JOURNAL OF RESEARCH of the National Bureau of Standards — B. Mathematics and Mathematical Physics
Vol. 71B, No. 4, October–December 1967

Solving Equations Exactly

Morris Newman

Institute for Basic Standards, National Bureau of Standards, Washington, D.C. 20234

(August 22, 1967)

A congruential method for finding the exact solution of a system of linear equations with integral coefficients is described, and complete details of the program are given. Typical numerical results obtained with an existing program are given as well.

Key Words: Exact solutions, Hilbert matrices, linear equations, modular arithmetic.

1. Introduction

The problem of the solution of a given set of linear equations $Ax=b$ on a high-speed digital computer has been studied intensively, and there are a large number of methods, more or less satisfactory, for carrying out such a solution. Nevertheless occasions arise when existing methods are inadequate, either because the solutions are required exactly, or because the coefficient matrix A is "ill-conditioned." A notorious example of the latter is furnished by the Hilbert matrices $A=H_n$ given by

$$H_n=\left(\frac{1}{i+j-1}\right), \qquad 1 \leqq i, j \leqq n.$$

Here even for moderate values of n (say 7 or 8) the solution by any of the usual methods becomes awkward, if not impossible. For these reasons, as well as many others, the method of solution described in this note is of interest. It is not at all sensitive to the condition of A, since it determines the exact solution (and not an approximate one) by number-theoretical methods. Thus the usual ills caused by round-off, truncation, etc., do not exist. It can fail, of course, but not for any of the reasons which cause the usual methods to fail. Its principal disadvantages are that it is limited to systems with integral elements, and that it is somewhat time-consuming. It is not particularly suited to hand computation, and definitely finds its role in high-speed digital computation.

A related discussion and elaboration are given by I. Borosh and A. S. Fraenkel in their article.[1]

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

2. Description of the Method

Let A be a nonsingular integral $n \times n$ matrix, b an integral $n \times 1$ vector. Put

$$d=\det\ (A)$$

and denote the adjoint of A by A^{adj}, so that A^{adj} is also a nonsingular integral $n \times n$ matrix, satisfying

$$AA^{\mathrm{adj}}=A^{\mathrm{adj}}A=dI.$$

For any matrix $B=(b_{ij})$ define

$$M(B)=\max_{i,j}\ |b_{ij}|.$$

Let x be the solution of the system

$$Ax=b.$$

Then

$$x=\frac{1}{d}\, A^{\mathrm{adj}}b=\frac{1}{d}\, y,$$

where

$$y=A^{\mathrm{adj}}b$$

is also an integral vector.

The method is based on the following simple observation: Suppose that m is an integer such that

$$(d, m)=1,$$

$$m>2\ \max\ (|d|, M(y)).$$

Then any pair of solutions d_m, y_m, of the congruences

$$d \equiv d_m \bmod m,$$

$$Ay_m \equiv db \bmod m$$

satisfying

$$|d_m| < \tfrac{1}{2} m,$$

$$M(y_m) < \tfrac{1}{2} m$$

must in fact coincide with d, y.

It is clear that $d = d_m$, since $d \equiv d_m \bmod m$ and $|d| < \frac{1}{2}m$, $|d_m| < \frac{1}{2}m$. Furthermore since $Ay \equiv db \bmod m$, $Ay_m \equiv db \bmod m$, we have $A(y - y_m) \equiv 0 \bmod m$. Hence $A^{\mathrm{adj}}A(y-y_m) \equiv 0 \bmod m$, $d(y - y_m) \equiv 0 \bmod m$; and since $(d, m) = 1$, it follows that $y \equiv y_m \bmod m$. Finally, since both $M(y) < \frac{1}{2}m$, $M(y_m) < \frac{1}{2}m$, we have $y = y_m$.

In applying the observation above we choose $m = m_1 \cdot m_2 \ldots m_s$, where $(m_i, m_j) = 1$ for $i \neq j$, and use the Chinese remainder theorem. Define m_i' by

$$\frac{m}{m_i} m_i' \equiv 1 \bmod m_i, \qquad 0 \leqq m_i' < m_i, \qquad 1 \leqq i \leqq s.$$

Then the solution of the system

$$z \equiv a_i \bmod m_i, \qquad 1 \leqq i \leqq s$$

is given by

$$z \equiv \sum_{i=1}^{s} \frac{m}{m_i} m_i' a_i \bmod m$$

For each i, $1 \leqq i \leqq s$ determine d_{m_i}, y_{m_i} so that

$$d \equiv d_{m_i} \bmod m_i,$$

$$Ay_{m_i} \equiv db \bmod m_i.$$

(We shall show how this can be done without having computed d previously.)

Next determine d_m, y_m by

$$d_m \equiv \sum_{i=i}^{s} \frac{m}{m_i} m_i' d_{m_i} \bmod m,$$

$$y_m \equiv \sum_{i=i}^{s} \frac{m}{m_i} m_i' y_{m_i} \bmod m.$$

Finally, determine d, y so that

$$d \equiv d_m \bmod m,$$

$$y \equiv y_m \bmod m,$$

and $|d| < \frac{1}{2}m$, $M(y) < \frac{1}{2}m$.

The practical utility of this method rests primarily on the fact that the bulk of the computations are performed modulo the "single-length" numbers m_i, $1 \leqq i \leqq s$. The final reconstruction according to the Chinese remainder theorem is done accumulatively throughout the computation and is the only time when multi-length operations are required. In practice the moduli m_i are chosen as prime powers, for then the solution of the congruential system

$$d \equiv d_{m_i} \bmod m_i,$$

$$Ay_{m_i} \equiv db \bmod m_i$$

becomes particularly simple.

The ideal program would first determine a permissible value for m. This can be done by Hadamard's inequality, for example, which states that the absolute value of a determinant does not exceed the product of the euclidean lengths of its row vectors. This implies that

$$|d| \leqq n^{\frac{n}{2}} M(A)^n,$$

$$M(y) \leqq n(n-1)^{\frac{n-1}{2}} M(A)^{n-1} M(b)$$

which are at once derivable from Hadamard's inequality and from the specific form of the elements of A^{adj} as cofactors of the elements of A. Thus it is certainly sufficient to choose

$$m > c = 2 \max \left(n^{\frac{n}{2}} M(A)^n, \quad n(n-1)^{\frac{n-1}{2}} M(A)^{n-1} M(b)\right)$$

and satisfying

$$(m, d) = 1.$$

The simplest way to do this effectively would be to generate a sequence of different primes $p_1, p_2, \ldots, p_s$ such that $(d, p_i) = 1$, $1 \leqq i \leqq s$, $p_1 p_2 \ldots p_s > c$; and then choose $m = p_1 p_2 \ldots p_s$.

In practice, the procedure outlined above is unnecessarily conservative and time consuming. A good practical alternative is to operate instead with a predetermined set of moduli m_i, knowing full well that in certain instances the process will fail. Failure principally occurs if $(m_i, d) > 1$ for some i. By choosing the m_i as large primes, the probability of such an occurrence can be made so small that this is not an important practical consideration.

3. The Detailed Program

Suppose that the machine for which the program is being written is capable of multiplying or adding together any pair of whole numbers $< K$ in absolute value. We choose as moduli the s largest primes $m_i < K$, and precompute and store the numbers $\frac{m}{m_i}$, m_i'. Notice that $0 \leqq m_i' < m_i < K$, but that the numbers $\frac{m}{m_i}$ must be stored as multilength numbers. In addition m and $\frac{m-1}{2}$ are required and must be stored similarly. The choice of s is a matter of expediency: If $K \approx 10^{10}$ then $s=10$ should suffice for most purposes. The program (in broad outline) is as follows:

(1) Read in n, A, b.

(2) Clear s-length answer cells d, y.

(3) Set $i=1$ (modulus tally).

(4) Transfer A, b modulo m_i to working temporaries.

(5) Determine d_{m_i}, y_{m_i} so that

$$d_{m_i} \equiv d \bmod m_i,$$

$$Ay_{m_i} \equiv db \bmod m_i,$$

and d_{m_i} and the components of y_{m_i} lie between 0 and m_i-1, inclusive. (As will be shown, d need not be known.)

(6) Accumulate s-length results:

$$d+\frac{m}{m_i} m_i' d_{m_i} \rightarrow d,$$

$$y+\frac{m}{m_i} m_i' y_{m_i} \rightarrow y.$$

(7) If $i<s$, replace i by $i+1$ and go to (4); otherwise go on.

(8) Reduce d, y modulo m so that

$$|d| < \tfrac{1}{2}\ m,$$

$$M(y) < \tfrac{1}{2}\ m.$$

(9) Check s-length answers by multiplication:

$$Ay=db.$$

(10) Compute (in floating) the components of $x=\dfrac{1}{d}y$.

(11) Print results (d, y as s-length integers; x as floated numbers).

The basic computation of course takes place in (5), and we describe this in some detail. Suppose then that p is a prime, and we are interested in solving

$$d_p \equiv d \bmod p,$$

$$Ay_p \equiv db \bmod p.$$

Let $B=(b_{ij})$ be the $n \times (n+1)$ matrix (A, b). The procedure is as follows:

(5.1) Set $k=1$.

(5.2) Set $S=1$ (cell in which d_p is calculated).

(5.3) Determine r so that $r \leqq k \leqq n$, and $(b_{rk}, p)=1$. (If no such r exists, the machine prints out the information that the system is singular modulo p and halts.)

(5.4) If $r=k$, go to (5.6); otherwise go on.

(5.5) Interchange rows r and k, and replace S by $-S$. Continue to denote the resulting matrix by B.

(5.6) Determine b'_{kk} (by the euclidean algorithm) so $b_{kk}b'_{kk} \equiv 1 \bmod p$.

(5.7) Replace S by $b_{kk}S$, and b_{kt} by $b'_{kk}b_{kt}$, $k \leqq t \leqq n+1$, all computations being performed modulo p. Continue to denote the resulting matrix by B.

(5.8) For $1 \leqq s \leqq n$, $s \neq k$, replace b_{st} by

$$b_{st}-b_{sk}b_{kt}, \qquad k \leqq t \leqq n+1,$$

all computations being performed modulo p; and continue to denote the resulting matrix by B.

(5.9) If $k<n$, replace k by $k+1$ and go to (5.3); otherwise go on.

(5.10) $d_p=S$.

(5.11) $y_p^{(i)} \equiv Sb_{in+1} \pmod p$, $\quad 1 \leqq i \leqq n$.

It will be seen that the time required to accomplish step (5) is approximately equivalent to that required to solve a single linear system by the elimination method. Thus the total time is roughly that required to solve s such systems, and so is quite moderate.

We note one or two other points: In the accumulation described by (6), we reduce $m_i'd_{m_i}$ and $m_i'y_{m_i}$ modulo m_i before multiplication by $\dfrac{m}{m_i}$. The effect of this is that at the end of step (6), $0 \leqq d < sm$, $M(y) < sm$. The reduction required by step (8) is then relatively easy to accomplish.

It is certainly possible to provide for the contingency that $(d, m_i) > 1$ for some i by a more elaborate program. If this occurs, the faulty m_i must be discarded and a new one substituted. The necessary constants would then have to be recomputed and the problem started anew.

4. An Existing Program and Numerical Results

A pilot version of this program was written for the Q32 computer at the System Development Corporation, Santa Monica, Calif. The computer was used on a "time sharing" basis, under partial support of the National Institutes of Health on project number 2050404. The writer would like to thank Russell Kirsch of the National Bureau of Standards for generously providing access to this computer.

The following 10 primes were chosen as moduli: 9999889, 9999901, 9999907, 9999929, 9999931, 9999937, 9999943, 9999971, 9999973, 9999991.

In the appended JOVIAL program, the lines have the following function:

1–8.	Array declarations.
9–26.	Precomputed constants.
27–40.	Read in n, A, b.
41–42.	Clear answer cells.
43–44.	Set up modulus p.
45–51.	Transfer A, b modulo p to temporaries.
52–74.	Solve $Ax_p \equiv b \bmod p$.
75–78.	Compute d_p, y_p.
79–85.	Accumulate d, y.
86.	Reset modulus.
87–93.	Reduce to range $(-\frac{1}{2}\ m, \frac{1}{2}m)$.
94–97.	Print exact results (y, d).
98–109.	Multilength normalization subroutine.

110–117. Euclidean algorithm.
118. Error print.
119–122. Compute and print floated results (x).

A number of test problems were run with uniform success, among which were the Hilbert matrices of all orders up to and including $n=13$ (a limit imposed by considerations of time). Define

$$t_n = \text{L.C.M.}\ (1, 2, 3, \ldots, 2n-1).$$

Then $t_n H_n$ has integral elements. We chose b as the first unit vector, so that the output was the first column of $(t_n H_n)^{-1} = \frac{1}{t_n} H_n^{-1}$. Since H_n^{-1} is known exactly,[2] we were able to verify the results. In the numerical results that follow, the lines

$$A_0 \quad A_1 \quad A_2 \quad A_3 \quad A_4$$

$$A_5 \quad A_6 \quad A_7 \quad A_8 \quad A_9$$

stand for the 10-length integer

$$A_0 + 10^7 A_1 + \ldots + 10^{63} A_9.$$

The print-out has the form

y

d

x

We give the results for $n=9$, 10, 11, 12 as representative examples.

$n=9$

$t_n = 12252240$

2296512 8395179 399 0 0
0 0 0 0 0
−1860480 −5807169 −15993 0 0
0 0 0 0 0
8876160 9525337 205250 0 0
0 0 0 0 0
−3256960 −7152027 −1231505 0 0
0 0 0 0 0
8085120 5744088 4002393 0 0
0 0 0 0 0
−1092224 −6722299 −7471134 0 0
0 0 0 0 0
6170240 1488177 8004787 0 0
0 0 0 0 0
−2097280 −850387 −4574164 0 0
0 0 0 0 0
3134080 9645230 1080010 0 0
0 0 0 0 0
5508480 132897 480614 6 0
0 0 0 0 0
+.661103602E−005
−.264441441E−003
+.339366516E−002
−.203619910E−001
+.661764706E−001
−.123529412E+000
+.132352941E+000
−.756302521E−001
+.178571429E−001

1454720 2473993 2532211 4345 0
0 0 0 0 0
−7008640 −7462660 −344456 −215090 0
0 0 0 0 0
2138240 9402571 5511307 3441440 0
0 0 0 0 0
−7048320 −8802831 −8460751 −6097590 −2
0 0 0 0 0
1602944 8971893 5535157 9609881 10

[2] I. R. Savage and E. Lukacs, Tables of inverses of finite segments of the Hilbert matrix, NBS AMS **39**, 105–108 (1954).

0 0 0 0 0 $n=10$
-9007360 -7429732 -8837894 -4024703 -27
0 0 0 0 0 $t_n=232792560$
2773120 845307 5372030 7561453 41
0 0 0 0 0
-1275520 -3984119 -8999527 -279180 -38
0 0 0 0 0
5637760 6992059 4499763 139590 19
0 0 0 0 0
-2412416 -2253879 -2061061 -140580 -4
0 0 0 0 0
7928832 7977568 9387428 5426211 1011
0 0 0 0 0
+.429566993E−006
−.212635662E−004
+.340217058E−003
−.257997936E−002
+.108359133E−001
−.270897833E−001
+.412796698E−001
−.375939850E−001
+.187969925E−001
−.396825397E−002
7570112 599090 7074703 1 0
0 0 0 0 0
-4206720 -5945445 -4482183 -102 0
0 0 0 0 0
7031040 936185 7402580 1997 0
0 0 0 0 0
-5623040 -2071066 -5757414 -18645 0
0 0 0 0 0
4520960 5873099 2726424 97889 0
0 0 0 0 0
-2467072 -6793918 -6724558 -313245 0
0 0 0 0 0
1183360 416263 5752083 633949 0
0 0 0 0 0 $n=11$
-8664320 -1963766 -252678 -815078 0 $t_n=232792560$
0 0 0 0 0
1025920 2387982 1033370 645270 0
0 0 0 0 0
-6011520 -3283547 -7125942 -286786 0
0 0 0 0 0
6329472 7172313 1905861 54750 0
0 0 0 0 0
9768320 338817 4351703 3285011 0
0 0 0 0 0
+.519776062E−006
−.311865637E−004
+.608137992E−003
−.567595459E−002
+.297987616E−001
−.953560372E−001
+.192982456E+000
−.248120301E+000
+.196428571E+000
−.873015873E−001
+.166666667E−001
126080 3885341 3441107 9248615 3937
0 0 0 0 0
-4014720 -2801882 -1039178 -6275997 -281561
0 0 0 0 0

```
3676800 8710589 914159 3106599 6569771
0 0 0 0 0
−3864000 −5494130 −7784298 −2449239 −3909927
−7 0 0 0 0
4729600 7162434 5819510 3675134 3023534
47 0 0 0 0
−4094080 −1744323 −5084059 −3244700 −6326686
−187 0 0 0 0
6241920 3056831 7359009 5486372 4840050
482 0 0 0 0
−7910400 −721410 −8912605 −3235810 −4996514
−818 0 0 0 0                                         n=12
5456000 801567 5458450 4706456 4440571               t_n=5354228880
909 0 0 0 0
−2819200 −561097 −5820915 −294519 −6108400
−636 0 0 0 0
9127680 224438 8328366 117807 6443360
254 0 0 0 0
−7499520 −2008803 −5466306 −2444769 −3698464
−44 0 0 0 0
4841600 2905237 3881046 6773950 493200
14642 0 0 0 0
+.268946291E−007
−.192296598E−005
+.448692063E−004
−.504778570E−003
+.323058285E−002
−.128146453E−001
+.329519451E−001
−.559006211E−001
+.621118012E−001
−.434782609E−001
+.173913043E−001
−.303030303E−002
```

Jovial Program

```
ARRAY A 16 17 I;
ARRAY C 16 17 I;
ARRAY V 17 10 I;
ARRAY W 12 10 I;
ARRAY MOD 10 I;
ARRAY INV 10 I;
ARRAY Z 10 I;
ITEM SS F; ITEM TT F;  ITEM NUM F; ITEM DEN F;
VALUES MOD 9999889 9999901 9999907 9999929 9999931 9999937
9999943 9999971 9999973 9999991;
VALUES INV 7369185 2629157 6854848 9534445 6502552
3472916 5202223 3251378 129200 5053746;
VALUES W 4977739 8027834 5690123 538898 3121137 3768638 5584115
115082 9999483 0 5278071 15291    9546516 9532061 5812518
9202123 4699699 120098 9999471 0 6263753 7229845 9363502 8209581
9936740 6614348 4222392 122714 9999465 0 7782099 9773154 8216183
5103433 3253672 3499315 2247429 132922 9999443 0 3659841 431474
2231991 6034197 1042630 5307831 2048927 133898 9999441 0 9722683
2791024 2958205 5033108 173657 4212244 1433022 136874 9999435
0 6535597 4607850 2424276 3272840 1038248 8705894 785509 139922
9999429 0 2501001 296996 4616307 9611859 4673074 5857613 7307785
```

```
155097 9999401 0 9352927 6498696 8521281 1579588 4721737 3700720
7028477 156241 9999399 0 8058781 9352296 9214244 9260206 9463057
2740085 4312028 166897 9999381 0 7470971 3888109 6424092 5872382
4092685 4802284 3931831 2809951 172468 9999372 8735485 1944054
3212046 7936191 2046342 7401142 6965915 1404975 86234 4999686;

READ M ;N=M+1;
PRINT 14H (MATRIX BY ROWS);
I=0;
R1. H=0;
R2.READ C[I ,J];
IF J EQ N-2; GOTO R3;
J=J+1; GOTO R2;
R3. IF I EQ M-1; GOTO R4;
I=I+1; GOTO R1;
R4. PRINT 10H (RIGHT SIDE);
I=0;
R5. READ C[I ,N-1];
IF I EQ M-1; GOTO C1;
I=I+1; GOTO R5;

C1. I=0;C2.J=0;C3.V[I ,J]=0;IF J EQ 9;GOTO C4;
J=J+1;GOTO C3;C4.IF I EQ M ;GOTO R6;I=I+1;GOTO C2;

R6. L=0;
R7. P=MOD[L];

I=0;
R10. J=0;
R8. REMQUO (C[I ,J] ,P=Q ,A[I ,J]);
IF I EQ N-1; GOTO R9;
J=J+1; GOTO R8;
R9. IF I EQ M-1;GOTO R11;
I=I+1;GOTO R10;

R11. K=0;SGN =1;
Q0.R=K ;EX=1;
Q1.B=A[R ,K] ;GOTO EO;
Q2. IF D EQ 1;GOTO Q3;
IF R EQ M-1:GOTO ERR;R=R+1;GOTO Q1;
Q3.IF R EQ K ;GOTO Q5;
T=K ;SGN=-SGN;
Q4.F=A[R ,T] ;A[R ,T]=A[K ,T] ;A[K ,T]=F;
IF T EQ M ;GOTO Q5;
T=T+1;GOTO Q4;
Q5.S=0;Q8.IF K EQ S;GOTO Q7;
G=X*A[S ,K] ;REMQUO (G ,P=Q ,G);
T=K;
Q6.REMQUO (G*A[K ,T] ,P=Q ,F);F=A[S ,T]-F ;REMQUO (F ,P=Q ,A[S ,T]);
IF T EQ M ;GOTO Q7;T=T+1;GOTO Q6;
Q7. IF S EQ M-1;GOTO Q9;S=S+1;GOTO Q8;
Q9. IF K EQ M-1;GOTO Q12;K=K+1;GOTO Q0;
```

```
69.00  Q12.K=0;EX=2;Q15.B=A[K ,K] ;GOTO E0;
70.00  Q13.F=X *A[K ,M] ;REMQUO (F ,P=Q ,F);
71.00  IF F GQ  0;GOTO Q14;F=F+P;
72.00  Q14.A[K ,M]=F;
73.00  IF K EQ M-1;GOTO P16;K=K+1;GOTO Q15;
74.00  Q16.GOTO L18;

75.00  L18.I=0;R12.SGN=SGN *A[I ,I] ;REMQUO (SGN ,P=Q ,SGN);
76.00  IF I EQ M-1;GOTO F3;I=I+1;GOTO R12;F3. IF SGN GQ 0;
77.00  GOTO F1;SGN = SGN+P;GOTO F1;F1.K=M-1;F2.T=A[K ,M] *SGN;
78.00  REMQUO (T ,P=Q ,A[K ,M]);IF K EQ  0;GOTO R13;K=K-1;GOTO F2;

79.00  R13.EX=1;K=0;REMQUO (INV[L] *SGN ,P=Q ,R);
80.00  R14.Z[K]=W[L ,K] *R+V[M ,K] ;IF K EQ 9;GOTO NRM ;K=K+1;GOTO R14;
81.00  R15.K=0;R16.V[M ,K]=Z[K] ;IF K EQ 9;GOTO R17;K=K+1;GOTO R16;
82.00  R17.EX=2;I=0;R20.K=0;REMQUO (INV[L] *A[I ,M] ,P=Q ,R);
83.00  R18.Z[K]=W[L ,K] *R+V[I ,K] ;IF K EQ 9;GOTO NRM ;K=K+1;GOTO R18;
84.00  R22.K=0;R19.V[I ,K]=Z[K] ;IF K EQ 9;GOTO R21;K=K+1;GOTO R19;
85.00  R21.IF I EQ M-1;GOTO RM ;I=I+1;GOTO R20;

86.00  RM. IF L EQ 9;GOTO RR ;L=L+1;GOTO R7;

87.00  RR.I=0;RR 1 .K=9;CMP .IF V[I ,K] LS W[11 ,K] ;GOTO RR2;
88.00  IF V[I ,K] GR W[11 ,K] ;GOTO RR3;IF K EQ  0;GOTO RR3;
89.00  K=K-1;GOTO CMP;
90.00  RR3.EX=3;K=0; RR4.Z[K]=V[I ,K]-W[10 ,K];IF K EQ 9;GOTO NRM;
91.00  K=K+1;GOTO RR4;
92.00  RR6.K=0;RR5.V[I ,K]=Z[K] ;IF K EQ 9;GOTO RR1;K=K+1;GOTO RR5;
93.00  RR2.IF I EQ M ;GOTO PR ;I=I+1;GOTO RR1;

94.00  PR.I=0;
95.00  PR1.PRINT V[I ,0] ,V[I ,1] ,V[I ,2] ,V[I ,3] ,V[I ,4];
96.00  PRINT V[I ,5] ,V[I ,6] ,V[I ,7] ,V[I ,8] ,V[I ,9];
97.00  IF I EQ M ;GOTO G4;I=I+1 ;GOTO PR1;

98.00  NRM.K=0;
99.00  N1.REMQUO (Z[K] ,10000000=Q ,Z[K]) ;Z[K+1]=Q+Z[K+1];
100.00 IF K EQ 8;GOTO N2;K=K+1;GOTO N1;
101.00 N2.K=9;
102.00 N3.IF Z[K] GR  0;GOTO N6;IF Z[K] LS 0;GOTO N7;
103.00 IF K EQ  0;GOTO NN ;K=K-1;GOTO N3;
104.00 N6.K=0;N4.IF Z[K] GQ  0;GOTO N5;Z[K]=Z[K]+10000000;
105.00 Z[K+1]=Z[K+1]-1;N5.IF K EQ 8;GOTO NN;K=K+1;GOTO N4;
106.00 N7.K =0;N8.IF Z[K] LQ  0;GOTO N9;Z[K]=Z[K]-10000000;
107.00 Z[K+1]=Z[K+1]+1 ;N9.IF K EQ 8;GOTO NN ;K=K+1;GOTO N8;
108.00 NN.IF EX EQ 1;GOTO R15;IF EX EQ 2;GOTO R22;IF EX EQ 3;
109.00 GOTO RR6;

110.00 E0.F=B ;D=P ;S=1 ;X=0;
111.00 E1.REMQUO (F ,D=G ,T);
112.00 IF T EQ  0;GOTO E2;
113.00 F=D ;D=T ;T=S-G *X;
```

```
S=X ;X=T ;GOTO E1;
E2.IF D GQ  0;GOTO E3;
D=-D ;X=-X;
E3.IF X LS  0;X=P+X ;IF EX EQ 1;GOTO Q2;IF EX EQ 2;GOTO Q13;

ERR .PRINT P ,13H (DIVIDES DET A) ;GOTO RM;

G4.I=0;G3.NUM=V[I ,9] ;DEN=V[M ,9] ;K=2;
G1.TT=V[I ,10-K] ;SS=V[M ,10-K] ;NUM=10000000. *NUM+TT;
DEN=10000000. *DEN+SS ;IF K EQ 10;GOTO G2;K=K+1 ;GOTO G1;
G2.PRINT NUM/DEN ;IF I EQ M-1;STOP ;I=I+1 ;GOTO G3;
```

(Paper 71B4–240)