CS/AM - 7701

COO-2280-29

TWO-STEP, USER'S GUIDE

Solution of a Linear System of Equations
Using the Exact Two - Step Method

MASTER

Ruth Kobbe, Erwin H. Bareiss, and Arlyn Antolak
Northwestern University

February 1977

NOTICE
This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research and Development Administration, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Prepared For

THE U.S. ENERGY RESEARCH AND DEVELOPMENT ADMINISTRATION
UNDER CONTRACT NO. E(11-1)-2280

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

ABSTRACT

The report is a guide for the use of the CDC-6400 Computer Program TWO-STEP for the exact solution of linear matrix equations over the integral domain. The rational result is given as output and also converted into decimal form to twenty decimal places.

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research and Development Administration, nor any of their employees, nor any of their contractors subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product or process disclosed or represents that its use would not infringe privately owned rights.

TWO-STEP

Solution of a Linear System of Equations Using the Exact Two-Step Method

Keywords: Gaussian elimination, multi-precision, inverses,
simultaneous linear equations

<u>Contents</u>	<u>Page</u>
Introduction.....	1
Purpose and General Description.....	2
Usage.....	2
Program Construction and Limitations.....	4
Description of Input.....	4
Description of Output.....	5
Sample Input Data.....	5
Sample Output - 5 x 5 Hilbert Segment.....	6
- 16 x 16 Hilbert Segment.....	11
- Singular Matrix.....	16
- Arbitrary Matrix.....	17
TWO-STEP Computer Code.....	19
Appendix.....	25

Introduction

This computer program is an implementation of the multistep integer-preserving Gaussian elimination method for solving systems of linear equations exactly if the entries in the augmented matrix are integers. Background information is given in the following references:

E. H. Bareiss, "Sylvester's Identity and Multi-Step Integer-Preserving Gaussian Elimination", Math. Comp. 22 (565-578) 1968

E. H. Bareiss, "Computational Solutions of Matrix Problems over an Integral Domain", Jour. Inst. Math. Appl. 10 (68-104) 1973

This report is a revision of C00-2280-24 and applies to the correspondingly revised computer code.

Purpose and General Description

TWO-STEP solves the linear system of equations $Ax = B$ in which both A and B contain only integer values. The computational method employs an integer-preserving Gaussian elimination method to triangularize A in the system $[A : B]$ and performs the back substitution using only one non-integer divide. TWO-STEP makes use of the integer multiprecision routines IMPP (see Appendix).

Usage

If the matrices A and B do not contain integer values then they must be scaled to integers. As a simple example, consider the problem of inverting the 3x3 segment of the Hilbert Matrix:

$$A = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix} ; \quad B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

After scaling to integers:

$$A_1 = \begin{bmatrix} 60 & 30 & 20 \\ 30 & 20 & 15 \\ 20 & 15 & 12 \end{bmatrix} ; \quad B_1 = \begin{bmatrix} 60 & 0 & 0 \\ 0 & 60 & 0 \\ 0 & 0 & 60 \end{bmatrix}$$

Note that the scaling in this case was accomplished by multiplying each element of both matrices by 60. Now, after triangularization, $[A_1 : B_1]$ is

$$\left[\begin{array}{ccc|ccc} 60 & 30 & 20 & 60 & 0 & 0 \\ 0 & 300 & 300 & -1800 & 3600 & 0 \\ 0 & 0 & 100 & 3000 & -18000 & 18000 \end{array} \right]$$

Results after back substitution are:

$$A_1^{-1} B_1 = \begin{bmatrix} 9 & -36 & 30 \\ -36 & 192 & -180 \\ 30 & -180 & -180 \end{bmatrix} = A^{-1}$$

Matrices A and B could also be scaled by premultiplication with a diagonal matrix of the least common denominator of the corresponding row elements. For example, using

$$D_1 = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 12 & 0 \\ 0 & 0 & 60 \end{bmatrix}$$

yields for A_2 and B_2 ,

$$A_2 = \begin{bmatrix} 6 & 3 & 2 \\ 6 & 4 & 3 \\ 20 & 15 & 12 \end{bmatrix} ; \quad B_2 = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 12 & 0 \\ 0 & 0 & 60 \end{bmatrix}$$

The entries can also be reduced further by postmultiplication of A by a diagonal matrix D_2^{-1} where the elements of D_2 are the greatest common divisors of the corresponding columns. Hence for $Ax = B$, we have $(D_1 A D_2^{-1}) (D_2 x) = D_1 B$. Defining $A_3 = D_1 A D_2^{-1}$, $x_3 = D_2 x$, and $B_3 = D_1 B$, the equation can also be written $A_3 x_3 = B_3$. Letting

$$D_2 = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

A_3 and B_3 become,

$$A_3 = \begin{pmatrix} 3 & 3 & 2 \\ 3 & 4 & 3 \\ 10 & 15 & 12 \end{pmatrix}; \quad B_3 = \begin{pmatrix} 6 & 0 & 0 \\ 0 & 12 & 0 \\ 0 & 0 & 60 \end{pmatrix}$$

and $x = D_2^{-1} x_3$. These are but a few of the ways to scale A and B to integers, and, depending on the problem, the matrices can be put in the most convenient and useful form.

Program Construction and Limitations

TWO-STEP consists of a main program and a subroutine. The main program:

- reads the input data in character form,
- calls the subprogram,
- prints the results in character form.

The subroutine:

- converts the numbers in A and B from character form to multiprecision integer form,
- reduces A in [A : B] to upper triangular form, oversteoring the original values of A,
- performs the back substitution storing the integer form solution in B and the corresponding floating point solution in another work matrix,
- both solutions are printed in the main program,
- converts the data in A and B from multiprecision to character form.

Program limitations are:

- for A(n,n), B(m,n) : $3 \leq n \leq 24$; $0 \leq m \leq 24$,
- no more than 160 digits per input number; for intermediate calculations, the limit is 224 digits per number,
- Hadamard's inequality may be used to estimate the number of digits needed for a matrix:

maximum number of digits $\leq$ number of digits in

$$\left(\prod_{i=1}^n \sum_{k=1}^n a_{ik}^2 \right)^{\frac{1}{2}}$$

Description of Input

Card 1: columns 1-5: size of A matrix (n).
columns 6-10: numbers of vectors in right-hand side (m).
columns 11-15: number of 10-digit words per number for input (nwords).

All numbers on card 1 must be right-adjusted integers.

Card 2: Variable Format Card

Since the data is read in character format, the "R" - type format specification should be used. The same format is used both for the matrix A data and the vector data (the B matrix or right-hand sides). For example, assuming that the data is punched with 6 numbers per card, 12 digits per number, then the format is (6(R2,R10)) and nwords on card 1 is 2. Blanks are read as zeros.

Matrix data:

The data is read in row-order.
Data for each row must begin on a new card.

Sample Output - 5x5 Hilbert Segment

The 5x5 Hilbert segment is

$$H_5 = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{bmatrix}$$

We want to solve $H_5 X = I$ (i.e. the inverse of H_5). H_5 was scaled to integer values by premultiplying the matrix by the least common multiple of all the denominators, which is 2520, to obtain the following matrix, now called the original matrix.

ORIGINAL MATRIX. N = 5

ROW	1	2520	1260	840	630	504
ROW	2	1260	840	630	504	420
ROW	3	840	630	504	420	360
ROW	4	630	504	420	360	315
ROW	5	504	420	360	315	280

Right hand side column vectors for the 5x5 identity matrix are listed columnwise and displayed as follows:

RIGHT HAND COLUMN VECTORS.		NUMBER OF VECTORS M =	5
COL 1	2520		
COL 2	2520		
COL 3		2520	
COL 4			2520
COL 5			2520

To solve for the inverse of H_5 directly, the identity matrix was also scaled by 2520. Input to TWO-STEP consisted of the scaled Hilbert segment and the above matrix. Here the zeros are suppressed in the output.

The output for the upper triangularized matrix for the 5x5 transformed Hilbert segment appears as follows:

TIME FOR MATRIX CALC = .52 SECONDS.

UPPER TRIANGULARIZED MATRIX

ROW	1	COLUMN	1	2520
ROW	1	COLUMN	2	1260
ROW	1	COLUMN	3	840
ROW	1	COLUMN	4	630
ROW	1	COLUMN	5	504
ROW	2	COLUMN	2	529200
ROW	2	COLUMN	3	529200
ROW	2	COLUMN	4	476280
ROW	2	COLUMN	5	423360
ROW	3	COLUMN	3	7408800
ROW	3	COLUMN	4	11113200
ROW	3	COLUMN	5	12700800
ROW	4	COLUMN	4	6667920
ROW	4	COLUMN	5	13335840
ROW	5	COLUMN	5	381024

The final solution vectors for the 5x5 Hilbert inverse are represented on the following two pages.

VECTOR RESULTS IN INTEGER FORM WITH $X(I,J) = Y(I,J) / \text{DETERMINANT}$

DETERMINANT =

381024

THE DOUBLE PRECISION EQUIVALENT OF THE DETERMINANT IS

3.410240000000000000000000+05

	I	J	Y(I,J) =	X(I,J) =
	-	-		
VECTOR 1 ROW 1	1	1	9525600	2.50000000000000000000+01
VECTOR 1 ROW 2	1	2	-114307200	-3.00000000000000000000+02
VECTOR 1 ROW 3	1	3	400075200	1.05000000000000000000+03
VECTOR 1 ROW 4	1	4	-533433600	-1.40000000000000000000+03
VECTOR 1 ROW 5	1	5	240045120	6.30000000000000000000+02
VECTOR 2 ROW 1	2	1	-114307200	-3.00000000000000000000+02
VECTOR 2 ROW 2	2	2	1828915200	4.80000000000000000000+03
VECTOR 2 ROW 3	2	3	-7201353600	-1.89000000000000000000+04
VECTOR 2 ROW 4	2	4	10241925120	2.68800000000000000000+04
VECTOR 2 ROW 5	2	5	-4800902400	-1.26000000000000000000+04

VECTOR	3 ROW	1	400075200	1.05000000000000000000+03
VECTOR	3 ROW	2	-7201353600	-1.89000000000000000000+04
VECTOR	3 ROW	3	30245685120	7.93800000000000000000+04
VECTOR	3 ROW	4	-44808422400	-1.17600000000000000000+05
VECTOR	3 ROW	5	21604060800	5.67000000000000000000+04
VECTOR	4 ROW	1	-533433600	-1.40000000000000000000+03
VECTOR	4 ROW	2	10241925120	2.68800000000000000000+04
VECTOR	4 ROW	3	-44808422400	-1.17600000000000000000+05
VECTOR	4 ROW	4	68279500800	1.79200000000000000000+05
VECTOR	4 ROW	5	-33606316800	-8.82000000000000000000+04
VECTOR	5 ROW	1	240045120	6.30000000000000000000+02
VECTOR	5 ROW	2	-4800902400	-1.26000000000000000000+04
VECTOR	5 ROW	3	21604060800	5.67000000000000000000+04
VECTOR	5 ROW	4	-33606316800	-8.82000000000000000000+04
VECTOR	5 ROW	5	16803158400	4.41000000000000000000+04

Sample Output - 16x16 Hilbert Segment

The computer printout on the next three pages is the output for a scaled 16x16 Hilbert segment.

ORIGINAL MATRIX. N = 16

ROW 1	720720 120120 65520 45045	360360 102960 60060	240240 90090 55440	180180 80080 51480	144144 72072 48048
ROW 2	6126120 1750320 1021020 720720	4084080 1531530 942480	3063060 1361360 875160	2450448 1225224 916816	2042040 1113840 765765
ROW 3	4084080 1531530 942480 680680	3063060 1361360 875160	2450448 1225224 816816	2042040 1113840 765765	1750320 1021020 720720
ROW 4	58198140 25865840 16628040 12252240	46558512 23279256 15519504	38798760 21162960 14549535	33256080 19399380 13693680	29099070 17907120 12932920
ROW 5	46558512 23279256 15519504 11639628	38798760 21162960 14549535	33256080 19399380 13693680	29099070 17907120 12932920	25865840 16628040 12252240

ROW 6	38799760 21162960 14549535 11085360	33256080 19399380 13693680	29099070 17907120 12932920	25865840 16628040 12252240	23279256 15519504 11639628
ROW 7	33256080 19399380 13693680 10581480	29099070 17907120 12932920	25865840 16628040 12252240	23279256 15519504 11639628	21162960 14549535 11085360
ROW 8	669278610 411863760 297457160 232792560	594914320 382444920 281801520	535422888 356948592 267711444	486748080 334639305 254963280	446185740 314954640 243374040
ROW 9	594914320 382444920 281801520 223092870	535422888 356948592 267711444	486748080 334639305 254963280	446185740 314954640 243374040	411863760 297457160 232792560
ROW 10	2677114440 1784742960 1338557220 1070845776	2433740400 1673196525 1274816400	2230928700 1574773200 1216870200	2059318800 1487285800 1163962900	1912224600 1409007600 1115464350
ROW 11	2433740400 1673196525 1274816400 1029659400	2230928700 1574773200 1216870200	2059318800 1487285800 1163962800	1912224600 1409007600 1115464350	1784742960 1338557220 1070845776

ROW 12

6692785100
4724319600
3650610600
2974571600

6177956400
4461857400
3491888400

5736673800
4227022800
3346393050

5354228880
4015671660
3212537328

5019589575
3824449200
3088978200

ROW 13

6177956400
4461857400
3491888400
2868336900

5736673800
4227022800
3346393050

5354228880
4015671660
3212537328

5019589575
3824449200
3088978200

4724319600
3650610600
2974571600

ROW 14

166363540200
122583661200
97045398450
80313433200

155272637520
116454478140
93163582512

145568097675
110909026800
89580367800

137005268400
105867707400
86262576400

129393864600
101264763600
83181770100

ROW 15

155272637520
116454478140
93163582512
77636318760

145568097675
110909026800
89580367800

137005268400
105867707400
86262576400

129393864600
101264763600
83181770100

122583661200
97045398450
80313433200

ROW 16

4512611027925
3438179830800
2776991401800
2329099562800

4247163320400
3281898929400
2674139868400

4011209802600
3139207671600
2578634873100

3800093497200
3008407351950
2489716429200

3610088822340
2888071057872
2406725811560

The next two pages give part of the first column and the entire last column for the 16x16 Hilbert inverse solution vectors.

VECTOR RESULTS IN INTEGER FORM WITH $X(I,J) = Y(I,J) / \text{DETERMINANT}$

DETERMINANT =

2340579145905

THE DOUBLE PRECISION EQUIVALENT OF THE DETERMINANT IS

2.34057914590500000000D+12

	I	J	Y(I,J) =	X(I,J) =
	-	-		
VECTOR	1 ROW	1	599144261351680	2.560000000000000000D+02
VECTOR	1 ROW	2	-76396503322339200	-3.264000000000000000D+04
VECTOR	1 ROW	3	3208653139538246400	1.370880000000000000D+06
VECTOR	1 ROW	4	-66044777122162238400	-2.821728000000000000D+07
VECTOR	1 ROW	5	792537325465946960800	3.386073600000000000D+08
VECTOR	1 ROW	6	-6102537406087790828160	-2.607276672000000000D+09
VECTOR	1 ROW	7	31965672127126523385600	1.365716352000000000D+10
VECTOR	1 ROW	8	-118158823755628399343200	-5.048272944000000000D+10
VECTOR	1 ROW	9	315090196681675730515200	1.346206118400000000D+11
VECTOR	1 ROW	10	-612675382436591698224000	-2.617623008000000000D+11
VECTOR	1 ROW	11	868885087819166408390400	3.712265356800000000D+11

Sample Output - Singular Matrix

If the input matrix is singular, the output is as follows:

ORIGINAL MATRIX. N = 4

ROW	1	22	33	44	55
ROW	2	66	77	88	99
ROW	3	11	33	55	77
ROW	4	22	44	66	88

RIGHT HAND COLUMN VECTORS. NUMBER OF VECTORS M = 1

COL	1	11	22	33	44
-----	---	----	----	----	----

MATRIX IS SINGULAR

TOTAL TIME = .13 SECONDS.

Sample Output - Arbitrary Matrix

A nonsymmetric system of linear equations was solved using TWO-STEP whose elements were picked at random. Here the 4x4 input original matrix and column vector are given, as well as the matrix after it was upper triangularized.

ORIGINAL MATRIX. N = 4

ROW 1	3	9	6	4
ROW 2	8	2	9	3
ROW 3	1	0	5	7
ROW 4	6	6	5	5

RIGHT HAND COLUMN VECTORS. NUMBER OF VECTORS M = 1

COL 1	89	22	-25	37
-------	----	----	-----	----

TIME FOR MATRIX CALC = .12 SECONDS.

UPPER TRIANGULARIZED MATRIX

ROW 1 COLUMN 1	3
ROW 1 COLUMN 2	9
ROW 1 COLUMN 3	6
ROW 1 COLUMN 4	4
ROW 2 COLUMN 2	-66
ROW 2 COLUMN 3	-21
ROW 2 COLUMN 4	-23
ROW 3 COLUMN 3	-261
ROW 3 COLUMN 4	-443
ROW 4 COLUMN 4	-1718

-18-

- 1718

-1.718000000000000000000000+03

TOTAL TIME = .31 SECONDS.

TWO-STEP Computer Code

The listing of the program TWO-STEP is given on the next six pages, and is:

[illegible]

```

000301      C**** CALL SUBROUTINE AND CHECK TIME.
000302      CALL DIAG
000303      IF(IC .EQ. 2) GO TO 260
000304      IF(IC .EQ. 3) GO TO 261
000305      CALL SECOND(T2)
000306      TIME = T2-T1
000307      PRINT 191
000308      PRINT 180,TIME
000309      180 FORMAT(//'* TIME FOR MATRIX CALC = *,F8.2,* SECONDS.*')
000310      C**** PRINT DIAGONALIZED MATRIX.
000311      PRINT 185
000312      185 FORMAT(//'* UPPER TRIANGULARIZED MATRIX*//)
000313      DO 190 I=1,N
000314      DO 190 J=I,N
000315      PRINT 200 ,I,J,(A(NW,I,J),NW=1,NWD)
000316      PRINT 300
000317      300 FORMAT(1X,/)
000318      190 CONTINUE
000319      200 FORMAT(* ROW *,I3,* COLUMN *,I3,1X,8A10/(20X,8A10))
000320      IF(M.EQ.0) GO TO 240
000321      PRINT 191
000322      191 FORMAT(1H1)
000323      PRINT 195
000324      195 FORMAT(//'* VECTOR RESULTS IN INTEGER FORM WITH X(I,J) = Y(I,J) / D
000325      1ETERMINANT*//)
000326      PRINT 300
000327      PRINT 196,(A(NF,N,N),NF=1,NWD)
000328      196 FORMAT(1X,*DETERMINANT = *,5X,8A10/(20X,8A10))
000329      PRINT 198
000330      198 FORMAT(1X,/)
000331      PRINT 197,X
000332      197 FORMAT(1X,*THE DOUBLE PRECISION EQUIVALENT OF THE DETERMINANT IS*,
000333      119X,D27.20)
000334      PRINT 199
000335      199 FORMAT(////9X,*I*,7X,*J*,74X,*Y(I,J) =*,25X,*X(I,J) =*/9X,*--*,7X,*
000336      1--*,2X,80(*.*),6X,27(*.*),/)
000337      DO 230 I=1,M
000338      DO 230 J=1,N
000339      PRINT 210,I,J,(B(NW,J,I),NW=1,NWD)
000340      210 FORMAT(* VECTOR *,I3,* ROW *,I3,1X,8A10/(20X,8A10))
000341      PRINT 220,REMAIN(J,I)
000342      220 FORMAT(**,105X,D27.20)
000343      PRINT 300
000344      230 CONTINUE
000345      240 CALL SECOND(TT2)
000346      TIME = TT2-TT1
000347      PRINT 250,TIME
000348      250 FORMAT(//'* TOTAL TIME =*,F8.2,* SECONDS.*')
000349      GO TO 10
000350      260 PRINT 270
000351      270 FORMAT(//'* MATRIX IS SINGULAR*//)
000352      GO TO 240
000353      261 PRINT 271
000354      271 FORMAT(//'* DETERMINANT IS ZERO, BACK SUBSTITUTION NOT POSSIBLE*//)
000355      GO TO 240
000356      END

```



```

SUBROUTINE DIAG
C
C.... TWO-STEP INTEGER-PRESERVING GAUSSIAN ELIMINATION -- DIAGONAL FORM
C
000002 DOUBLE REMAIN,X,Y
000002 COMMON A(16,24,24),B(16,24,24),NWD,REMAIN(24,24),N,M,IC
000002 COMMON X
000002 DIMENSION RP(32,24)
000002 INTEGER A,B,C0,C1,C2,A11,A22,AKK,A1K,AK1,BT,TEMP,TEMP2,SWAP
000002 DIMENSION C0(16),C1(16),C2(16),A11(16),A22(32),AKK(16),AK1(16),
141K(16),BT(32),TEMP(32),ITEMP(32),TEMP2(32),MAR1(64),MAR2(64)
C
000002 RTEST =0.
000003 NMIN1=N-1
000005 NWD2= 2*NWD
000007 IC=1
000010 K=0
000011 DO 1 I=1,N
000012 DO 1 J=1,N
000013 CALL MOVE(NWD,NWD,A(1,I,J),C0)
000021 CALL MREAD(NWD,NWD,C0,A(1,I,J))
000030 1 CONTINUE
000035 DO 2 I=1,N
000036 DO 2 J=1,M
000037 CALL MOVE(NWD,NWD,B(1,I,J),C0)
000045 CALL PREAD(NWD,NWD,C0,B(1,I,J))
000054 2 CONTINUE
C
000061 C0=1
CALL MOVE(1,NWD,1,C0)
C
C 10 A22=C0
10 CALL MOVE(NWD,NWD2,C0,A22)
000064 MN=M+N
000067 IF (K.GE.NMIN1) GO TO 200
000071 K=K+2
000074 K1=K+1
000075 KMIN1=K1-2
C
C.... INTERCHANGE ROWS IF NECESSARY TO ASSURE NON-ZERO DIAGONAL ELEMENT
C
000100 DO 40 I=KMIN1,N
C
C A11=A(I,K-1)
C IF (A11.EQ.0) GO TO 40
C CALL MOVE(NWD,NWD,A(1,I,K-1),A11)
000101 DO 21 KK=1,NWD
000107 IF(A11(KK).NE.0) GO TO 22
000111 21 CONTINUE
000112 GO TO 40
000115 22 IF (I.EQ.KMIN1) GO TO 50
000117 IC=-IC
000120 DO 20 J=1,N
C
C SWAP=A(K-1,J)
C DO 20 KK=1,NWD
C SWAP = A(KK,K-1,J)
C A(K-1,J)=A(I,J)
C A(KK,K-1,J) = A(KK,I,J)
000122 20 A(I,J)=SWAP
000123 C 20
000130

```

F 2
F 3
F 4

F 9
F 10
F 11
F 12

F 13
F 14
F 15

F 16
F 17
F 18
F 19
F 20
F 21
F 22
F 23
F 24
F 25

F 26
F 27
F 28
F 29

F 30
F 31

000137	20	A(KK,I,J) = SWAP	
000146		IF (M.EQ.0) GO TO 53	
000147		00 30 J=1,M	F 32
	C	SWAP=B(K-1,J)	F 33
000151		GO 30 KK=1,NWO	F 34
000152		SWAP = B(KK,K-1,J)	
	C	B(K-1,J)=B(I,J)	
000157		B(KK,K-1,J) = B(KK,I,J)	F 35
	C 30	B(I,J)=SWAP	
000166	30	B(KK,I,J) = SWAP	F 36
000173		GO TO 93	
000176	40	CONTINUE	F 37
000201		GO TO 90	F 38
	C		F 39
	C 50	A1K=A(K-1,K)	F 40
000201	50	CALL MOVE(NWO,NWO,A(1,K-1,K),A1K)	F 41
	C		F 42
	C		F 42
	C....	INTERCHANGE ROWS IF NECESSARY TO ASSURE NON-ZERO PRINCIPAL MINOR	F 43
	C		F 44
000207		00 80 I=K,N	F 45
	C	A1I=A(I,K-1)	F 46
	C	AKK=A(I,K)	F 47
000211		CALL MOVE(NWO,NWO,A(1,I,K-1),A1I)	
000217		CALL MOVE(NWO,NWO,A(1,I,K),AKK)	
	C	BT=AKK*A1I-A1I*A1K	F 48
000226		CALL NMULT(NWO, AKK,A1I,MAR1)	
000231		CALL NMULT(NWO,A1I,A1K,MAR2)	
000234		CALL NSUB(NWO2,MAR1,MAR2,BT)	
	C	IF (BT.EQ.0) GO TO 80	F 49
000237		00 61 KK=1,NWO2	
000241		IF(BT(KK).NE.0) GO TO 62	
000242	61	CONTINUE	
000245		GO TO 80	
000245	62	CONTINUE	
	C	C0=BT/A22	F 50
000245		CALL NOIV(NWO2,BT,A22,TEMP2,TEMP)	
000251		CALL MOVE(NWO2,NWO,TEMP2,C0)	
000254		IF (I.EQ.K) GO TO 100	F 51
000256		IC=-IC	F 52
000257		00 60 J=1,N	F 53
	C	SWAP=A(K,J)	F 54
000261		00 63 KK=1,NWO	
000262		SWAP = A(KK,K,J)	
	C	A(K,J)=A(I,J)	F 55
000267		A(KK,K,J) = A(KK,I,J)	
	C 60	A(I,J)=SWAP	F 56
000276	60	A(KK,I,J) = SWAP	
000305		IF (M.EQ.0) GO TO 100	F 57
000306		00 73 J=1,M	F 58
	C	SWAP=B(K,J)	F 59
000310		GO 73 KK=1,NWO	
000311		SWAP = B(KK,K,J)	
	C	B(K,J)=B(I,J)	F 60
000316		B(KK,K,J) = B(KK,I,J)	
	C 70	B(I,J)=SWAP	F 61
000325	70	B(KK,I,J) = SWAP	
000334		GO TO 100	F 62

```

000335      80  CONTINUE
C
C.... MATRIX IS SINGULAR -- ERROR RETURN
C
000340      90  IC=2
000341      RETURN
C
C.... TWO-STEP ALGORITHM FOR REDUCTION OF NON-PIVOT ROWS
C
000342      100 I=K
000344      110 I=I+1
000346      IF (I=N) 120,120,160
C 120  C1=(A1K*A(I,K-1)-A11*A(I,K))/A22
000350      120  CALL NMULT(NWO,A1K,A(1,I,K-1),MAR1)
000357      CALL NMULT(NWO,A11,A(1,I,K),MAR2)
000366      CALL NSUB(NWO2,MAR1,MAR2,TEMP)
000371      CALL NOIV(NWO2,TEMP,A22,TEMP2,MAR1)
000375      CALL MOVE(NWO2,NWO,TEMP2,C1)
C      C2=(AKK*A(I,K-1)-AK1*A(I,K))/A22
000400      CALL NMULT(NWO,AKK,A(1,I,K-1),MAR1)
000407      CALL NMULT(NWO,AK1,A(1,I,K),MAR2)
000416      CALL NSUB(NWO2,MAR1,MAR2,TEMP)
000421      CALL NOIV(NWO2,TEMP,A22,TEMP2,MAR1)
000425      CALL MOVE(NWO2,NWO,TEMP2,C2)
000430      IF (K1.GT.N) GO TO 140
000434      DO 130 J=K1,N
C 130  A(I,J)=(C0*A(I,J)+C1*A(K,J)-C2*A(K-1,J))/A22
000435      CALL NMULT(NWO,C0,A(1,I,J),MAR1)
000443      CALL NMULT(NWO,C1,A(1,K,J),MAR2)
000452      CALL NADD(NWO2,MAR1,MAR2,TEMP2)
000455      CALL NMULT(NWO,C2,A(1,K-1,J),MAR1)
000464      CALL NSUB(NWO2,TEMP2,MAR1,TEMP)
000467      CALL NOIV(NWO2,TEMP,A22,TEMP2,MAR1)
000473      CALL MOVE(NWO2,NWO,TEMP2,A(1,I,J))
000502      130  CONTINUE
000505      140  IF (M.EQ.0) GO TO 110
000506      DO 150 J=1,M
C 150  B(I,J)=(C0*B(I,J)+C1*B(K,J)-C2*B(K-1,J))/A22
000510      CALL NMULT(NWO,C0,B(1,I,J),MAR1)
000516      CALL NMULT(NWO,C1,B(1,K,J),MAR2)
000525      CALL NADD(NWO2,MAR1,MAR2,TEMP2)
000530      CALL NMULT(NWO,C2,B(1,K-1,J),MAR1)
000537      CALL NSUB(NWO2,TEMP2,MAR1,TEMP)
000542      CALL NOIV(NWO2,TEMP,A22,TEMP2,MAR1)
000546      CALL MOVE(NWO2,NWO,TEMP2,B(1,I,J))
000555      150  CONTINUE
000560      GO TO 110
C
C.... ONE-STEP ALGORITHM FOR REDUCTION OF TWO PIVOT ROWS
C
C 160  A(K,K)=C0
000560      160  CALL MOVE(NWO,NWO,C0,A(1,K,K))
000566      IF (K1.GT.N) GO TO 100
000572      DO 170 J=K1,N
C 170  A(K,J)=(A11*A(K,J)-AK1*A(K-1,J))/A22
000573      CALL NMULT(NWO,A11,A(1,K,J),MAR1)
000601      CALL NMULT(NWO,AK1,A(1,K-1,J),MAR2)
000610      CALL NSUB(NWO2,MAR1,MAR2,TEMP)

```

F 63
F 64
F 65
F 66
F 67
F 68
F 69
F 70
F 71
F 72
F 73
F 74
F 75

F 76

F 77
F 78
F 79

F 80
F 81
F 82

F 83
F 84
F 85
F 86
F 87

F 88
F 89
F 90

```

000613      CALL NDIV(NWD2,TEMP,A22,TEMP2,MAR1)
000617      CALL MOVE(NWD2,NWD,TEMP2,A(1,K,J) )
000626      170 CONTINUE
000631      180 IF (M.EQ.0) GO TO 10
000632      DO 190 J=1,M
C 190      B(K,J)=(A11*B(K,J)-AK1*B(K-1,J))/A22
000634      CALL NMULT(NWD,A11,B(1,K,J),MAR1)
000642      CALL NMULT(NWD,AK1,B(1,K-1,J),MAR2)
000651      CALL NSUB(NWD2,MAR1,MAR2,TEMP)
000654      CALL NDIV(NWD2,TEMP,A22,TEMP2,MAR1)
000660      CALL MOVE(NWD2,NWD,TEMP2,B(1,K,J))
000667      190 CONTINUE
000672      GO TO 10
000672      200 IF(M.EQ.0) GO TO 1000
000673      CALL MOVE(NWD,NWD2,A(1,N,N),TEMP2)
000701      CALL DOUBL(NWD2,TEMP2,X)
000704      CALL MOVE(NWD,NWD2,A(1,N,N),TEMP2)
000712      IF (X.EQ. PTEST) GO TO 220
000714      GO TO 221
000715      220 IC = 3
000716      RETURN
000717      221 DO 250 K= 1,M
000721      CALL MOVE(NWD,NWD2,B(1,N,K),RP(1,N) )
000730      DO 240 II=1,NMIN1
000732      I = N-II
000734      IP1 = I+1
000736      CALL MOVE(1,NWD2,0,ITEMP)
000741      DO 230 J=IP1,N
000743      CALL MOVE(NWD,NWD2,A(1,I,J),MAR2)
000751      CALL NMULT(NWD2,MAR2,RP(1,J),MAR1)
000756      CALL MOVE(2*NWD2,NWD2,MAR1,MAR2)
000763      CALL NAOD(NWD2,MAR2,ITEMP,TEMP)
000766      CALL MOVE(NWD2,NWD2,TEMP,ITEMP)
000771      230 CONTINUE
000774      CALL NMULT(NWD,A(1,N,N),B(1,I,K),MAR1 )
001005      CALL NSUB(NWD2,MAR1,ITEMP,TEMP)
001010      CALL MOVE(NWD,NWD2,A(1,I,I),MAR1)
001016      CALL NDIV(NWD2,TEMP,MAR1,RP(1,I),MAR2 )
001023      240 CONTINUE
001026      DO 245 I=1,N
001027      CALL MOVE(NWD2,NWD,RP(1,I),B(1,I,K))
001036      CALL NDIV(NWD2,RP(1,I),TEMP2,TEMP,MAR1)
001043      CALL DOUBL (NWD2,MAR1,Y)
001046      CALL DOUBL (NWD2,TEMP,Z)
001051      REMAIN (I,K) = Y/X + Z
001074      245 CONTINUE
001076      250 CONTINUE
001101      1000 IF(IC.GT.0) GO TO 1010
001104      CALL MOVE(NWD, A(1,N,N), A(1,N,N) )
001112      1010 DO 1040 I=1,N
001114      DO 1020 J=1,N
001115      CALL CONVTI(NWD,A(1,I,J),TEMP)
001122      DO 1015 K=1,NWD
001124      1015 A(K,I,J) = TEMP(NWD+K)
001136      1020 CONTINUE
001141      DO 1030 J=1,M
001142      CALL CONVTI(NWD,B(1,I,J),TEMP)
001147      DO 1025 K=1,NWD
001151      1025 B(K,I,J) = TEMP(NWD+K)
001163      1030 CONTINUE
001166      1040 CONTINUE
001170      RETURN
001171      END

```

F 91
F 92
F 93

F 94

F 106

APPENDIX

Integer Multi-Precision Package

R. Kobbe

Table of Contents

	Page
Purpose	1
Form of Multi-precision Integers.....	1
Usage.....	2
IMPP Operations.....	2
MOVE.....	2
MOVEC.....	3
MREAD.....	3
CONVTI.....	4
SINGLE.....	4
DOUBL.....	4
SINGLF.....	4
DOUBLF.....	4
NADD.....	5
NSUB.....	5
NMULT.....	5
NMULTH.....	5
NDIV.....	6
NSQRT.....	6
NSQRTW.....	6
Error Indications.....	7
Examples.....	7

Purpose

To allow the user to read, write, and perform arithmetic operations with integer data containing an arbitrarily large number of digits.

Form of Multi-precision Numbers

A multi-precision integer (MPI) is stored in an array, say A. The most significant part of the number is stored in the first word of the array, A(1); the least significant part of the number is stored in the last word of the array, A(N). The sign of the number is indicated by the second bit from the left of A(1); 0=plus, 1=minus. The lower 45 bits of each word contain integer data, allowing approximately 13 decimal digits to be stored in each word.

Usage

The user must specify the precision for each operation. It is convenient for the user to maintain the same precision for a number throughout a program even if, during part of the operations, the number requires far less precision. To allow the user to do this without a great loss of time, the multi-precision routines test for zero words in multi-precision numbers before doing time-consuming operations of multiplication, division, or conversion.

IMPP Operations

Move operations:

MOVE	Copies an MPI to another
MOVEC	Copies an MPI but complements the sign

Conversion to/from printable characters:

MREAD	Creates an MPI from character digits
CONVTI	Converts an MPI to character digits

Conversion to real numbers:

SINGLE	Produces a single precision real number
DOUBL	Produces a double precision real number
SINGLF	Functional form of SINGLE
DOUBLF	Functional form of DOUBL

Arithmetic operations:

NADD	Adds two MPI's
NSUB	Subtracts one MPI from another
NMULT	Multiplies two MPI's together
NMULTH	Multiplies two MPI's together
NDIV	Divides one MPI by another, producing an integer quotient and a remainder
NSQRT	Produces the (integral) square root of an MPI
NSQRTW	Provides the (integral) square root of an MPI, using user-provided working storage

Detailed descriptions of each operation follow. A note about terminology: LENX is used to indicate the length of an array called X. Thus LENA is the length of array A. All MPI names represent integer entities, regardless of implicit FORTRAN types.

MOVE - Copy one MPI to another

Syntax:

CALL MOVE (LENA, LENB, A, B)

Single-precision equivalent:

B = A

Notes:

Moving is done from right to left. MIN (LENA,LENB) words will be moved. If LENA is less than LENB (A is smaller than B), then surplus words in B are zero-filled. If LENB is smaller than LENA, only LENB words are moved. A and B may be the same array only if LENA is less than or equal to LENB or LENA is greater than or equal to twice LENB; otherwise overlap may occur in moving and the contents of the array may not be what was desired.

Move may be used to convert single-precision integers to MPI-S for positive values less than or equal to $2^{45}-1$.

Examples:

Create a 10-precision 2:

```
DIMENSION A(10)
CALL MOVE (1, 10, 2, A)
```

Create a 5-precision negative 2:

```
CALL MOVE (1, 5, 200000000000000000002B, A)
```

MOVE is also a quick way to set a large array to zero:

```
DIMENSION X(100,100)
CALL MOVE (1, 10000, 0, X)
```

MOVEC - Copy an MPI with sign changed

Syntax:

```
CALL MOVEC (LEN, A, B)
```

Single-precision equivalent:

B = - A

Notes:

A and B are the same length. If they are the same array, no actual moving is performed, just the sign change. Otherwise, movement is from right to left.

MREAD - Convert character digits to an MPI

Syntax:

```
CALL MREAD (LENC, LENMPI, C, MPI)
```

Single-precision equivalent:

DECODE (memory to memory read)

Notes:

C is an array containing display code (printable characters), 10 characters per word. The characters are assumed to be right-justified in C. Allowed in C are:

character digits: 0123456789

blank: converted to zero

sign: + or -. The sign must come before any character digits or it is treated as an error

00 octal: converted as zero

A word which is integer zero or ten blanks is considered to be ten character zeros. Other characters in C are considered to be errors, and are converted to zero. An error message will display the offending character. One word of display code requires .77 words of MPI. Characters may be stored in C via an assignment statement or they may be read from some input medium in A10 format.

CONVTI - Convert an MPI to printable form

Syntax:

```
CALL CONVTI (LENMPI, MPI, DPC)
```

Single-precision equivalent:

```
ENCODE (memory to memory print)
```

Notes:

DPC, which must be twice the size of the MPI (2*LENMPI), will contain display code characters corresponding to the MPI. DPC may be printed using multiple A10 format specifications.

SINGLF -

SINGLE - Produce a single-precision real equivalent of an MPI

Syntax:

```
REAL REALA  
CALL SINGLE (LENA, A, REALA)  
REALA = SINGLF (LENA, A)
```

Notes:

This operation should not be used when the MPI has more than 300 digits.

DOUBLF -

DOUBL - Produce a double-precision real equivalent of an MPI

Syntax:

```
DOUBLE PRECISION DOUBLF, DBLA  
CALL DOUBL (LENA, A, DBLA)  
DBLA = DOUBLF (LENA, A)
```

Notes:

When using the functional form of this operation, the function name and the result variable must be typed DOUBLE in the calling routine. See notes for SINGLE/F.

NADD - Produce the MPI sum of two MPI's

Syntax:

CALL NADD (LEN, A, B, C)

Single-precision equivalent:

$$C = A + B$$

Notes:

The result may be the same array as either or both of the addends. Addition takes place from right to left. All arrays are the same length.

NSUB - Subtract one MPI from another and form an MPI difference

Syntax:

CALL NSUB (LEN, A, B, C)

Single-precision equivalent:

$$C = A - B$$

Notes:

See NADD

NMULT - Produce the MPI product of two MPI's

Syntax:

CALL NMULT (LEN, A, B, PROD)

Single-precision equivalent:

$$PROD = A * B$$

Notes:

A and B are the same length. PROD must be twice that size (2*LEN). The multiplier and the multiplicand may be the same array, but the product must be a different array than either of the other two. Multiplication is done from right to left, accumulating partial products along the way.

NMULTH - Produce a half-size product of two MPI's

Syntax:

CALL NMULTH (LEN, A, B, PROD)

Single-precision equivalent:

$$PROD = A * B$$

Notes:

Same as NMULT, with the following exception: all arrays are the same length. This operation may be used when it is known that the MPI's in A and B use only half their available precision or less. NMULTH is used by the divide routine and is thus also made available to the user.

NDIV - Divide one MPI by another and produce an integer quotient and an integer remainder

Syntax:

CALL NDIV (LEN, A, B, QUOT, REM)

Single-precision equivalent:

QUOT = A / B

REM = A - QUOT * B

Notes:

All arrays are the same length. The quotient and remainder should be different arrays, and not the same as either the dividend or divisor, since they are also used for working storage for intermediate results. The dividend and the divisor may be the same array.

NSQRT - Produce the square root of a multiprecision number

Syntax:

CALL NSQRT (LENA, A, SQRTA)

Single-precision equivalent:

SQRTA = SQRT(A)

Notes:

Both the MPI and its square root must be arrays of the same length. The maximum length for these arrays is 100. If a larger array size is desired, use NSQRTW. If the number is not a perfect square, the square root, when squared, is accurate to half its digits.

NSQRTW - Same as NSQRT, but with user-defined working storage

Syntax:

CALL NSQRTW (LENA, A, SQRTA, WORK)

Single-precision equivalent:

SQRTA = SQRT(A)

Notes:

Same as for NSQRT, with the following addition and exception. WORK is an array of size at least 3*LENA which will be used for temporary storage while calculating the square root. There is no upper bound on array size for NSQRTW, and it can therefore be used for extremely large numbers.

Error Indications

1. ARRAY LENGTH GREATER THAN MAXIMUM LENGTH ALLOWED, JOB ABORTED

Fatal error issued by NSQRT when the length of the multiprecision integer array is greater than 100. Recovery: none. User action: use a smaller array size and rerun, or else use NSQRTW, which allows larger arrays.

2. ILLEGAL CHARACTER [] IN MULTIPRECISION NUMBER, TREATED AS ZERO

Non-fatal error issued by MREAD when it finds a non-numeric character to convert to binary. The offending character is displayed between the square brackets. Recovery: the character is replaced by zero and conversion continues. User action: check for reason behind presence of illegal character (e.g., numeric shift key not pressed when keypunching number, etc.).

39. NEGATIVE ARGUMENT TO NSQRT, ZERO RETURNED

Non-fatal error issued by NSQRT when it receives an illegal argument. Recovery: the square root is set to zero. User action: check for reason behind negative argument.

Examples

(1) To calculate and print $N!$ for any $N \leq 100$.

$100! < 10^{200}$ so that $\frac{200}{13} = 15$ words are sufficient to

contain the number.

```
PROGRAM FACT (INPUT, OUTPUT)
DIMENSION NFACT(15), MUL(15), ITMP(30)
CALL MOVE (1,15,1,NFACT)
CALL MOVE (1,15,0,MUL)
READ 10,N
10 FORMAT (I5)
DO 20 I=2,N
MUL(15)=I, or, CALL MOVE(1,15,I,MUL)
CALL NMULT(15,NFACT,MUL,ITMP)
CALL MOVE(15,15,ITMP,NFACT)
20 CONTINUE
CALL CONVTI(15,NFACT,ITMP)
PRINT 30,N,ITMP
30 FORMAT (*I*, I5, * FACTORIAL = *, 10A10 / (15X,10A10))
STOP
END
```

- (2) To find the integer value and the first 14 digits of the fractional part of a multi-precision number A of length N divided by a multi-precision number B of length N

```
CALL NDIV (N,A,B,C,D)
CALL CONVTI(N,C,IOUT)
CALL SINGLE(N,D,X)
CALL SINGLE(N,B,Y)
W=X/Y
PRINT 100,IOUT
100 FORMAT (* INTEGER PART OF QUOTIENT=*, 10A10)
PRINT 110,W
110 FORMAT (* FRACTIONAL PART OF QUOTIENT=*,E24.14)
```