27
10-11-77
25
SAND77-0871

Unlimited Release

A Linear Algebraic Equation Polyalgorithm for the CDC 6600 and 7600. A User's Guide for LEQSOL()

MASTER

Richard J. Hanson, John A. Wisniewski

Prepared by Sandia Laboratories, Albuquerque, New Mexico 87115
and Livermore, California 94550 for the United States Energy Research
and Development Administration under Contract AT(29-1)-789

Printed August 1977

Sandia Laboratories

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.

Issued by Sandia Laboratories, operated for the United States Energy Research & Development Administration by Sandia Corporation.

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 & 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.

SF 1004-DF (3-75)

SAND77-0871
Unlimited Release
Printed August 1977

A Linear Algebraic Equation Polyalgorithm for
the CDC 6600 and 7600. A User's Guide for LEQSOL()

Richard J. Hanson
John A. Wisniewski
Sandia Laboratories
Albuquerque, New Mexico 87115

Abstract

A guide for persons who must compute numerical solutions of linear algebraic systems of equations is presented. Subprogram usage instructions are presented. Five often occurring FORTRAN program examples arising from linear algebraic equations are listed. These are: Solving $Ax = b$, Computing Matrix Products $A^{-1}D$ or EA^{-1} , Iterative Refinement, and Determinant Evaluation. A final example illustrates an analysis of a poorly conditioned problem. Some discussion of how to deal with constraint requirements on the solution should this occur is also presented.

The subprogram for solving the systems is LEQSOL(). The code has additional features beyond those of its Sandia Math Library counterpart SAXB(). One feature is to provide an estimate of the accuracy of the computed solution. The subprogram LEQSOL() is considerably more efficient than SAXB() for large values of N.

Printed in the United States of America

Available from

National Technical Information Service

U. S. Department of Commerce

5285 Port Royal Road

Springfield, Virginia 22161

Price: Printed Copy \$4.00; Microfiche \$2.25

24.50

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.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Table of Contents

	Page
Introduction	3
Usage Directory, Table 1	4
1. Description of Subprogram Arguments	4
2. Standard Usage, Example	6
3. Computing Matrix Products $A^{-1}D$ or EA^{-1} , Example	10
4. Iterative Refinement, Example	18
5. Determinant Evaluation, Example	24
6. Poorly Conditioned Problems	28
6.1 Accuracy Estimates with ACC Parameter, Example	28
6.2 Singular Value Analysis, SVA, Example and Discussion	29
7. Constraining the Solution Values	34
8. The Linear Algebraic Equation Polyalgorithm for the CDC 6600 and 7600	36

A Linear Algebraic Equation Polyalgorithm for
the CDC 6600 and 7600. A User's Guide for LEQSOL.

Introduction

This is a user's guide for persons who want to compute numerical solutions of square linear algebraic systems of the form

$$\begin{aligned} a_{11}x_1 + \dots + a_{1N}x_N &= b_1 \\ \vdots \\ a_{N1}x_1 + \dots + a_{NN}x_N &= b_N \end{aligned}$$

In matrix-vector notation this is written as an equation

$$\underline{Ax} = \underline{b}$$

where A is the N by N matrix $A = \{a_{ij}\}$, $\underline{x}$ is the N -vector $\underline{x} = (x_1, \dots, x_N)^T$, and $\underline{b}$ is the N -vector $\underline{b} = (b_1, \dots, b_N)^T$.

To aid the user in solving a linear equation problem that has already been formulated, we suggest that section 2, Standard Usage, Example be examined. See Table 1. The user may wish, in any case, to turn immediately to the one of the sections listed in this table that is appropriate.

The program that is described here is a polyalgorithm for solving $\underline{Ax} = \underline{b}$. This code utilizes the particular architecture of the CDC 6600 and 7600 to achieve a much higher degree of efficiency than its Sandia Mathematics Library counterpart, SAXB(), for moderate to large values of N . The supporting evidence and discussion of this is found in Section 8.

Table 1

Usage Directory

	<u>Page</u>
1. <u>Description of Subprogram Arguments</u>	4
2. <u>Standard Usage, Example</u>	6
3. <u>Computing Matrix Products $A^{-1}D$ or EA^{-1}, Example</u> . . .	10
4. <u>Iterative Refinement, Example</u>	18
5. <u>Determinant Evaluation, Example</u>	24
6. <u>Poorly Conditioned Problems</u>	28
6.1 Accuracy Estimates with ACC Parameter, Example .	28
6.2 Singular Value Analysis, SVA, Example and Discussion	29
7. <u>Constraining the Solution Values</u>	34

1. Description of Subprogram Arguments

SUBROUTINE LEQSOL(A,LDA,N,X,B,KEY,ACC,IPIVOT)

The user must dimension the following arrays which appear in the call list

$A(LDA, \geq N), B(\geq N), X(\geq N), IPIVOT(\geq N)$

This allows for the solution of a range of problems, having different sized matrices, in the given working array space.

Input:

$A(*,*)$ The leading N by N subarray of $A(*,*)$ must contain the coefficient matrix A on the initial call (KEY = 1). The parameter KEY, also an output parameter, is defined below.

 On any subsequent call (KEY = 2 or 3), for a system with the same coefficient matrix but different right hand side vectors, b, $A(*,*)$ must contain the LU decomposition information that was returned after the initial call (KEY = 1).

LDA The actual declared leading dimension of $A(*,*)$. This value must satisfy $LDA \geq N$. The condition $LDA < N$ is considered an error.

N The number of equations to be solved. A value of N outside the range $1 \leq N \leq LDA$ is considered an error.

B(*) Right hand side vector of constants, $\underline{b}$.

KEY Set KEY to one of the following values to accomplish a given task. The parameter KEY is also used as an output variable.
When KEY =

- 1 We decompose A and solve for $\underline{x}$. We also estimate the relative vector accuracy in the solution, assuming $\underline{b}$ has all digits accurate. See the output parameter ACC below.
- 2 The matrix A has been decomposed into its LU factors prior to reentry to the subprogram. With a new right hand side $\underline{b}$, we solve $A\underline{x} = \underline{b}$ for $\underline{x}$.
- 3 The matrix A has been decomposed into its LU factors prior to reentry to the subprogram. With a new right hand side vector B, we solve $\underline{x}^T A = \underline{b}^T$ for $\underline{x}$.

An input value of KEY not in the range $1 \leq KEY \leq 3$ is considered an error.

For input values of KEY = 2 or 3 the contents of A(*,*), LDA, N, and IPIVOT(*) must not be changed by the user from the values resulting from the call with KEY = 1.

Output:

X(*) Contains the N-vector solution, $\underline{x}$, to the specified system if KEY = 0 on output.

KEY Will contain a code indicating the status of the computation upon completion. An output value of KEY =

- 0 Means normal termination, no errors detected.
- 1 Means KEY was not in the range $1 \leq KEY \leq 3$.
- 2 Means N was not in the range $1 \leq N \leq LDA$.
- 3 Means a diagonal zero in U was detected and hence A is singular.

ACC If KEY = 1 on input and KEY = 0 on output, this parameter contains an estimate of the error in the components of $\underline{x}$ relative to the largest component of $\underline{x}$ in magnitude, assuming that all digits of $\underline{b}$ were accurate. With this assumption, the estimated number of decimal digits of accuracy in $\underline{x}$ can be obtained by computing

$$\text{DIGITS} = -\text{ALOG10}(\text{ACC})$$

The parameter ACC is obtained by generating an additional right hand side vector using a known solution and computing how accurately the known solution is recovered. This is a relatively crude but inexpensive estimate of the error. A more accurate estimate can be obtained using iterative refinement. See Section 4 below for an example.

The following output variables are generally of no interest to the user.

A(*,*)	Contains the decomposition information used in solving $Ax = b$ or $x^T A = b^T$. The nonunit lower triangular part of L is in the strictly lower triangular part of A(*,*). The upper-triangular part of U is in the upper triangular part of A(*,*) except for the diagonal terms of U. The nonzero diagonal terms of U are stored as reciprocals along the main diagonal of A(*,*).
B(*)	Has been overwritten with intermediate results from the computations.
IPIVOT(*)	Contains the row interchanges used in pivoting. The value $K = \text{IPIVOT}(J)$ represents the row number that was interchanged with row J, $J = 1, \dots, N$. In all cases $K \geq J$, $J = 1, \dots, N$.

2. Standard Usage, Example

In this section an example of solving a system of linear equations $Ax = b$ is presented by means of a FORTRAN program.

The correct usage of LEQSOL() to compute the solution of a single system of linear equations requires that the parameters

A(*,*), LDA, N, B(*) and KEY

be defined on input.

The solution and other useful output parameters are

X(*), KEY and ACC .

The above input and output parameters are described in detail in Section 1. The output values of the parameters KEY and ACC must be carefully

examined by the user in order to determine the usefulness of the solution returned in $X(*)$. For output values of $KEY \neq 0$ no solution is returned. For output values of $KEY = 0$ the parameter ACC contains an estimate of the error in the components of $\underline{x}$ relative to the largest component of $\underline{x}$ in magnitude, assuming that all digits of $\underline{b}$ were accurate. Output values of ACC which are large relative to machine accuracy ($7.1E-15$ on the CDC 6600 and 7600) indicates that a significant loss of precision in the solution vector $X(*)$ can be expected.

A sequence of FORTRAN statements defining the input parameters, printing the input data, calling LEQSOL() to compute the solution, and printing the output parameters is now illustrated.

```

C      SUBROUTINE EXMPL1
C      SAMPLE PROGRAM THAT DEMONSTRATES A STANDARD
C      USAGE EXAMPLE OF SUBPROGRAM LEQSOL( ) FOR SOLVING
C      LINEAR ALGEBRAIC EQUATIONS AX=B. THE SAMPLE
C      MATRIX IS 3 BY 3.
C      THE FILES FXMATH AND FXIMSL MUST BE DECLARED AS
C      LIBRARIES IN THE JOB.

C      DIMENSION A(3,3),B(3),X(3),IPIVOT(3)

C      DEFINE THE MATRIX A AND RIGHT-SIDE, B.
C      A(1,1)=0.4087
C      A(2,1)=0.3515
C      A(3,1)=0.6590

C      A(1,2)=0.1593
C      A(2,2)=0.9665
C      A(3,2)=0.9343

C      A(1,3)=0.6593
C      A(2,3)=0.6245
C      A(3,3)=0.9039

C      B(1)=0.1014
C      B(2)=0.0928
C      B(3)=0.3121

C      DEFINE THE LEADING DIMENSION OF THE ARRAY A(*,*).
C      LDA=3
C      DEFINE THE PROBLEM OR MATRIX DIMENSION.
C      N=3
C      SET KEY=1 TO SOLVE AX=B FOR X.
C      KEY=1
C      PRINT THE INPUT MATRIX A AND THE INPUT VECTOR B USING
C      IMSL MATRIX AND VECTOR PRINTING SUBPROGRAMS.
C      WRITE(6,1000)
C      CALL USWTFM(1,0,0,N,N,LDA,A)
C      CALL USWTFV(1,N,1,1,B)

C      SOLVE AX=B USING THE FXMATH SUBPROGRAM LEQSOL( ).
C      CALL LEQSOL(A,LDA,N,X,B,KEY,ACC,IPIVOT)

C      PRINT THE COMPUTED SOLUTION TO AX=B.
C      WRITE(6,1001)
C      CALL USWTFV(1,N,1,1,X)

C      WRITE THE OUTPUT VALUE OF KEY, INDICATING PROGRAM SUCCESS,
C      AND THE ESTIMATE OF THE ACCURACY OF THE SOLUTION, ACC.
C      WRITE(6,1002) KEY,ACC
C      RETURN
1000 FORMAT(44H1SAMPLE PROBLEM MATRIX AND RIGHT-SIDE VECTOR)
1001 FORMAT(///36H0COMPUTED SOLUTION OF SAMPLE PROBLEM)
1002 FORMAT(//21H0OUTPUT VALUE OF KEY=,I4,1X,16H(KEY=0 IS GOOD).,/
11X,40HESTIMATE OF COMPONENT RELATIVE ACCURACY=,1PE12.3)
END

```

SAMPLE PROBLEM MATRIX AND RIGHT-SIDE VECTOR

	1	2	3
1	.4087000000	.1593000000	.6593000000
2	.3515000000	.9665000000	.6245000000
3	.6590000000	.9343000000	.9039000000
1	.1014000000	.0928000000	.3121000000

COMPUTED SOLUTION OF SAMPLE PROBLEM

1 1.3538964425 .0551590040 -.6988082897

OUTPUT VALUE OF KEY= 0 (KEY=0 IS GOOD).
ESTIMATE OF COMPONENT RELATIVE ACCURACY= 2.842E-14

3. Computing Matrix Products $A^{-1}D$ or EA^{-1} , Example

In certain linear algebra applications, subexpressions may appear in formulas that seem to require the computation of an explicit inverse matrix. It is about three times more efficient to repose computations involving matrix products of the form $A^{-1}D$ or EA^{-1} as the solution of linear algebraic systems than to form A^{-1} and then compute the products using dot products of rows and columns.

Suppose D is an N by K real matrix. Write $D = [\underline{d}_1, \dots, \underline{d}_K]$ as its K column vectors, each having N components. We then solve the K systems

$$A \underline{x}_j = \underline{d}_j, \quad j = 1, \dots, K$$

for the column vectors $\underline{x}_j \equiv A^{-1} \underline{d}_j$. The N by K matrix

$$[\underline{x}_1, \dots, \underline{x}_K] \equiv A^{-1}D.$$

Similarly, to compute EA^{-1} where E is an L by N real matrix, we

write $E = \begin{bmatrix} \underline{e}_1^T \\ \vdots \\ \underline{e}_L^T \end{bmatrix}$ as its L row vectors, each having N components. We then

solve the L systems

$$\underline{y}_j^T A = \underline{e}_j^T, \quad j = 1, \dots, L$$

for the row vectors $\underline{y}_j^T = \underline{e}_j^T A^{-1}$. The L by N matrix

$$\begin{bmatrix} \underline{y}_1^T \\ \vdots \\ \underline{y}_L^T \end{bmatrix} = EA^{-1}.$$

An important point is that the decomposition of the matrix A , as

described in Section 8, is the most expensive part of the computation.

This decomposition is computed only with the initial call to LEQSOL(),

KEY = 1. On subsequent calls to LEQSOL(), additional systems of the form

$\underline{Ax} = \underline{b}$ or $\underline{x}^T A = \underline{b}^T$ can be computed by having KEY = 2 or KEY = 3, respectively,

without recomputing the decomposition of A.

We now present examples of computing products $A^{-1}D$ or EA^{-1} . For the matrix A we use the 3 by 3 example shown in Standard Usage, Example.


```

C      SUBROUTINE EXMPL2
C      SAMPLE PROGRAM THAT DEMONSTRATES COMPUTATION
C      OF THE MATRIX PRODUCT (INVERSE OF A)*D USING
C      SUBPROGRAM LEQSOL( ).
C      THE SAMPLE MATRIX IS THE 3 BY 3 MATRIX USED
C      TO ILLUSTRATE THE STANDARD USAGE OF LEQSOL( ).
C      THE FILES FXMATH AND FXIMSL MUST BE DECLARED AS
C      LIBRARIES IN THE JOB.
C
C      DIMENSION A(3,3),B(3),X(3),IPIVOT(3),D(3,2)
C
C      DEFINE THE MATRIX A.
C      A(1,1)=0.4087
C      A(2,1)=0.3515
C      A(3,1)=0.6590
C
C      A(1,2)=0.1593
C      A(2,2)=0.9665
C      A(3,2)=0.9343
C
C      A(1,3)=0.6593
C      A(2,3)=0.6245
C      A(3,3)=0.9039
C
C      DEFINE THE LEADING DIMENSION OF THE ARRAY A(*,*).
C      LDA=3
C      DEFINE THE PROBLEM OR MATRIX DIMENSION.
C      N=3
C      DEFINE THE 3 BY 2 MATRIX D.
C      D(1,1)=0.6311
C      D(2,1)=0.8172
C      D(3,1)=0.1573
C
C      D(1,2)=0.7075
C      D(2,2)=0.9120
C      D(3,2)=0.4145
C
C      DEFINE THE LEADING DIMENSION OF THE ARRAY D(*,*).
C      LDD=3
C
C      DEFINE THE NUMBER OF COLS IN THE MATRIX D.
C      K=2
C
C      PRINT THE MATRICES A AND D USING THE
C      IMSL MATRIX PRINTING SUBPROGRAM.
C      WRITE(6,1003)
C      CALL USWTFM(1,0,0,N,N,LDA,A)
C      CALL USWTFM(1,0,0,N,K,LDD,D)
C
C      SET KEY=1 TO COMPUTE THE DECOMPOSITION OF A AND
C      SOLVE A SYSTEM. THE FIRST COLUMN OF D IS THE
C      FIRST RIGHT HAND SIDE VECTOR
C      KEY=1
C      DO 10 J=1,K

```

```

C
C      COPY COL J OF D TO THE ARRAY B(*).
C      CALL SCOPY(N,D(1,J),1,B,1)
C
C      SOLVE THE SYSTEM WITH THE RIGHT-SIDE IN B(*).
C      CALL LEQSOL(A,LDA,N,X,B,KEY,ACC,IPIVOT)
C
C      COPY THE SOLN VECTOR TO THE J-TH COL OF D(*,*).
C      CALL SCOPY(N,X,1,D(1,J),1)
C
C      SET KEY=2 TO USE THE DECOMPOSITION OF A ALREADY
C      COMPUTED FOR THE SUBSEQUENT NEW RIGHT-SIDE VECTORS,
C      THE REMAINING COLS OF D.
C      KEY=2
C      10  CONTINUE
C
C      PRINT THE MATRIX PRODUCT (INVERSE OF A)*D.
C      WRITE(6,1004)
C      CALL USWTFM(1,0,0,N,K,LDD,D)
C      RETURN
1003  FORMAT(45H1SAMPLE PROBLEM MATRICES. COMPUTE THE INVERSE,/,
131H OF THE FIRST TIMES THE SECOND.)
1004  FORMAT(///48H THE PRODUCT OF THE ABOVE MATRICES AS INDICATED.)
      END

```

SAMPLE PROBLEM MATRICES. COMPUTE THE INVERSE
OF THE FIRST TIMES THE SECOND.

	1	2	3
1	.4087000000	.1593000000	.6593000000
2	.3515000000	.9665000000	.6245000000
3	.6590000000	.9343000000	.9039000000

	1	2
1	.6311000000	.7075000000
2	.8172000000	.9120000000
3	.1573000000	.4145000000

THE PRODUCT OF THE ABOVE MATRICES AS INDICATED.

	1	2
1	-6.9300833505	-5.9083747843
2	-.0337048028	.0384297509
3	5.2613290466	4.7264233505

SUBROUTINE EXMPL3

SAMPLE PROGRAM THAT DEMONSTRATES COMPUTATION
OF THE MATRIX PRODUCT $E \cdot (\text{INVERSE OF } A)$ USING
SUBPROGRAM LEQSOL().

THE SAMPLE MATRIX A IS THE 3 BY 3 MATRIX USED
TO ILLUSTRATE THE STANDARD USAGE OF LEQSOL().
THE FILES FXMATH AND FXIMSL MUST BE DECLARED
AS LIBRARIES IN THE JOB.

DIMENSION A(3,3),B(3),X(3),IPIVOT(3),E(2,3)

DEFINE THE MATRIX A.

A(1,1)=0.4087

A(2,1)=0.3515

A(3,1)=0.6590

A(1,2)=0.1593

A(2,2)=0.9665

A(3,2)=0.9343

A(1,3)=0.6593

A(2,3)=0.6245

A(3,3)=0.9039

DEFINE THE LEADING DIMENSION OF THE ARRAY A(*,*).

LDA=3

DEFINE THE PROBLEM OR MATRIX DIMENSION.

N=3

DEFINE THE 2 BY 3 MATRIX E.

E(1,1)=0.6311

E(2,1)=0.7075

E(1,2)=0.8172

E(2,2)=0.9120

E(1,3)=0.1573

E(2,3)=0.4144

DEFINE THE LEADING DIMENSION OF THE ARRAY E(*,*).

LDE=2

DEFINE THE NUMBER OF ROWS IN THE MATRIX E.

L=2

PRINT THE MATRICES A AND E USING THE
IMSL MATRIX PRINTING SUBPROGRAM.

WRITE(6,1005)

CALL USWTFM(1,0,0,N,N,LDA,A)

CALL USWTFM(1,0,0,L,N,LDE,E)

SET KEY=1 TO COMPUTE THE DECOMPOSITION OF A AND
SOLVE A DUMMY SYSTEM. THE FIRST RIGHT HAND
SIDE VECTOR HAS ALL ENTRIES EQUAL TO THE VALUE ONE.
THIS SOLUTION VECTOR IS IGNORED.

```

C      THE SUBSEQUENT CALLS TO LEQSOL( ) COMPUTE THE
C      MATRIX PRODUCT E*(INVERSE OF A).
C      KEY=1
C
C      FILL IN B(*) WITH THE VALUE ONE.
C      CALL SCOPY(N,1.E0,0,B,1)
C
C      COMPUTE THE DECOMPOSITION OF A.
C      CALL LEQSOL(A,LDA,N,X,B,KEY,ACC,IPIVOT)
C
C      COMPUTE E*(INVERSE OF A)
C      DO 20 I=1,L
C
C          SET KEY=3 TO USE THE DECOMPOSITION OF A ALREADY
C          COMPUTED FOR THE SUBSEQUENT NEW RIGHT-SIDE
C          VECTORS, THE ROWS OF E.
C          KEY=3
C
C          COPY ROW I OF E TO THE ARRAY B(*).
C          CALL SCOPY(N,E(I,1),LDE,B,1)
C
C          SOLVE THE SYSTEM WITH THE RIGHT-SIDE IN B(*).
C          CALL LEQSOL(A,LDA,N,X,B,KEY,ACC,IPIVOT)
C
C          COPY THE SOLN VECTOR TO THE I-TH ROW OF E(*,*).
C          CALL SCOPY(N,X,1,E(I,1),LDE)
C      20 CONTINUE
C
C      PRINT THE MATRIX PRODUCT E*(INVERSE OF A).
C      WRITE(6,1006)
C      CALL USWTFM(1,0,0,L,N,LDE,E)
C      RETURN
1005 FORMAT(45H1SAMPLE PROBLEM MATRICES.  COMPUTE THE SECOND,/
1 32H TIMES THE INVERSE OF THE FIRST.)
1006 FORMAT(////48H THE PRODUCT OF THE ABOVE MATRICES AS INDICATED.)
END

```

SAMPLE PROBLEM MATRICES. COMPUTE THE SECOND
TIMES THE INVERSE OF THE FIRST.

	1	2	3
1	.4087000000	.1593000000	.6593000000
2	.3515000000	.9665000000	.6245000000
3	.6590000000	.9343000000	.9039000000

	1	2	3
1	.6311000000	.8172000000	.1573000000
2	.7075000000	.9120000000	.4144000000

THE PRODUCT OF THE ABOVE MATRICES AS INDICATED.

	1	2	3
1	-3.0233476761	-2.8788532896	4.3682232572
2	-2.3327189144	-2.2879398703	3.7406571847

4. Iterative Refinement, Example

In order to compute solutions to linear algebraic systems $\underline{Ax} = \underline{b}$ that are accurate to full precision, a process of iterative refinement is often suggested, Ref. [3], p. 255. This technique has other uses besides obtaining an accurate solution; it can be used to accurately estimate the computed accuracy of the solution under the assumption that all digits of the input data are exact.

The iterative process consists of a loop of the form

$$\left. \begin{array}{l} \text{Solve } \underline{Ax}_1 = \underline{b} \text{ for } \underline{x}_1 \\ \\ \text{Compute } \underline{r}_i = \underline{b} - \underline{Ax}_i \\ \text{Solve for a correction } \underline{Ay}_i = \underline{r}_i \\ \text{Compute } \underline{x}_{i+1} = \underline{x}_i + \underline{y}_i \end{array} \right\} i = 1, 2, \dots$$

It is important that the vector $\underline{r}_i$ be computed with more accuracy than the storage of the data. For this reason we use the Sandia Math. Library function subprogram SDSDOT() that accumulates the dot product of vectors and adds a constant, all in double precision arithmetic. Notice that forming the vectors $\underline{r}_i$ requires storing $\underline{A}$ and $\underline{b}$ or recreating them.

The systems of equations that we solve in this iterative process all have the same coefficient matrices but different right-side vectors. This allows us to use the feature of LEQSOL() for computing solutions with new right-side vectors.

We now present an example of computing one iteration of iterative refinement. The matrix used, Ref. [3], p. 258, is 3 by 3 and has a condition number approximately 1.E+5, and hence we can expect to lose about five

digits of accuracy when computing the solution. A discussion of condition numbers and their relationships to accuracy bounds for systems of linear algebraic equations can be found in Ref. [10], Chapter 4.

In Section 6, we show how a user can obtain smaller approximate least squares solutions using the subprogram SVA(). A more thorough discussion of singular value analysis is found in Ref. [8], Chapters 25 and 26. Following the code for the sample program and the output from it and SVA(), Section 6 discusses certain interpretations of the data and what the smaller approximate solutions will do in terms of satisfying the equations.

```

C      SUBROUTINE EXMPL4
C      SAMPLE PROGRAM THAT DEMONSTRATES THE USE
C      OF ITERATIVE REFINEMENT TO COMPUTE AN ACCURATE
C      SOLUTION OF AX=B AND ESTIMATE THE ACCURACY IN THE
C      FIRST APPROXIMATION TO THE SOLUTION.
C      THE SAMPLE MATRIX IS 3 BY 3.
C      THE FILES FXMATH AND FXIMSL MUST BE
C      DECLARED AS LIBRARIES IN THE JOB.
C      DIMENSION A(3,3),B(3),X(3),IPIVOT(3)
C      DIMENSION C(3,3),F(3),XD(3)
C
C      DEFINE THE MATRIX A.
C      A(1,1)=0.876543
C      A(2,1)=0.612314
C      A(3,1)=0.317321
C
C      A(1,2)=0.617341
C      A(2,2)=0.784461
C      A(3,2)=0.446779
C
C      A(1,3)=0.589973
C      A(2,3)=0.827742
C      A(3,3)=0.476349
C
C      DEFINE THE LEADING DIMENSION OF THE ARRAY A(*,*).
C      LDA=3
C
C      DEFINE THE PROBLEM OR MATRIX DIMENSION.
C      N=3
C
C      DEFINE THE RIGHT-SIDE VECTOR IN B(*).
C      B(1)=0.4135762133
C      B(2)=0.3712631241
C      B(3)=0.5163213142
C
C      DEFINE THE LEADING DIMENSION OF THE ARRAY C(*,*).
C      LDC=3
C
C      SAVE THE MATRIX AND RIGHT-SIDE VECTOR IN C(*,*) AND F(*)
C      FOR COMPUTATION OF RESIDUALS.
C      DO 30 J=1,N
C      CALL SCOPY(N,A(1,J),1,C(1,J),1)
30    CONTINUE
C      CALL SCOPY(N,B,1,F,1)
C
C      PRINT THE MATRIX A AND THE RIGHT-
C      SIDE VECTOR B USING IMSL
C      MATRIX AND VECTOR PRINTING SUBPROGRAMS.
C      WRITE(6,1007)
C      CALL USWTFM(1,0,0,N,N,LDA,A)
C      CALL USWTFV(1,N,1,1,B)
C
C      SOLVE AX=B USING THE FXMATH SUBPROGRAM LEQSOL( ).
C      THIS IS THE FIRST APPROXIMATION TO THE SOLUTION.

```

```

C      SET KEY=1 TO SOLVE AX=B.
C      KEY=1
C      CALL LEQSOL(A,LDA,N,X,B,KEY,ACC,IPIVOT)
C      PRINT THE APPROXIMATE SOLUTION TO AX=B.
C      WRITE(6,1008)
C      WRITE(6,1012) (X(I),I=1,N)
C
C      PRINT THE OUTPUT VALUE OF KEY AND
C      THE RELATIVE ACCURACY ESTIMATE, ACC.
C      WRITE(6,1009) KEY,ACC
C
C      COMPUTE THE RESIDUAL VECTOR B-AX USING
C      DOUBLE PRECISION ACCUMULATION OF THE
C      DOT PRODUCTS. USE THE FXMATH FUNCTION SUBPROGRAM
C      SDSDOT( ) FOR THIS.
C      DO 40 I=1,N
C          B(I)=-SDSDOT(N,-F(I),C(I,1),LDC,X,1)
40      CONTINUE
C
C      SET KEY=2 TO USE THE DECOMPOSITION OF A
C      ALREADY COMPUTED FOR THE NEW RIGHT-SIDE
C      VECTOR, B-AX.
C      KEY=2
C
C      COMPUTE CORRECTION VECTOR BY SOLVING THE
C      SYSTEM AY=(B-AX). THIS SOLUTION VECTOR
C      IS IN THE ARRAY XD(*).
C      CALL LEQSOL(A,LDA,N,XD,B,KEY,ACC,IPIVOT)
C
C      PRINT THE CORRECTION VECTOR.
C      WRITE(6,1010)
C      CALL USWTFV(1,N,1,1,XD)
C
C      ESTIMATE THE RELATIVE ACCURACY OF THE FIRST
C      APPROXIMATION USING THE FXMATH FUNCTION
C      SUBPROGRAM SASUM( ).
C      ACCI=SASUM(N,XD,1)/SASUM(N,X,1)
C
C      ADD IN THE CORRECTION VECTOR TO OBTAIN
C      A MORE ACCURATE SOLUTION.
C      DO 50 I=1,N
C          X(I)=X(I)+XD(I)
50      CONTINUE
C
C      PRINT THE RELATIVE ACCURACY ESTIMATE AND
C      THE CORRECTED SOLUTION VECTOR.
C      WRITE(6,1011) ACCI
C      WRITE(6,1012) (X(I),I=1,N)
C      RETURN
C
1007 FORMAT(36H1SAMPLE PROBLEM MATRIX FOR ITERATIVE,/,
1 43H REFINEMENT EXAMPLE, AND RIGHT-SIDE VECTOR.)
1008 FORMAT(////44H FIRST APPROXIMATION TO THE SOLUTION VECTOR.)

```

```
1009 FORMAT(///21H OUTPUT VALUE OF KEY=,I4,1X,16H(KEY=0 IS GOOD).,/,  
1 36H ESTIMATE OF RELATIVE ACCURACY, ACC=,1PE12.4)  
1010 FORMAT(///22H THE CORRECTION VECTOR)  
1011 FORMAT(///40H ESTIMATE OF RELATIVE ACCURACY USING THE,/,  
1 19H CORRECTION VECTOR=,1PE12.4,/,  
2 30H THE CORRECTED SOLUTION VECTOR)  
1012 FORMAT(//4X,1H1,6(1X,F15.8))  
END
```

SAMPLE PROBLEM MATRIX FOR ITERATIVE
REFINEMENT EXAMPLE, AND RIGHT-SIDE VECTOR.

	1	2	3
1	.8765430000	.6173410000	.5899730000
2	.6123140000	.7844610000	.8277420000
3	.3173210000	.4467790000	.4763490000
1	.4135762133	.3712631241	.5163213142

FIRST APPROXIMATION TO THE SOLUTION VECTOR.

1 1604.41223072 -12126.77942154 10306.29435776

OUTPUT VALUE OF KEY= 0 (KEY=0 IS GOOD).
ESTIMATE OF RELATIVE ACCURACY, ACC= 1.8100E-10

THE CORRECTION VECTOR

1 .0000002106 -.0000015912 .0000013521

ESTIMATE OF RELATIVE ACCURACY USING THE
CORRECTION VECTOR= 1.3121E-10

THE CORRECTED SOLUTION VECTOR

1 1604.41223093 -12126.77942313 10306.29435911

5. Determinant Evaluation, Example

Subroutine LEQSOL() decomposes the matrix A into the form $PA = LU$ where P is a permutation matrix, L is unit lower triangular and U is upper triangular. Since P represents the row interchanges which were performed on A, $\det(P) = \pm 1$. Since the determinant of a triangular matrix equals the product of the diagonal terms, we have $\det(L) = 1$, $\det(U) = u_{11}u_{22}\dots u_{NN}$. So, $\det(A) = \det(P)\det(U)$.

For reasons of efficiency and convenience, we store the reciprocals of the u_{II} in $A(I,I)$. The matrix P is stored in a linear array IPIVOT(*). The value $K=IPIVOT(I)$ represents the row number that was interchanged with row I, $I=1,\dots,N$, so $\det(A)$ can be computed by

```
DET = 1.
      FOR I = 1,N
      IF (IPIVOT(I).NE.I) DET = -DET
      DET = DET*A(I,I)
DET = 1./DET
```

If the matrix A is poorly scaled, this above loop can underflow or overflow even though the final result does not underflow. A way around this is to compute $\det(A)$ by

```
DET = 1.
DETL = 0.
      FOR I = 1,N
      IF (IPIVOT(I).NE.I) SDET = -SDET
      T = A(I,I)
      IF (T.LT.O.) SDET = -SDET
      DETL = DETL + ALOG (ABS(T))
DET = SDET*EXP(-DETL)
```

We now present the determinant computation of the sample matrix used in Section 2, Standard Usage, Example

```

C      SUBROUTINE EXMPL5
C      SAMPLE PROGRAM THAT DEMONSTRATES THE
C      EVALUATION OF DETERMINANTS USING THE
C      SUBPROGRAM LEQSOL( ).
C      TWO METHODS ARE SHOWN.

C      THE FIRST METHOD SIMPLY FORMS
C      THE PRODUCT OF THE DIAGONAL ENTRIES
C      IN THE U MATRIX OF THE LU DECOMPOSITION
C      OBTAINED FOR A. WHILE THIS IS
C      MATHEMATICALLY CORRECT, A PARTIAL
C      PRODUCT CAN EXCEED THE FLOATING
C      POINT RANGE EVEN THOUGH THE DETERMINANT
C      ITSELF IS WITHIN RANGE.

C      THE SECOND METHOD USES NATURAL
C      LOGARITHMS AND EXPONENTIALS TO
C      ALLEVIATE THE PROBLEM OF BADLY
C      SCALED PARTIAL PRODUCTS THAT IS OFTEN
C      A WEAKNESS OF THE FIRST METHOD

C      THE SAMPLE MATRIX IS THE 3 BY 3 MATRIX USED
C      TO ILLUSTRATE THE STANDARD USAGE OF LEQSOL( ).
C      THE FILES FXMATH AND FXIMSL MUST BE DECLARED AS
C      LIBRARIES IN THE JOB.

C      DIMENSION A(3,3),B(3),X(3),IPIVOT(3)
C      DEFINE THE MATRIX A.
C      A(1,1)=0.4087
C      A(2,1)=0.3515
C      A(3,1)=0.6590

C      A(1,2)=0.1593
C      A(2,2)=0.9665
C      A(3,2)=0.9343

C      A(1,3)=0.6593
C      A(2,3)=0.6245
C      A(3,3)=0.9039

C      DEFINE THE LEADING DIMENSION OF THE ARRAY A(*,*).
C      LDA=3
C      DEFINE THE PROBLEM OR MATRIX DIMENSION.
C      N=3

C      DEFINE A DUMMY RIGHT-SIDE VECTOR OF ONES.
C      CALL SCOPY(N,1.E0,0,B,1)
C      PRINT THE MATRIX A.
C      WRITE(6,1013)
C      CALL USWTFM(1,0,0,N,N,LDA,A)

C      SET KEY=1 TO COMPUTE THE DECOMPOSITION OF A.
C      KEY=1

C      COMPUTE THE DECOMPOSITION.

```



```

CALL LEQSOL(A,LDA,N,X,B,KEY,ACC,IPIVOT)
C
C
C   COMPUTE THE DETERMINANT, FIRST METHOD.
DET=1.E0
      DO 60 J=1,N
      IF(IPIVOT(J).NE.J) DET=-DET
      DET=DET*A(J,J)
60    CONTINUE
      DET=1.E0/DET
C
C   PRINT DETERMINANT VALUE, FIRST METHOD.
WRITE(6,1014) DET
C
C   COMPUTE THE DETERMINANT, SECOND METHOD.
SDET=1.E0
DETL=0.E0
      DO 70 J=1,N
      IF(IPIVOT(J).NE.J) SDET=-SDET
      T=A(J,J)
      IF(T.LT.0.E0) SDET=-SDET
      DETL=DETL+ALOG(ABS(T))
70    CONTINUE
      DET=SDET*EXP(-DETL)
C
C   PRINT DETERMINANT VALUE, SECOND METHOD.
WRITE(6,1015) DET
RETURN
1013 FORMAT(26HISAMPLE PROBLEM MATRIX FOR,/,
1 24H DETERMINANT EVALUATION.)
1014 FORMAT(////33H VALUE OF DETERMINANT, METHOD 1,=,1PE20.13)
1015 FORMAT(////33H VALUE OF DETERMINANT, METHOD 2,=,1PE20.13)
END

```

SAMPLE PROBLEM MATRIX FOR
DETERMINANT EVALUATION.

	1	2	3
1	.4087000000	.1593000000	.6593000000
2	.3515000000	.9665000000	.6245000000
3	.6590000000	.9343000000	.9039000000

VALUE OF DETERMINANT, METHOD 1,=-6.9875128020003E-02

VALUE OF DETERMINANT, METHOD 2,=-6.9875128020004E-02

6. Poorly Conditioned Problems

The solution to $\underline{Ax} = \underline{b}$ may, in many applications, may be different than expected because the problem is ill-posed. A system is considered ill-posed if the solution does not have a small change when the data matrix $[A:b]$ is subjected to a small change. The user is often made aware of this situation by obtaining a large solution vector, relative to the size of the numbers in A and b, or by observing that the estimate of accuracy, ACC, is large. Roughly, when $ACC \div 1.E-14$ on the CDC 6600 and 7600, the solution is as accurate as can be computed with the subprogram LEQSOL(). If, for example, $ACC \geq 1.E-3$ the solution $\underline{x}$ returned by LEQSOL is highly inaccurate, and a method for solving the system $\underline{Ax} = \underline{b}$ which takes into account the fact that the matrix A is ill-posed may be more appropriate.

6.1. Accuracy Estimates with ACC Parameter, Example

We will use the example of Section 4. Iterative Refinement, Example.

In the output for this example the value of $ACC = 1.81E-10$. This is, in fact, a very good estimate of the actual relative accuracy value obtained using the single iteration of iterative refinement, $1.31E-10$. The value of ACC computed shows that about 4-5 decimal places were lost in the computation. The results obtained are as accurate as we could expect. The matrix A used in the example has a condition number of $1.E5$ which indicates that we could expect to lose about 5 digits of accuracy when solving the system $\underline{Ax} = \underline{b}$ using fixed precision floating point arithmetic. A discussion of condition numbers and their relationships to accuracy bounds for systems of linear algebraic equations can be found in Ref. [10], Chapter 4.

What is often more serious than the loss of accuracy in the

solution is that the solution vector is very large compared with the data. Due to errors in the data, or an inaccurate model, there is often no reason to obtain a very accurate solution to a linear system of equations. We may, very often, be just as happy with small solutions that are approximate solutions. What follows below is an example and discussion of using singular value analysis, a technique for generating small, approximate solutions.

6.2 Singular Value Analysis, SVA, Example and Discussion

```

C      SUBROUTINE EXMPL6
C      SAMPLE PROGRAM THAT DEMONSTRATES THE
C      USE OF SUBPROGRAM SVA( ) TO ANALYZE
5      C      A SYSTEM OF LINEAR ALGEBRAIC EQUATIONS
C      AX=B.  THE SUBPROGRAM SVA( ) PRINTS
C      A SEQUENCE OF APPROXIMATE LEAST SQUARES
C      SOLUTIONS TO AX=B.  THE CORRESPONDING RESIDUAL
C      VECTOR LENGTHS ARE ALSO AVAILABLE.
10     C
C      THE LIBRARIES FXMATH AND FXIMSL MUST
C      BE DECLARED IN THE JOB.
C
C      DIMENSION A(3,3),B(3),SING(9)
C      DATA IBLANK/1H /
15     C      DEFINE THE MATRIX A.
C      A(1,1)=0.876543
C      A(2,1)=0.612314
C      A(3,1)=0.317321
C
20     C      A(1,2)=0.617341
C      A(2,2)=0.784461
C      A(3,2)=0.446779
C
C      A(1,3)=0.589973
25     C      A(2,3)=0.827742
C      A(3,3)=0.476349
C
C      DEFINE THE LEADING DIMENSION OF THE ARRAY A(*,*).
C      LDA=3
30     C
C      DEFINE THE PROBLEM OR MATRIX DIMENSION.
C      N=3
C
C      DEFINE THE RIGHT-SIDE VECTOR IN B(*).
35     C      B(1)=0.4135762133
C      B(2)=0.3712631241
C      B(3)=0.5163213142
C
C      PRINT THE MATRIX A AND RIGHT-
40     C      SIDE VECTOR B USING IMSL
C      MATRIX AND VECTOR PRINTING SUBPROGRAMS.
C      WRITE(6,1016)
C      CALL USWTFM(1,0,0,N,N,LDA,A)
C      CALL USWTFV(1,N,1,1,B)
45     C
C      NO COL SCALING IS USED..
C      ISCALE=1
C      COMPUTE THE SINGULAR VALUE ANALYSIS DATA..
50     C      CALL SVA(A,LDA,N,N,N,B,SING,IBLANK,ISCALE,D)
C      RETURN
1016  FORMAT(26H1SAMPLE PROBLEM MATRIX AND,/
147H  RIGHT-SIDE VECTOR FOR SINGULAR VALUE ANALYSIS.)
C      END

```

SAMPLE PROBLEM MATRIX AND
RIGHT-SIDE VECTOR FOR SINGULAR VALUE ANALYSIS.

	1	2	3
1	.8765430000	.6173410000	.5899730000
2	.6123140000	.7844610000	.8277420000
3	.3173210000	.4467790000	.4763490000

1	.4135762133	.3712631241	.5163213142
---	-------------	-------------	-------------

SINGULAR VALUE ANALYSIS OF THE LEAST SQUARES PROBLEM, $A \cdot X = B$, SCALED AS $(A \cdot D) \cdot Y = B$

M = 3, N = 3, MDATA = 3

SCALING OPTION NO. 1. D IS THE IDENTITY MATRIX.

V-MATRIX OF THE SINGULAR VALUE DECOMPOSITION OF $A \cdot D$.
(ELEMENTS OF V SCALED UP BY A FACTOR OF 10^{**4})

	COL 1	COL 2	COL 3
1	-5728.	8136.	1003.
2	-5735.	-3103.	-7582.
3	-5857.	-4918.	6443.

INDEX	SING. VALUE	P COEF	RECIP. S.V.	G COEF	G**2	C.S.S.	N.S.R.C.S.S.
0						5.7547E-01	4.3798E-01
1	1.9001	-3.7238E-01	5.2630E-01	-7.0754E-01	5.0062E-01	7.4851E-02	1.9346E-01
2	.2999	-2.2343E-01	3.3347E+00	-6.6999E-02	4.4889E-03	7.0362E-02	2.6526E-01
3	.1658E-04	1.5995E+04	6.0301E+04	2.6526E-01	7.0362E-02	0.	0.

INDEX	YNORM	RNORM	LOG10(YNORM)	LOG10(RNORM)
0	0.	.75860E+00	-1000.00000	-.11999
1	.37238E+00	.27359E+00	-.42902	-.56290
2	.43426E+00	.26526E+00	-.36225	-.57633
3	.15995E+05	0.	4.20399	-1000.00000

NORMS OF SOLUTION AND RESIDUAL VECTORS FOR A RANGE OF VALUES OF THE LEVENBERG-MARQUARDT PARAMETER, LAMBDA.

LAMBDA	YNORM	RNORM	LOG10(LAMBDA)	LOG10(YNORM)	LOG10(RNORM)
.19001E+02	.36873E-02	.75207E+00	1.27877	-2.43329	-.12374
.84297E+01	.18006E-01	.72679E+00	.92581	-1.74457	-.13859
.37399E+01	.76411E-01	.62535E+00	.57286	-1.11684	-.20387
.16592E+01	.21139E+00	.41022E+00	.21990	-.67492	-.38699
.73612E+00	.32534E+00	.28669E+00	-.13305	-.48766	-.54259
.32658E+00	.37586E+00	.26851E+00	-.48600	-.42498	-.57105
.14489E+00	.41216E+00	.26559E+00	-.83896	-.38493	-.57578
.64281E-01	.42893E+00	.26528E+00	-1.19191	-.36762	-.57630
.28519E-01	.43320E+00	.26526E+00	-1.54487	-.36331	-.57633
.12652E-01	.43491E+00	.26526E+00	-1.89782	-.36160	-.57633
.56133E-02	.45611E+00	.26526E+00	-2.25078	-.34093	-.57634
.24904E-02	.83162E+00	.26525E+00	-2.60373	-.08008	-.57635
.11049E-02	.36287E+01	.26520E+00	-2.95669	.55975	-.57643
.49018E-03	.18292E+02	.26495E+00	-3.30964	1.26225	-.57683

.21747E-03	.92475E+02	.26372E+00	-3.66260	1.96602	-.57885
.96482E-04	.45899E+03	.25765E+00	-4.01555	2.66180	-.58898
.42805E-04	.20875E+04	.23064E+00	-4.36851	3.31962	-.63707
.18991E-04	.69203E+04	.15050E+00	-4.72146	3.34012	-.82247
.84252E-05	.12714E+05	.54421E-01	-5.07442	4.10427	-1.26423
.37379E-05	.15222E+05	.12825E-01	-5.42737	4.18247	-1.89194
.16583E-05	.15837E+05	.26263E-02	-5.78033	4.19967	-2.58065

SEQUENCE OF CANDIDATE SOLUTIONS, X

	SOLN 1	SOLN 2	SOLN 3
1	.21328982E+00	.31521768E-01	.16044122E+04
2	.21356075E+00	.28288970E+00	-.12126779E+05
3	.21809371E+00	.32797003E+00	.10366294E+05

Discussion of Singular Value Analysis for the Example

The singular value decomposition, Ref. [8], Chapter 18, is another way to decompose the matrix A. The decomposition has the form $AV = US$, where U and V are N by N orthogonal matrices and S is diagonal. In this particular example, the last column of the V matrix is

$$\underline{v} = \begin{pmatrix} 0.1003 \\ -0.7582 \\ 0.6443 \end{pmatrix}$$

The vector $\underline{y} \equiv A\underline{v}$ has length $\doteq 1.66E-4$ and is thus close to the zero vector. Since the vector $\underline{v}$ is not close to the zero vector, but $A\underline{v}$ is close to the zero vector we see that the matrix A is nearly singular.

Due to the near singularity of the matrix A, the third candidate solution, which is found on the third page of this example, is quite large. In particular,

$$\underline{x}_3 \doteq \begin{pmatrix} 1604 \\ -12127 \\ 10306 \end{pmatrix}$$

This solution is the same solution that is returned by subprogram LEQSOL. If we desire an approximate solution that is about the same size as the data, the second candidate solution,

$$\underline{x}_2 \doteq \begin{pmatrix} 0.0315 \\ 0.2829 \\ 0.3280 \end{pmatrix}$$

is quite reasonable. This solution is reasonable since it has length $\doteq 0.43$ compared to the length of the right-side vector $\underline{b}$, $\doteq 0.75$. In addition it is a reasonable approximate solution since the residual

vector length (of $\underline{r} = \underline{b} - \underline{Ax}_2$) $\doteq 0.27$ is fairly small.

Another often-used technique for regularization (making solutions more properly behaved by keeping the length of the solution vector $\underline{x}$ small) is to solve the augmented least squares problem

$$\begin{bmatrix} A \\ \lambda I \end{bmatrix} \underline{x} \cong \begin{bmatrix} \underline{b} \\ 0 \end{bmatrix}.$$

This is called Levenberg-Marquardt stabilization, loc. cit.

To achieve the same approximate solution and residual vector length as with the second candidate solution we can take $\lambda \doteq 0.12\text{E-}1$. Notice from the second page of this example that the residual vector depends very weakly on λ as $\lambda \rightarrow 0$. In fact, SVA() shows that the solution of the augmented least squares problem will have a length exceeding 15,000 to achieve a residual vector of length $\doteq 0.01$. There are no solutions of smaller length that can achieve this residual vector length, loc. cit., p. 193.

7. Constraining the Solution Vector

Many applications where linear equations $\underline{Ax} = \underline{b}$ occur involve solution vectors $\underline{x}$ which must satisfy certain constraints. These constraints usually correspond to physical knowledge of the parameters. If a solution to $\underline{Ax} = \underline{b}$ does not satisfy a bound or constraint that the user knows must be present in the physical problem, there is no way that one can generally solve $\underline{Ax} = \underline{b}$, get each equation satisfied, and still have the variables satisfy the constraints. What one can do is to solve $\underline{Ax} = \underline{b}$ in a least squares sense subject to equality constraints $\underline{Ex} = \underline{f}$ and inequality constraints $\underline{Gx} \geq \underline{h}$. The subprograms to use for these computations are

SLSEI() and NNLS(), Ref. [9,8].

We'll now give three examples of how to formulate the equality and inequality constraint matrices.

Constrained Example 1

Solve $\underline{Ax} = \underline{b}$.

Each $\underline{x}$ component is to be non-negative.

Solution Use NNLS().

Constrained Example 2

Solve $\underline{Ax} = \underline{b}$.

Each $\underline{x}$ component is to be non-negative and $\sum_{i=1}^N x_i \leq 1$.

Solution Use SLSEI() to solve $\underline{Ax} = \underline{b}$

subject to $\underline{Gx} \geq \underline{h}$ where

$$G_{(N+1) \times N} = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \\ -1 & \dots & -1 \end{bmatrix}, \quad \underline{h} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix}$$

Constrained Example 3

Solve $\underline{Ax} = \underline{b}$

Each $\underline{x}$ component is to be non-negative and $\sum_{i=1}^N x_i = 1$.

Solution Use SLSEI() to solve $\underline{Ax} = \underline{b}$

subject to $\underline{Gx} \geq \underline{h}$, $\underline{Ex} = \underline{f}$ with

$$G_{N \times N} = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix}, \quad \underline{h} = \underline{0},$$

$$E_{1 \times N} = [1 \dots 1], \quad \underline{f} = [1].$$

8. The Linear Algebraic Equation Polyalgorithm for the CDC 6600 and 7600

8.1 Introduction

In Sandia's working environment the solution of square linear systems of algebraic equations is a significant part of the incidence of the mathematical library usage. For example, during the period Jan. 1, 1976 - Dec. 31, 1976 calls to the linear equation solvers comprised 8.2% of approximately 35,500 library accesses of subprograms. These figures were obtained using the monitoring techniques of Bailey and Jones, Ref. [6]. The 8.2% value for linear equations is probably an underestimate of that library subprogram usage. Thus a very efficient and reliable linear equation solver is needed in this environment.

Parlett and Wang, Ref. [5], show that it is not possible to write a universally efficient FORTRAN code for solving linear algebraic equations. This is mainly due to the quality of the object program output by a compiler from the FORTRAN source code.

We utilized coding techniques and assembly language subprograms, Ref. [1], that achieved near optimal efficiency on the CDC 6600 and 7600 in writing this program, LEQSOL(). By using the FORTRAN versions of the assembly language subprograms, loc. cit., the program LEQSOL() is portable. Since it is written to achieve near optimal efficiency only on the CDC 6600 and 7600, its performance on other machines may be suboptimal.

What we will describe is an algorithm for linear equation solving on the CDC 6600 and 7600 that utilizes the particular architecture of these machines to achieve a high degree of efficiency. The algorithm design is motivated by the desire to have the program perform efficiently for small and large values of N.

8.2 The CDC 6600 and 7600 Polyalgorithm to Solve $\underline{Ax} = \underline{b}$

Remark The value of N is the dimension of the N by N matrix A. All arithmetic is done in single precision, ~14D. If the machine is the CDC 6600 use one of methods 1., 2. or 3. If the machine is the CDC 7600 use one of methods 4. or 5.

1. If $1 \leq N \leq 28$ solve $\underline{Ax} = \underline{b}$ using Gaussian elimination with partial pivoting, Ref. [2]. The inner loop is in-line FORTRAN as in loc. cit.
2. If $29 \leq N \leq 47$ solve $\underline{Ax} = \underline{b}$ using Gaussian elimination with partial pivoting, Ref. [2]. The inner loop is an assembly language coded subprogram for the elementary vector operation $\underline{y} := \underline{ax} + \underline{y}$.
3. If $N \geq 48$ solve $\underline{Ax} = \underline{b}$ using the Doolittle direct decomposition of A, Ref. [3], p. 225. The inner loop is an assembly language coded dot product operation subprogram $w := \underline{x}^T \underline{y}$.
4. If $1 \leq N \leq 183$ solve $\underline{Ax} = \underline{b}$ using Gaussian elimination with partial pivoting, Ref. [2]. The inner loop is a FORTRAN coded subprogram for the elementary vector operation $\underline{y} := \underline{ax} + \underline{y}$ using a specially coded loop.
5. If $N \geq 184$ use the same method as in 3.

The essential part of the subprogram used in method 4 consists of the loop

```
DO 10 I=M,N,4
Y(I)=A*X(I)+Y(I)
Y(I+1)=A*X(I+1)+Y(I+1)
Y(I+2)=A*X(I+2)+Y(I+2)
Y(I+3)=A*X(I+3)+Y(I+3)
10 CONTINUE
```

The CDC 7600 FTN FORTRAN compiler, OPT = 2, generates extremely efficient object code that utilizes this machine's architecture. Essentially no

improvement could be made by coding this subprogram in assembly language.

The LINPACK project, Ref. [4], is proposing an algorithm similar to method 4 of the above polyalgorithm for all values of N and for all machines. This is certainly a reasonable choice if only one method is to be used.

The subprogram LEQSOL() detects whether the machine is the CDC 6600 or CDC 7600 by the simple device of computing the number of times a given task can be done in one millisecond of computer time. This is compared to prestored values for both machines. The minimum error in the differences is used to detect the machine on which we are running. This test is made only once for each job usage of LEQSOL(). The test has proved to be completely reliable in practice. A misclassification here would result in suboptimal performance but would still yield correct results.

8.3 Performance of the Polyalgorithm Code LEQSOL() on the CDC 6600

The floating point operation count for each of the methods 1. - 5. is asymptotically $N^3/3$. The tradeoff points are caused by such factors as subroutine linkage overhead, index arithmetic, and the fetching and storing of quantities. For example the method of 3. requires $N^3/3$ fetches but N^2 stores. The methods of 1. and 2. require $N^3/3$ fetches and stores, Ref. [5].

Based on the operation count, each of methods 1., 2. and 3. have their average executions times accurately modeled by the cubic polynomial

$$t_i(N) = a_i N^3 + b_i N^2 + c_i N + d_i, \quad i = 1, 2, 3.$$

The coefficients were obtained by least squares fitting to actual timing data using five runs with $N = 5$ (5) 100.

The coefficients were obtained by least squares fitting to actual timing data using five runs with $N = 5$ (5) 100.

To emphasize the behavior of the $t_i(N)$ for the larger values of N , we plotted the functions $t_i(N)/N^3$ for $i = 1, 2, 3$, in Figure 1.

The tradeoff values of $N = 28$ and $N = 47$ were obtained experimentally by plotting the function

$$\min(t_i(N)/N^3)$$

and observing the discontinuities in the first differences.

Some ratios of computing times for the individual methods compared to the polyalgorithm are now given. All timings were done using the FTN FORTRAN compiler with $OPT = 2$. This results in the highest level of object code optimization.

- The time for method 1 is 1.71 times that of the polyalgorithm time at $N = 100$.
- The time for method 2 is 1.33 times that of the polyalgorithm time at $N = 100$.
- The time for method 1 is 2.83 times that of the polyalgorithm for sufficiently large values of N .

8.4 Performance of the Polyalgorithm Code LEQSOL() on the CDC 7600

In much the same manner as for the CDC 6600, the tradeoff value of $N = 184$ was determined. At this value of N the polyalgorithm switches from method 4. to method 5.

Based on individual timing runs the following ratios of computing times were observed.

- The time for method 1 is 1.90 times that of the polyalgorithm at $N = 100$.
- The time for method 1 is 2.0 times that of the polyalgorithm at $N = 200$.

8.5 Comparison of the Polyalgorithm Code LEQSOL() with Mathlib Code SAXB()

The basic techniques for solving $\underline{Ax} = \underline{b}$ with LEQSOL() and SAXB() are quite similar. The methods used in LEQSOL() are given in Section 8.2. The method used in SAXB() is Gaussian elimination with weighted row pivoting. The entire code is written with in-line Fortran.

The subprogram LEQSOL() does no row or column scaling and this may occasionally be necessary. Usually a large value of ACC indicates an ill-posed system. Relevant formulas and some guidelines are found in Ref. [7], Chap. 11, for example. For a general purpose subprogram there seems little point in always doing column or row scaling, and this led to our choice of not providing such a feature in LEQSOL().

A large value of ACC, greater than 0.001, for example, may indicate that column or row scaling will yield a more accurate solution.

8.5.1 New Algorithmic Features

The features that have been added to LEQSOL() over those of SAXB() are:

- Estimate of accuracy of the solution. This is done by using a known vector $\hat{x} = (1, \dots, 1)^T$, to generate $\hat{b} = A\hat{x}$. The accuracy of this system, with $\hat{b}$ as the right-hand side vector, is used as the estimate of accuracy of $\underline{Ax} = \underline{b}$.
- Solve systems of the form $\underline{x}^T A = \underline{b}^T$, ($A^T \underline{x} = \underline{b}$), as well as solve systems of the form $\underline{Ax} = \underline{b}$, without transposing A and recomputing the decomposition.

The subprogram SAXB() allows for several right-hand side vectors at one entry, whereas LEQSOL() has only one right-hand side vector per entry. Since both routines allow for reentry

without redecomposition, there seems to be little advantage to either design.

8.5.2 Timing Performance on the CDC 6600 and CDC 7600

For values of $N = 5(5)100$ we timed solutions to N by N systems $Ax = b$ using LEQSOL() and SAXB(). The results were averaged over five separate runs. Runs were also made for $N = 200$ on the CDC 7600. Since the operation count for all of these methods, in both codes, is asymptotically $N^3/3$, we have divided each timing result by N^3 . These results are shown in Figures 2 and 3. The plots there are shown with linear interpolation between the discrete data points.

Some ratios of computing times for the two subprograms are now given. Both codes were compiled using the FTN FORTRAN compiler with OPT = 2.

- The time for SAXB() is 1.76 times that of LEQSOL() at $N = 100$ on the CDC 6600.
- The time for SAXB() is 1.67 and 2.04 times that of LEQSOL(), at $N = 100$ and 200 respectively, on the CDC 7600.
- The absolute time differences amount to a savings of 0.7 secs. for $N = 100$, per system, on the CDC 6600, and for $N = 200$ a savings of 1.0 secs., per system, on the CDC 7600.
- For those smaller values of N where SAXB() outperforms LEQSOL(), the absolute time differences are 1-3 milliseconds.

The fact that LEQSOL() takes this additional time, for small N , is due primarily to an extra $1.5N^2$ operations needed to estimate the accuracy of the solution.

8.6 Discussion

For Sandia Labs. Math Library users, we believe that the subprogram LEQSOL() is superior to SAXB(). It provides two new important features beyond those of SAXB(), and is progressively more efficient for the larger values of N. For the smaller values of N these new features are generally not costly in computer time.

References

1. Lawson, C. L., Hanson, R. J., Kincaid, D. R., Krogh, F. T., "Basic Linear Algebra Subprograms for FORTRAN Usage," submitted to Trans. Math. Software, June, (1977).
2. Moler, C. B., "Matrix Computations with FORTRAN and Paging," Comm. ACM, Vol. 15, No. 4, p. 268, April, (1972).
3. Wilkinson, J. H., The Algebraic Eigenvalue Problem, Clarendon Press, (1965).
4. Moler, C. B., et al. LINPACK. Argonne National Laboratories.
5. Parlett, B. N., Wang, Y., "The Influence of the Compiler on the Cost of Mathematical Software--in Particular on the Cost of Triangular Factorization," Trans. Math. Software, Vol. 1, No. 1, p. 35, March, (1975).
6. Bailey, C. B., Jones, R. E., "Usage and Argument Monitoring of Mathematical Library Routines," Trans. Math. Software, Vol. 1, No. 3, p. 196, September, (1975).
7. Forsythe, G. E., and Moler, C. B., "Computer Solution of Linear Algebraic System," Prentice-Hall, (1967).
8. Lawson, C. L., and Hanson, R. J., "Solving Least Squares Problems," Prentice-Hall, (1974).
9. Hanson, R. J., "Selected Algorithms for Least Squares Computations. A User's Guide," SAND 77-1090.
10. Stewart, G. W., "Introduction to Matrix Computations," Academic Press, (1973).

Figure 1: CDC 6600 Timing for Linear Equation Solution of Systems
Comparing Three Methods and a Polyalgorithm

Figure 2: CDC 660C Timing for Linear Equation Solution of Systems Comparing the Codes LEQSOL () and SAXB ()

Figure 3: CDC 7600 Timing for Linear Equation Solution of Systems Comparing the Codes LEQSOL() and SAXB()

Distribution:

2613 K. Haskell
2613 R. E. Jones
2613 M. R. Scott
2613 H. R. Watts
5000 A. Narath
5100 J. K. Galt
5110 F. L. Vook
5120 G. J. Simmons
5121 H. T. Davis
5121 G. P. Steck
5121 R. J. Thompson
5122 D. E. Amos
5122 R. J. Hanson (10)
5122 L. F. Shampine
5122 J. A. Wisniewski (10)
5130 G. A. Samara
5150 J. E. Schirber
5160 W. Herrmann
5200 E. H. Beckner
5400 A. W. Snyder
5700 J. H. Scott
5800 R. S. Claassen
8266 E. A. Aas (2)
8325 R. E. Huddleston
8325 T. H. Jefferson
3151 W. L. Garner (3)
For ERDA/TIC (Unlim. Release)
3171-1 R. Campbell (25)
For ERDA/TIC