JOURNAL OF RESEARCH of the National Bureau of Standards—B. Mathematical Sciences
Vol. 75B, Nos. 1 and 2, January–June 1971

Exact Solutions of Linear Equations With Rational Coefficients*

A. S. Fraenkel and D. Loewenthal*****

(January 11, 1971)

Improvements of a congruential method for finding the exact solutions of systems of linear equations with rational coefficients are described. Typical execution times on the CDC 1604–A are given, as well as the Fortran program.

Key words: Congruences; exact solutions; linear equations; modular arithmetic.

1. Introduction

An algorithm for computing the exact solutions of linear equations with rational coefficients, and its computer implementation, were described in [1].[1] The basic idea of the algorithm is to convert the original system of equations into a system of congruences modulo various primes p_i, and combining the solutions by a procedure suggested by the Chinese Remainder Theorem. This process is continued until the sequence of solutions modulo $p_1p_2 \ldots p_k$, $k=1, 2, \ldots$ converges to the true solution. The major part of the computation is performed in single precision. See also Newman [4] who used the method for computing the inverses of ill-conditioned matrices, and Knuth [3, p. 256], who remarked that for ill-conditioned matrices the procedure "gives a method for obtaining the *true* answers in less time than any known method can produce reliable *approximate* answers!"

Our program provides for a final substitution check for verifying convergence, i.e., verifying that the computed values satisfy the original system. It is applied after two successive iterations produce no change in the solution vectors. If so implemented, and assuming sufficient memory space, the method produces the exact solution space for *any* solvable linear system with rational coefficients. Moreover, the algorithm is effective, in the sense that the exact solution space is produced within a reasonable time for systems that are not too large. It cannot end up with a wrong or no solution, as additional iterations are made should the substitution check fail.

The implementation of the algorithm has since been improved. As a result, the computation of the 9 independent exact solutions of a system of 111 homogeneous equations in 120 unknowns of rank 111 with integral coefficients in the range $[-2180, 2568]$ which took 60 min on a CDC 1604–A by the old method, now takes only 19 min on the same computer. The program is still in the form of standard Fortran subroutines, and no pains were taken to write a particularly economic program.

In the sequel it is assumed that the reader is reasonably familiar with the essential features of [1].

AMS Subject Classification: Primary 1006, Secondary 6535.

*An invited paper. This work was sponsored, in part, by the U.S. National Bureau of Standards. Reproduction in whole or in part is permitted for any purpose of the U.S. Government, GP #9661.

**University of California, Los Angeles. Permanent address: The Weizmann Institute of Science, Rehovot, Israel, and Bar Ilan University, Ramat Gan, Israel.
***Tel-Aviv University, Ramat Aviv, Tel-Aviv, Israel.

[1] Figures in brackets indicate the literature references at the end of this paper.

2. The Improvements

There are four main improvements: Use of larger primes; using the Cantor representation for constructing the solution mod $p_1p_2 \ldots p_k$ after the kth iteration (by the Chinese Remainder Theorem); a logical simplification in determining whether to retain or discard a new prime; and using a faster algorithm for finding the greatest common divisor (g.c.d.) of the components of any solution vector. These improvements will now be described one for one.

(i) *Use of larger primes.* In [1], primes of the order of 10^7, just less than half the machine word length, were employed. Primes of the order of 10^{14}, just less than one machine word are now used. Such primes were supplied to us through the courtesy of Herschel F. Smith from IBM. This saves about half the number of iterations (primes). In the above example, it reduced the number of iterations from six to four. Of course, we could have used the squares of primes of the order of 10^7 to obtain the same effect. However, the probability that a prime divides any of the principal minors of a matrix decreases with the size of the prime. Hence it is of some advantage to use the largest primes just less than a machine word.

(ii) *Use of the Cantor representation.* Let $m_1, \ldots, m_s$ be odd positive integers, relatively prime in pairs. Any number N in the range

$$\left[-\frac{m_1m_2 \ldots m_s}{2}\right] < N \leq \left[\frac{m_1m_2 \ldots m_s}{2}\right]$$

is uniquely representable by the *Cantor representation*, also referred to, by several authors, as the *mixed radix representation:*

$$N = a_1 + a_2m_1 + a_3m_1m_2 + \ldots + a_sm_1m_2 \ldots m_{s-1},$$

where

$$\left[-\frac{m_i}{2}\right] < a_i \leq \left[\frac{m_i}{2}\right], \qquad i = 1, 2, \ldots, s.$$

The Cantor representation of a number given by its residues mod m_i, $i = 1, 2, \ldots, s$, can be determined by computations in which all numbers occurring have absolute value not exceeding max m_i, in the present case single precision. See e.g., [2]. The solution check of [1] is now performed when for any fixed k, $a_k = 0$ for all components of the solution vectors.

Thus, the main computation of the solution vectors is now performed in single precision, resulting in a large time saving. Multiple precision is required only in the final conversion of the Cantor representation to decimal representation, and in the—optional—subsequent computation of the g.c.d. of the components of each solution vector.

(iii) *A logical simplification.* On pp. 110–111 of [1], a method was described which guarantees that each triangularization converges to the *same* largest nonsingular submatrix of the coefficient matrix A. For this, a lexicographic ordering of rows and columns of A had to be checked. This has now been changed as follows: The triangularization of A (mod p_1) induces a certain interchange of rows and columns in A. If ρ is the rank of A mod p_1, let $i_1, \ldots, i_\rho, j_1, \ldots, j_\rho$ be the rows and columns of A appearing in the triangularized principal submatrix of order ρ. The rows and columns of A itself are now reordered, so that $i_1, \ldots, i_\rho, j_1, \ldots, j_\rho$ appear as its first ρ rows and columns. Call this permuted matrix B. In each subsequent iteration mod p_i ($i > 1$), the triangularization is performed without changing any of the first ρ rows and columns in B. In the triangularized matrix, denote by ρ_i the order of the largest nonsingular principal submatrix. If $\rho_i < \rho$, p_i must be discarded; if $\rho_i > \rho$, the primes $p_1, \ldots, p_{i-1}$ must be discarded and a new matrix B with a greater rank is formed from A; if $\rho_i = \rho$, which is the normal case, the solution mod p_i is used to determine the coefficients a_i of the Cantor representation of all components of the solution vectors.

This modification does not save much time. In practice it only saves logical operations and manipulations, as it almost never happens that a given large prime divides a nonzero principal minor. However, the logical simplification results in a more compact and elegant program.

(iv) *Computation of the g.c.d.* The last step in [1] was to find the g.c.d. of the components of each of the solution basis vectors and to divide them out, so as to obtain *primitive* solutions, i.e., solutions such that the g.c.d. of the components of each solution vector is unity. Instead of computing the g.c.d. by the Euclidean Algorithm, i.e., by a series of multiple precision divisions, it is found by a series of shifts, using an economic algorithm of Stein [5], which, according to Knuth [3, p. 297], was previously given by Silver and Terzian:

Let t be any positive integer stored in binary form in a computer register. The highest power of 2 dividing t can easily be determined by shifting t to the right until its least significant nonzero bit is located in the least significant position of the register. This results in an odd integer $t' = 2^{-k}t$. We shall designate this operation by *shift*, i.e., $t' = \text{shift}(t)$.

Let a, b be two positive integers, and let

$$a_0 = \text{shift } (a) = 2^{-k}a, \qquad b_0 = \text{shift } (b) = 2^{-l}b.$$

Let $m = \min(k, l)$. The algorithm now proceeds as follows:

$$\begin{aligned} a_1 &= \text{shift } |a_0 - b_0|, & b_1 &= \min(a_0, b_0) \\ a_2 &= \text{shift } |a_1 - b_1|, & b_2 &= \min(a_1, b_1) \\ &\cdot \\ &\cdot \\ &\cdot \\ a_n &= \text{shift } |a_{n-1} - b_{n-1}|, & b_n &= \min(a_{n-1}, b_{n-1}) \\ a_{n+1} &= \text{shift } |a_n - b_n|. \end{aligned}$$

The procedure terminates when $a_n = b_n$, and then $(a, b) = 2^m a_n$.

In our example, a multiple precision g.c.d. was obtained for each of the 9 solution vectors. The computation of these g.c.d. and dividing them out now takes 2.5 min instead of the previous 7.5 min.

Finally, we should remark that the substitution check, which is also a multiple-precision operation, takes 6 min for our example. It is extremely rare that a substitution check fails when large primes are used, and therefore it seems reasonable to dispose of it in general. In fact, no substitution check is normally made in any of the conventional iterative schemes for solving linear equations—including those for which convergence is not guaranteed a priori even when the system is known to be solvable.

On the other hand, it is easy to fabricate a failing case. For simplicity assume that there is only one solution vector $V = (v_1, v_2, \ldots, v_t)$, whose Cantor representation is

$$v_i = a_{i1} + a_{i2}p_1 + a_{i3}p_1p_2 + \ldots + a_{ik}p_1p_2 \ldots p_{k-1} + \ldots + a_{is}p_1p_2 \ldots p_{s-1}, \quad 1 \leq i \leq t,$$

where $p_1, p_2, \ldots$ are distinct primes. For obtaining a failing case, we simply choose $a_{ik} = 0$, $1 \leq i \leq t$, and a_{ij}, $j \neq k$ arbitrarily, with the only condition that $a_{il} \neq 0$ for some $l > k$ and for some i.

A vector V of this form can clearly also be characterized as follows. Suppose that $V^{k-1} = (v_1^{k-1}, v_2^{k-1}, \ldots, v_t^{k-1})$ is the solution mod $p_1p_2 \ldots p_{k-1}$, i.e.,

$$v_i \equiv v_i^{k-1} (\text{mod } p_1p_2 \ldots p_{k-1}), \qquad 1 \leq i \leq t.$$

Then

$$a_{ik} = 0,\ 1 \leq i \leq t,\ a_{il} \neq 0,\ l > k \qquad \text{for some } i$$

if and only if

$$v_i \equiv v_i^{k-1} (\text{mod } p_k),\ 1 \leq i \leq t,\ v_i \neq v_i^{k-1} \qquad \text{for some } i.$$

Disposing of the substitution check and the g.c.d. computation, solution time in our example

reduces to 10 min. By comparison, the approximate computation of a *well-conditioned* 111×111 system by standard numerical floating number techniques takes about 3 min on the CDC 1604–A.

3. Machine Program

The program given below is the full version, i.e., it includes computation of the g.c.d. and the substitution check imbedded in the program "Solve." The matrix of the homogeneous equations coefficients is read into the computer from magnetic tape no. 1. Other data to the program is provided by three data cards. The first card contains three numbers: the number of rows and columns of the matrix, and the number of primes supplied. The second and third cards contain the supplied primes.

Functions used in this program are:

"Inv"—computes the inverse of an integer t mod p which is required for solving the system mod p.

"Irem"—IREM(t_1, t_2, p) outputs an integer t satisfying $t \equiv t_1 t_2 (\mathrm{mod}\ p)$, $-p < t < p$. Since the product $t_1 t_2$ is normally a double precision number, this routine is written in machine language.

"Shift"—shifts a multiple-precision number to the right k positions until a binary 1 appears at the least significant position. On its left, k binary zeros are shifted in. Because of its multiple-precision character, this routine is also written in machine language. In the program listed below it appears directly after "Irem," before "Setpreci."

"Mod"—MOD (t, p) transforms the single-precision integer t obtained as a result of an addition or subtraction of two numbers x_i, $-(p-1)/2 \leq x_i \leq (p-1)/2$, $i = 1, 2$, or of an IREM operation, to an integer t' satisfying $t' \equiv t (\mathrm{mod}\ p)$ and $-(p-1)/2 \leq t' \leq (p-1)/2$.

"Setpreci"—A Fortran subroutine which controls the multi-precision package which is written in machine language. Because this package is in a binary deck form, it is not included in the program listed below. Functions used from this package are: "If(Itest(t))"—for checking if a multiple precision integer t is negative zero or positive; "Multout"—for printing a multiple precision integer.

```
      PROGRAM SOLVE
      COMMON M1,M2,NZ,M,N,KP,KQ,KI,MA(111,120),IZ(120),NEW(9,9),
     1 NT(4,120),KT(20),IY(120),JSOL(120),KSOL(120),JS(112,9,6),KG(8)
     2 ,KE(8),KF(8),KO(8),NA(6),NB(6),NC(6),ND(6)
      TYPE MULTIPL6(6) A,B,C,KG,D,GCD,KC ,KSOL
      COMMON/SHIFTS/KK
      EQUIVALENCE (A,NA),(B,NB),(C,NC),(D,ND)
      READ 55,M,N,NZ
      READ 797,(KT(I),I=1,nz)
      REWIND 1 $ DO 987 I=1,M
  987 READ TAPE 1,(MA(I,J),J=1,N)
      PRINT 7,M,N,NZ $ DO 63 I=1,M
   63 IY(I)=I $ KR=0 $ PRINT 2,(KT(I),I=1,NZ) $ DO 94 J=1,N
   94 IZ(J)=J $ DO 200 L=1,NZ $ KP=KT(L) $ KQ=KP/2. $ KD=MK=1
      DO 75 I=1,M$DO 75 J=1,N$IS=MA(I,J)-(MA(I,J)/KP)*KP$ IF(IS)76,75,75
   76 IS=KP+IS
   75 MA(I,J)=IS
   23 DO 122 K=MK,KR $ IF(IT.EQ.1)10,43
    8 PRINT 88,KP $ MS=1 $ GO TO 201
   43 I1=IY(K) $ J1=IZ(K) $ KK=MA(I1,J1)
   10 MK=MK+1 $ IF(KK-1)8,33,34
   34 KD=IREM(KD,KK,KP) $ IM=INV(KK,KP) $ DO 32 JL=MK,N $ J2=IZ(JL)
   32 MA(I1,J2)=IREM(MA(I1,J2),IM,KP)
```

```
 33 DO 122 I = MK,M $ I2 = IY(I) $ IL = MA(I2,J1) $ IF(IL.EQ.0)122,31
 31 DO1J = MK,N$J2 = IZ(J)$IS = MA(I2,J2) - IREM(MA(I1,J2),IL,KP)$IF(IS)72,1,1
 72 IS = KP + IS
  1 MA(I2,J2) = IS
122 CONTINUE
    DO6I = MK,M$I1 = IY(I)$DO6J = MK,N$J1 = IZ(J)$KK = MA(I1,J1)$IF(KK.EQ.0)6,5
  5 KR = MK$IT = 1$IY(I) = IY(KR)$IY(KR) = I1$IZ(J) = IZ(KR)$IZ(KR) = J1$GOTO23
  6 CONTINUE $ IW = IT - 1 $ KF(IT) = KP $ IS = 1 $ KE(1) = 1 $ DO 70 I = 1,IW
    I1 = I + 1 $ IS = IREM(IS,KF(I),KP)
 70 KE(I1) = IS
    PRINT 51,KP,KD,IT $ IF(IT.EQ.1)47,29
 29 IS = KE(IT) $ IV = INV(IS,KP) $ NU = 0 $ DO 944 KL = 1,IW
    KS = IREM(KO(KL),KE(KL),KP) $ IF(KS)971,979,979
971 KS = KP + KS
979 NU = NU + KS $ IF(NU - KP)944,974,974
974 NU = NU - KP
944 CONTINUE $ IS = KD - NU $ MS = KO(IT) = MOD(IREM(IS,IV,KP)) $ GO TO 95
 47 MS = 1 $ KO(1) = MOD(KD) $ KI = N - KR $ PRINT 56,KR,KI,(IY(I),I = 1,M)
    PRINT 58,(IZ(J),J = 1,N)
 95 DO 26K=1,KI$DO26I=1,KR$I1=I-1$K1=KR-I1$MX=0$DO38J=1,I1$K2=MK-J
    MX = MX - IREM(JSOL(K2),MA(IY(K1),IZ(K2)),KP) $ IF(MX)78,38,38
 78 MX = MX + KP
 38 CONTINUE$IS = MX - IREM(KD,MA(IY(K1),IZ(KR + K)),KP)$IF(IS)37,27,27
 37 IS = IS + KP
 27 JSOL(K1) = IS $ IF(IT.EQ.1)28,90
 28 JS(K1,K,1) = MOD(IS) $ GO TO 26
 90 NU=0 $DO44 KL=1,IW $ KS=IREM(JS(K1,K,KL),KE(KL),KP)$IF(KS)71,79,79
 71 KS = KP + KS
 79 NU = NU + KS $ IF(NU - KP)44,74,74
 74 NU = NU - KP
 44 CONTINUE $ JS(K1,K,IT) = 0 $ IS = IS - NU $ IF(IS.EQ.0)26,45
 45 MS = 1 $ JS(K1,K,IT) = MOD(IREM(IS,IV,KP))
 26 CONTINUE $ IT = IT + 1
201 REWIND 1 $ DO 202 I = 1,M $ READ TAPE 1,(MA(I,J),J = 1,N)
202 CONTINUE $ IF(MS.EQ.0)203,200
200 CONTINUE $ GO TO 80
203 PRINT 153
    KG(1) = 1 $ DO 710 I = 2,IW
710 KG(I) = KG(I - 1)*KF(I - 1)
    KC = KO(1) $ DO 720 I = 2,IW
720 KC = KC + KO(I)*KG(I)
    C = KC $ IV = 7 - IW $ IF(ITEST(C))751,80,752
751 C = - C
752 CALL SHIFT(NC(IV),IW) $ I5 = KK
    DO 49 J = 1,KI $ NOS = I5 $ A = C $ PRINT 100,J
    DO 50 I = 1,KI $ KSOL(KR + I) = 0
 50 CONTINUE $ KSOL(KR + J) = KC
    DO 40 I = 1,KR $ B = JS(I,J,1) $ JSOL(I) = 1 $ DO 42 K = 2,IW
 42 B = B + JS(I,J,K)*KG(K)
    CALL MULTOUT(B,51)
    KSOL(I) = B
    IF(ITEST(B))715,716,717
```

```
716 JSOL(I)=0 $ GO TO 40
715 B=-B $ JSOL(I)=-1
717 DO 718 K=1,IW $ I1=7-K
718 JS(I,J,K)=NB(I1) $ CALL SHIFT(NB(IV),IW) $ IF(KK.GE.NOS)721,722
722 NOS=KK
721 D=A-B
    IF(ITEST(D))725,40,724
725 D=-D
724 CALL SHIFT(ND(IV),IW) $ A=B $ B=D $ GO TO 721
 40 CONTINUE $ GCD=A*2**NOS
    PRINT 669
    CALL MULTOUT(GCD,51)
    DO 730 I=1,KR $ IF(JSOL(I).EQ.0)730,731
731 DO 732 K=1,IW $ I1=7-K
732 ND(I1)=JS(I,J,K) $ NV=D=D/GCD $ J2=IZ(I) $ IF(JSOL(I).EQ.1)733,734
734 NV=-NV
733 PRINT 538,(NT(K,J2),K=1,M1)
    PRINT 539,NV
730 CONTINUE $ J2=IZ(KR+J) $ NV=KC/GCD
    PRINT 538,(NT(K,J2),K=1,M1)
    PRINT 539,NV
    DO 61 I=1,M $ A=0 $ DO 52 JL=1,N
 52 A=A+KSOL(JL)*MA(I,IZ(JL)) $ IF(ITEST(A))82,61,82
 82 PRINT 53,J,I
 61 CONTINUE
 49 CONTINUE
  2 FORMAT(//50X,8H PRIMES.,//(I14,2X))
  7 FORMAT(1H1,3H M=,I3,3H N=,I3,4H NZ=,I3,//,20X,7H MATRIX,//)
 51 FORMAT(1H1,50X6HPRIME=,I14/I50,12H=DETERMINANT/I50,11H
                                                    ITERATIONS)
 53 FORMAT(1H1,20X,8H NOT YET,20X,2I20)
 55 FORMAT(23I5)
 56 FORMAT(//20X,11HFIRST PRIME,5X,5HRANK=,I3,5X,8HNULLITY=,I3//40X,14
   1HORDER OF LINES,//(23Iff))
 58 FORMAT(//,40X,16HORDER OF COLUMNS,//(23I5))
 88 FORMAT(50XI14,23H THIS PRIME WAS DROPPED)
100 FORMAT(1H1,//50X,I3///)
153 FORMAT(/,20X,32HCONVERGENCE OBTAINED AND CHECKED,//)
538 FORMAT(/,20X,10I3)
539 FORMAT(I20, 8HX1 X2 X3,//)
669 FORMAT(1H1,//50X,6H G.C.D//)
797 FORMAT(5(I14,2X))
 80 RETURN $ END
    FUNCTION INV(KX,KP)
    K1=KP $ K2=KX $ M1=0 $ M2=1 $ IF(K2-1)2,4,5
  4 INV=K2 $ RETURN
  6 M1=M2 $ M2=M3 $ K1=K2 $ K2=K3
  5 IQ=K1/K2 $ M3=M1-M2*IQ $ K3=K1-K2*IQ $ IF(K3-1)2,7,6
  7 K2=M3-(M3/KP)*KP $ IF(K2)8,2,4
  8 K2=K2+KP $ GO TO 4
  2 INV=0
    END
```

```
      FUNCTION MOD(IX)
      COMMON M1,M2,M3,M4,M5,KP,KQ
      IF(IX)1,2,3
    2 MOD=IX $ RETURN
    1 IF(KQ+IX)4,2,2
    4 MOD=KP+IX $ RETURN
    3 IF(KQ-IX)5,2,2
    5 MOD=IX-KP
      END
          IDENT          IREM
          ENTRY          IREM
IREM      SLJ            **
          SIU       1    IR3
          LIU       1    *-1
          SIL       1    IR1
          INI       1    1
          SIL       1    IR2
          INI       1    1
IR1       SIL       1    IR3
          LIU       1    **
          LDA       1
          LIL       7    IR1
IR2       MUI       1
          LIU       1    **
          DVI       1
          LLS            48
IR3       ENI       1    **
          SLJ            **
          END
          IDENT          SHIFT
          ENTRY          SHIFT
SHIFTS    BLOCK
          COMMON         NSHIFT
SHIFT     SLJ            **
          SIU       1    EX
          SIL       2    EX
          LIU       1    SHIFT
          ENA
          STA            =SNZ
          LDA       1
          SAL            N
          ARS            24
          INI       1    1
N         SIU       1    EX+2
          LIL       1    **
          INI       1    -1
          SAL            BEGIN
          SAL            CHANGE
          SAL            LOOP
          INA            1
          SAU            STORE
BEGIN     ENI       2
```

```
                LDA     1   **
                AJP         ZERO
                ENQ
SH              LRS         1
                QJP     M   NEG
                INI     2   1
                SLJ         SH
NEG             LLS         1
                ENQ     2
                STQ         NSHIFT
                QJP         EX
                IJP     1   LOOP
EXIT            STA     7   BEGIN
EX              ENI     1   **
                ENI     2   **
                LDA         NZ
                RAD         NSHIFT
                SLJ         **
LOOP            STA         =ST
                LDA     1   **
                ENQ
                LRS     2
                STA         =ST1
                QRS         1
                LDA         T
                ADL         =O377777777777777
STORE           STA     1   **
                LDA         T1
                IJP     1   LOOP
                SLJ         EXIT
ZERO            SIL     1   =SNN
ZERO1           INI     2   47
                IJP     1   CONT
                ENA         -1
                STA         NSHIFT
                SLJ         EX
CONT            LDA     7   BEGIN
                AJP         ZERO1
                SIL     2   =SNZ
                LIL     2   NN
CHANGE          LDA     7   BEGIN
                STA     2   **
                INI     2   -1
                IJP     1   CHANGE
                ENA
 +              STA     7   CHANGE
                IJP     2   *
                LIL     1   NN
                SLJ         BEGIN
                END
          SUBROUTINE SETPRECI(NUM,LOG)
          COMMON / MULTINCM / N,IA(50),IC(50),ID(50)
```

```
      DATA (N=6)
      RETURN
      ENTRY ERR2
      PRINT 100,NUM,LOG
  100 FORMAT (46H ERROR IN ADDITION TYPE OVER-FLOW CALL FROM
     1,O16, ,18H FIRST LOCATION ,O16)
      STOP
      ENTRY ERR1
      PRINT 101,NUM,LOG
  101 FORMAT (49H ERROR IN DIVISION TYPE ZERO-DIVISOR CALL
     1FROM ,O16, ,16H FIRST LOCATION ,O16 )
      STOP
      ENTRY PRINTOUT
      WRITE (NUM,102) (ID(K),K=1,N)
  102 FORMAT (5X,5(I14,X))
      END
    DATA
  111  120    10
7908189600581  7908189600583  7908189600587  7908189600589  7908189600593
7908189600599  7908189600601  7908189600607  7908189600611  7908189600613
```

4. References

[1] Borosh, I., and Fraenkel, A. S., Exact solutions of linear equations with rational coefficients by congruence techniques, Math. of Comp. **20,** 107–112 (1966).

[2] Fraenkel, A. S., New proof of the generalized Chinese Remainder Theorem, Proc. Amer. Math. Soc. **14,** 790–791 (1963).

[3] Knuth, D. E., The art of computer programming, Vol. **2:** seminumerical algorithms, Addison-Wesley, 1969.

[4] Newman, M., Solving equations exactly, J. Res. Nat. Bur. Stand. (U.S.), **71B** (Math. & Math. Phys.), No. 4, 171–179 (1967).

[5] Stein, J., Computational problems associated with Racah Algebra, J. Comp. Physics **1,** 397–405 (1967).

(Paper 75B1&2–345)