4,0,0,2,0);
            elliott(5,5,3,0,2,0,7891);
            elliott(0,6,0,0,4,0,L3);
L4:         elliott(2,6,temp,0,3,0,five);
            elliott(2,1,i,0,6,6,7888);
L1:         elliott(7,3,i,0,4,0,L2);
            elliott(7,5,7168,0,4,1,L5);
            elliott(3,0,temp,0,5,5,7);
L3:         elliott(7,1,0,0,2,0,temp);
            elliott(3,2,i,0,4,1,L5);
            elliott(3,0,temp,0,6,7,aa);
            elliott(2,0,0,0,3,2,aa);
            elliott(0,5,ab,0,4,1,L4);
            flag:=true;
            elliott(3,0,hold,0,2,0,7887);
            elliott(3,0,imask,0,0,3,7886);
            elliott(2,0,7886,1,7,2,0);
L5:         elliott(6,6,7888,0,0,0,0);
fin: end;
```

ZM07

COPY LEGEND FROM DATA TAPE

procedure copy(n); value n; integer n;

comment HUCC LIBRARY PROCEDURE ZM07:

AUTHOR : W.G. WARNE:

Copies characters from reader (1) to punch (1) up to but not including the first character having binary value n where $0 \leq n \leq k27$ in 8-hole mode or $0 \leq n \leq 31$ in 5-hole mode;

ZM08

Read ALGOL BUFFER

integer procedure BUFFER (d); value d; integer d;

comment HUCC LIBRARY PROCEDURE ZM08:

AUTHOR : W.G. WARNE:

Takes the value of the character in buffer(d) where d(1,2 or 3) is the device number. If d is not 1,2 or 3 the message "BUFFER error" is displayed followed by a data wait. Continuation assigns the value zero to BUFFER. The procedure has been tested with Algol 1/3 Tapes 1 and 2 only on a basic machine. There is thus no guarantee that it is operative with the backing store version of Algol 1;

```
procedure copy(n);
value n;  integer n;
comment Copies characters from reader 1 to punch 1 until a character with
            binary value n is met. This character is not copied.
            Note :- 0≤n≤127 in 8 hole mode and 0≤n≤31 in 5 hole mode.;
begin       integer t;
            switch L := L1,L2;
   L1:      elliott(0,6,0,0,7,1,0);
            elliott(2,0,t,0,0,5,n);
            elliott(4,6,L2,0,6,7,t);
            elliott(7,4,0,0,4,4,L1);
   L2:
end copy;
```

```
integer procedure BUFFER(d);
comment Takes binary value of character in buffer for device d (d=1,2 or 3).;
value d;  integer d;
if d < 1 or d > 3 then
      begin print punch(3), ££l?BUFFER error?,wait;
            BUFFER := 0
      end
else begin  elliott(3,0,7920,0,2,4,d);
            elliott(0,6,d,1,0,4,79);
            elliott(2,0,d,0,0,0,0);
            BUFFER := d
      end BUFFER;
```

PRINT MATRIX

```
procedure matprint(fields,rows,cols,i,j,margin,gap,var);
value fields,rows,cols;  integer fields,rows,cols,i,j;  real var;
string margin,gap;
comment   HUCC LIBRARY PROCEDURE ZP02:
          AUTHOR:  J. BOOTHROYD      :
```

A Jensen procedure for printing the elements of either one or two dimensional arrays with facilities for specifying

(a) fields .... the number of numbers printed per line

(b) margin .... the prefix required at the start of each line

(c) gap .... the number of spaces (or other symbols) printed between numbers.

Notes

1. i and j are the Jensen parameters, with ranges i,1(1)rows,j,1(1)cols.

2. var must be called as a real variable, A[i,j] for a matrix A[1:rows,1:cols] or a[j] for a vector a[1:cols]. For arrays with other than unit lower bound the appropriate adjustments should be made to the subscript expressions and to the values used for rows and cols. For example, a in the case of TABLE[0:6,0:10] the appropriate parameters would be

   rows = 7   cols = 11

   var = TABLE[i-1,j-1]

3. The procedure does not specify the style of number printing. The user has the option of any of the following methods of determining the required format:

   (a) By calling the required setting procedure (aligned(m,n),digits(n)etc) globally.

   (b) By calling the required setting procedure locally in a print statement.

   e.g. print aligned(2,3),matprint(   );

3 (cont.)

(c) By declaring a private format procedure to permit optional setting procedures depending on circumstances

e.g. <u>real</u> <u>procedure</u> format(x); <u>value</u> x; <u>real</u> x;

```
    begin if abs(x)<0.1 then scaled(7) else aligned(5,6)
          format:= x
    end;
```

Within matprint this procedure would be used as an actual parameter as in

```
matprint(5,10,10,i,j,££s5??,££s4??,format(A[i,j]))
```

4. It is the users responsibility to ensure that the total character requirements of the setting procedure, margin and gap are consistent with the requested value of fields and the limit of 120 characters per line.

<u>Examples</u>: To print A[1:20,1:20] with 5 spaces as margin, 3 spaces between numbers and 6 numbers per line.

```
matprint(6,20,20,i,j,££s5??,££s3??,A[i,j])
```

To print LIST[1:n] with each line starting one tab from the paper margin, 7 numbers per line, 4 spaces between numbers.

```
matprint(7,1,n,i,j,££t??,££s4??,LIST[j])
```

ZP02

```
procedure matprint(fields,rows,cols,i,j,margin,gap,var);  value fields,rows,cols;
            integer fields,rows,cols,i,j;  real var;  string margin,gap;
      begin integer k,line,np;  switch s:=more;
            for i:=1 step 1 until rows do
                  begin print ££12??;  j:=0;  np:=cols;
            more:       np:=np-fields;  line:=fields;
                        if np < 0 then line:=np+fields;
                        print margin;  sameline;
                        for k:=1 step 1 until line do
                              begin j:=j+1;
                                    if k ≠ 1 then print gap;
                                    print var
                              end;
                        print ££1??;  if np > 0 then goto more
                  end i
end matprint;
```

CONTROL FLEXOWRITER PAGE PRINTING

procedure pageout(string,type,resultline,array,start);

value type,resultline,start; **string** string;

integer type,resultline,start; **integer** **array** array;

comment HUCC LIBRARY PROCEDURE ZP03:

AUTHOR P. W. FORD :

A procedure to control the automatic pageing of results printed on Lamson Paragon Paraflo paper (Form No. 1112). The number of resultlines is internally set at 50 and change to a new page occurs if the next batch of output would cause this limit to be exceeded. Page numbering is provided and facilities are provided for including on each page indicative information such as job identification, result headings, etc. A call of pageout must occur before each print statement or group of print statements;

Parameters

type — an integer with permissible values 0,1,2.

type = 0 prepares the procedure by initialising the the line and page counters.

type = 1 for normal page control.

type = 2 initiates an immediate page change and resets the page counter.

resultline — an integer specifying the number of lines of output in the immediately following **print** statement(s). This parameter is ignored, and may be 0, if type = 0 or 2.

Parameters (cont.)

array — the identifier of an array containing some job or other indicative string to be output at the head of each newpage on the line above the page number. If this facility is not required (start = 0, see below) use any declared integer array identifier if one exists otherwise declare a minimal array, e.g. A[1:1] for this purpose.

start — an integer parameter which controls the outstring facility used for job identification. If start = 0 the **outstring** facility is ignored, while start ≠ 0 causes the string starting at array[start] to be output at the head of each newpage. Note that start may be an integer constant, and does not have to be an integer variable identifier.

string — Array string (including the empty string £?). This is output on each newpage following the line bearing the page number. This facility permits the printing of result headings, etc.

NOTES — STRINGS should NOT contain newline characters. This procedure does not control line changing. This must be done in the print statements which pageout monitors. The procedure must be initialised once with type = 0, and this is preferably done in the housekeeping.

ZP03

```
procedure pageout(string,type,resultline,array,start);
          value type,resultline,start;
          string string;
          integer type,resultline,start;
          integer array array;
          begin   own integer line,page;
                  switch sss:=L1,L2,L3;
                  if type=2 then goto L1;
                  if type=1 then
                        begin if line+resultline>50 then
                                L1:begin for line:=line+1 while line<62 do print ££1??;
                                         if type=2 then goto L3;
                                         goto L2
                                   end
                                         else line:=line+resultline
                        end;
                  if type=0 then
                     L3:begin page:=1;
                           L2:print ££1??;
                              if start ‡ 0 then outstring(array,start);
                              print ££lt8?page?,sameline,digits(3),page,££12??,string;
                              page:=page+1;
                              line:=if type=1 then resultline else 0
                        end
          end procedure pageout;
```