37
3/21/78
25 to WT
SAND77-1090
Unlimited Release

Selected Algorithms for Least Squares Computations — A Users Guide

M

MASTER

Richard J. Hanson

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

Printed March 1978

Sandia Laboratories

SE 2900 Q(7-73)

NOTICE

PORTIONS OF THIS REPORT ARE ILLEGIBLE. It
has been reproduced from the best available
copy to permit the broadest possible avail-
ability.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

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

DISCLAIMER

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

Issued by Sandia Laboratories, operated for the United States
Department of Energy 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 Department of Energy, 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(11-77)

SAND77-1090
Unlimited Release
Printed March 1978

SELECTED ALGORITHMS FOR LEAST SQUARES COMPUTATIONS M
A USER'S GUIDE

Richard J. Hanson
Sandia Laboratories
Albuquerque, New Mexico 87115

Abstract

This report is intended to be a practical computational guide for persons who must solve linear least squares problems.

A list of high-quality software, readily available to Sandia Laboratories personnel, is given. Six commonly occurring computational problems are illustrated by the inclusion of FORTRAN programs that utilize an appropriate FORTRAN subprogram of this list.

These problems are identified as Standard Usage (Section 2.), Sequential Accumulation (Section 5.), Normal Equations (Section 8.), Equality and Inequality Constraints (Section 9.), Constrained Curve Fitting by a Cubic Polynomial (Section 10.), and Ill-Posed Problems (Section 11).

Other problem areas discussed are identified as Formulating Curve Fitting Problems (Section 3), Problems with Many Columns (Section 6.), and Banded Least Squares Problems (Section 7.).

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: \$4.50 , Microfiche \$3.00

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, 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

Section Number	Page Number
1. Introduction	3
2. Standard Usage, Example ⁽¹⁾	9
3. Formulating Curve Fitting Problems	12
4. Fitting a Cubic Polynomial to Data	14
5. Problems with Many Rows, Sequential Accumulation, Example	16
6. Problems with Many Columns	24
7. Banded Least Squares Problems	26
8. Normal Equations Method, Example	28
9. Equality and Inequality Constraints, Example	32
10. Constrained Curve Fitting by a Cubic Polynomial, Example	38
11. Ill-Posed Problems, Example (Use of Singular Value Analysis)	44
 Table 1 Sandia Laboratories Least Squares Subprograms, Documentation, and Location (October, 1977)	 8
Table 2 Sample Discrete Data for Cubic Fitting	14
Figure 1 Unconstrained Cubic Polynomial Fit to Discrete Data of Table 2	41
Figure 2 Constrained Cubic Polynomial Fit to Discrete Data of Table 2	42
 References	 49
Appendix User Instructions for Subprogram SLSEI()	50

(1) Each section with 'Example' in the title contains an illustrative FORTRAN program computing the solution to a sample problem.

Selected Algorithms for Least Squares Computations -- A User's Guide

1. Introduction

This report is concerned with guiding a person who must formulate and solve linear least squares problems. Such a problem is written here as

$$\underline{A}\underline{x} = \underline{b}$$

The dimensions of the matrix A are m by n . No assumption about the relative size of m and n is made. If the user has such a least squares problem already formulated, it may be appropriate to turn directly to Section 2, Standard Usage, Example.

A number of commonly occurring computational problems are discussed in this report. Each section heading that contains the word 'Example' includes a FORTRAN program computing the solution to a sample problem. The reader may examine the Table of Contents for an appropriate title and then turn directly to that section.

We now turn to a general discussion of the least squares problem. In case there are fewer than m linearly independent column vectors in the matrix A , there will generally exist no solution $\underline{x}$ such that $\underline{A}\underline{x} = \underline{b}$. But, in fact, there will always exist solutions, $\hat{\underline{x}}$, that minimize the sum of squares of residuals

$$\rho(\underline{x}) = \underline{r}^T \underline{r}, \quad \underline{r} = \underline{b} - \underline{A}\underline{x}$$

The value $\rho^{\frac{1}{2}}(\hat{\underline{x}}) = \|\underline{b} - \underline{A}\hat{\underline{x}}\|$ is the residual vector length.

If there are fewer than n linearly independent columns in A then there will be a solution of minimum length. Let us call this unique solution $\underline{x}^+$. The set of all solutions is given by $\underline{x} = \underline{x}^+ + \underline{y}$, where the vector $\underline{y}$ is any

solution of $\underline{A}\underline{y} = \underline{0}$.

In case $m = n$ and A has n linearly independent column vectors and no other conditions, such as linear constraints, are to be placed on the solution, then we have a square nonsingular system. The reader can efficiently solve such systems using the subprogram LEQSOL(). A writeup and supporting documentation is found in Ref. [2]. The FORTRAN source and object code for LEQSOL() are on the personal files of the author.

There are often additional requirements placed on solutions of least squares problems such as linear equality and inequality constraints. These are written as matrix-vector constraints

$$\underline{E}\underline{x} = \underline{f} \quad (E \text{ is } m_E \text{ by } n)$$

and

$$\underline{G}\underline{x} \geq \underline{h} \quad (G \text{ is } m_G \text{ by } n)$$

There are a few basic facts to realize regarding least squares problems with such constraints.

First, there is no guarantee that the solution to $\underline{A}\underline{x} = \underline{b}$ will satisfy either of these constraint conditions. The person must include physical (or other) constraints as part of the problem statement.

Second, if the solution vector satisfies the inequalities $\underline{G}\underline{x} \geq \underline{h}$ then k of these will be satisfied as equalities and the remainder will be strict inequalities. Let $\underline{H}\underline{x} = \underline{d}$ denote this set of k equality constraints. The k by $n + 1$ matrix $[\underline{H}:\underline{d}]$ is some subset of the matrix $[\underline{G}:\underline{h}]$. Just which subset is the problem. The reader is cautioned against trying an exhaustive test of all subsets of k rows of G , $k = 1, \dots, m_G$. This approach may require on the order of 2^{m_G} operations. For large m_G this is prohibitively expensive. Good computational methods, such as those found in the subprograms SLSEI(),

NNLS(), LSEI() and WNNLSE() listed in Table 1, solve the inequality constrained least squares problem with a virtual order of $(m_E + m + m_G)n^2$ operations.

Third, even when $m = n$ there will generally exist no solution, $\tilde{x}$, such that $G\tilde{x} \geq h$ and $A\tilde{x} = b$.

Unless the set of all x satisfying $Ex = f$ and $Gx \geq h$ is contradictory (or empty), there will exist least squares solutions minimizing $\rho(x)$ and satisfying the constraints.

A user with a constrained least squares problem may find it appropriate to turn directly to Section 9, Equality and Inequality Constraints, Example.

Most of the mathematical background needed for the computational problems discussed in this report is found in Ref. [1]. The examples given in this report generally enlarge on the discussion of algorithms presented in that book. The user will require access to this book only for purposes of obtaining usage instructions for the subprogram HL2() described in Table 1. This code constructs a Householder orthogonal transformation and applies it to (or multiplies it by) a rectangular matrix. This writeup, Ref. [1], p. 271, will probably be needed only in connection with this report's Sections 5. and 6., Problems with Many Rows-Sequential Accumulation, Example, and Problems with Many Columns.

Least squares problems are intrinsically "more poorly conditioned" than square nonsingular systems. What this means is that the bound for the change in the solution (because of changes in the data) depends on the square of the condition number of A , no matter what method is used to solve $Ax \approx b$. Loosely speaking, this means that when $Ax \neq b$ a small change in the matrix A can cause a change in the solution vector much larger than for the corresponding change in linear n by n systems. A development of these

bounds and the definition of condition number is given in Ref. [1], Chapter 9. Methods for solving $A\tilde{x} \approx \tilde{b}$ based on triangularizing the matrix $[A:\tilde{b}]$ using orthogonal elimination matrices have proved their general numerical superiority over the traditional method of normal equations

$$A^T A \tilde{x} = A^T \tilde{b}$$

The normal equations arise naturally and almost automatically by setting the n partial derivatives to zero,

$$\frac{\partial \rho(\tilde{x})}{\partial x_i} = 0, \quad i = 1, \dots, n$$

This first variation condition is exactly the normal equations $A^T A \tilde{x} = A^T \tilde{b}$.

It is important, therefore, to realize that least squares problems can be solved (and the same solution obtained) by triangularizing the (overdetermined) system $A\tilde{x} \approx \tilde{b}$ using orthogonal elimination matrices. (It is crucial that one not use Gaussian elimination on $A\tilde{x} \approx \tilde{b}$ when the system is inconsistent. This introduces an artificial and essentially arbitrary weighting matrix into the system. The normal equations are $A^T(M^T M)A = A^T(M^T M)\tilde{b}$ where M is the Gaussian elimination matrix products written as a single matrix.)

Many code authors, once discovering the existence of orthogonal matrix elimination methods for least squares problems, have tended to abandon the normal equations method. It is true that the more stable orthogonal matrix elimination methods, such as those implemented in the subprograms SLSEI() and SVA(), and the other subprograms listed in Table 1, should be the user's first choice. Nevertheless there are special situations where these shorter and generally faster algorithms for solving $A\tilde{x} \approx \tilde{b}$,

by solving the normal equations $A^T A \underline{x} = A^T \underline{b}$, are appropriate. Other subprograms that are available to Sandia Laboratories personnel are the SODS/SUDS/CODS package of H.A. Watts, Ref. [3]. These codes allow one to solve the general least squares problem $A \underline{x} \approx \underline{b}$ and include equality constraints $E \underline{x} = \underline{f}$. The source and object code for these subprograms is located in the Sandia Math. Subprogram Library. The user instructions are found in Ref. [4]. It should be noted that this package does not allow the user to compute a covariance matrix (see Section 2 below) with the solution, nor does it allow inequality constraints as part of the problem statement.

The features of SODS and SUDS are essentially contained in the features of the subprogram HFTI() found in Ref. [1], p. 254.

Name of Subprogram	Purpose of Subprogram	Where Documentation is Located (1977)	Where Source and Object Code is Located (1977)
SLSEI()	Solve least squares problems with linear equality and inequality constraints. Some technical restrictions involved, Ref. [8]. Optionally compute a covariance matrix.	The Appendix of this report.	Personal computer files of R.J. Hanson or K. Haskell
SVA()	Solve a least squares problem using singular value analysis.	Ref. [4]	Sandia File FXMATH
NNLS()	Solve a least squares problem with nonnegative constraints on each variable.	Ref. [4]	Sandia File FXMATH
SVDRS()	Compute the singular value decomposition of a least squares system.	Ref. [4]	Sandia File FXMATH
⁽¹⁾ LSEI()	Solve a least squares problem with linear equality and inequality constraints. Optionally compute a covariance matrix.	Ref. [8]	Personal computer files of R.J. Hanson or K. Haskell (1977)
⁽¹⁾ WNNLSE()	Solve a least squares problem with equality constraints and nonnegative constraints on certain variables.	Ref. [8]	Personal computer files of R.J. Hanson or K. Haskell (1977)
H12()	Construct and apply a single Householder transformation to a rectangular matrix.	Ref. [1], p. 271	Sandia File FXMATH
SROTG(), SROT() SROTMG(), SROTM()	Construct and apply a standard or modified Givens transformation	Refs. [9] and [4]	Sandia File FXMATH

⁽¹⁾At the writing of this report, these codes had just been developed after a lengthy period of research, Ref. [8].

Table 1. Sandia Laboratories Least Squares Subprograms,
Documentation and Location (Oct., 1977)

2. Standard Usage, Example

In this section we show how to compute the solution of an m by n linear least squares problem $A\tilde{x} \approx \tilde{b}$ having no constraints. The covariance matrix

$$C \equiv \sigma^2 (A^T A)^{-1}, \quad \sigma^2 = \begin{cases} \rho(\hat{x})/(m-n), & m > n \\ 1, & m = n \end{cases}$$

is also computed in this example.

For the sample problem we chose $m = 5$, $n = 3$ and generated the matrix and right-side vector using the function subprogram GEN(), Ref. [1], p. 277. A "noise" level of 10^{-4} was used, loc. cit. Each data entry was scaled by 10^{-3} so that the values would print clearly using the IMSL vector-matrix output subprograms USWTFV() and USWTFM(), Ref. [5].

Note that the additional working storage for SLSEI(), beyond that for storing $[A:\tilde{b}]$ and $\tilde{x}$, is $2(n+1)$ locations of type REAL and n locations of type INTEGER when $m_E = m_G = 0$.

The solution of this problem is presented by means of the following FORTRAN program. The data matrix, residual vector length, solution vector and covariance matrix have been printed. This is shown following the listing of the program.

A safe and recommended approach to any unconstrained least squares problem is to use the printed output of the subprogram SVA() as a first and one-time computational step. This reveals any poor conditioning present in the system and may show the need for additional constraints as part of the problem statement. See Section 11. for an example using the subprogram SVA().

```

C      SUBROUTINE EXMP1
C      A SAMPLE PROGRAM TO SHOW THE
C      BASIC USAGE OF THE HOUSEHOLDER
C      ALGORITHM FOR SOLVING AN M BY N
C      LEAST SQUARES PROBLEM.
C      A SOLUTION VECTOR AND A COVARIANCE
C      MATRIX ARE COMPUTED
C      WE TAKE M=MA=5 AND N=3 IN
C      THIS EXAMPLE.
C      THE FILES FXMATH AND FXIMSL MUST BE
C      DECLARED AS LIBRARIES IN THE JOB.
C      DIMENSION A(5,4),X(3),WS(8),IP(3),IDOPE(3)
C
C      MDA=5
C      ME=0
C      MA=5
C      MG=0
C      N=3
C
C      SET FLAG TO COMPUTE A COVARIANCE MATRIX..
C      IDOPE(1)=1
C
C      DEFINE THE MATRIX AND RT-SIDE VECTOR..
C      NP1=N+1
C      DUMMY=GEN(-1.E0)
C      AN=1.E-4
C      DO 10 I=1,MA
C          DO 10 J=1,NP1
C              A(I,J)=GEN(AN)*1.E-3
C      10
C      PRINT THE DATA WITH IMSL
C      SUBPROGRAMS..
C      WRITE(6,1001)
C      CALL USWTFM(1,0,0,MA,NP1,MDA,A)
C
C      SOLVE THE LEAST SQUARES PROBLEM..
C      CALL SLSEI(A,MDA,ME,MA,MG,N,X,
C      1      RNORME,RNORML,MODE,WS,IP,IDOPE)
C      WRITE(6,1000) RNORML,MODE
C      CALL USWTFV(1,N,1,1,X)
C      CALL USWTFM(1,0,0,N,N,MDA,A)
C      RETURN
C      1000 FORMAT(24HRESIDUAL VECTOR LENGTH=,1PE12.5,/,
C      1      22H OUTPUT VALUE OF MODE=,14,19H (MODE=0 IS GOOD.),/,
C      2      39H SOLUTION VECTOR AND COVARIANCE MATRIX.)
C      1001 FORMAT(32H1DATA MATRIX AND RT-SIDE VECTOR.)
C      END

```


DATA MATRIX AND RT-SIDE VECTOR.

	1	2	3	4
1	-.0450290000	-.0950157000	.3549807000	.3050304000
2	-.2449883000	-.2949600000	.1550121000	.1050234000
3	-.4449969000	-.4950019000	-.0449938000	-.0949810000
4	.3549863000	.3049974000	-.2450146000	-.2950151000
5	.1549558000	.1050169000	-.4449764000	-.4950052000

RESIDUAL VECTOR LENGTH= 5.65859E-05
 OUTPUT VALUE OF MODE= 0 (MODE=0 IS GOOD.)
 SOLUTION VECTOR AND COVARIANCE MATRIX.

1	-1.0001584950	1.0001531073	1.0001209180
---	---------------	--------------	--------------

	1	2	3
1	.0000001609	-.0000001484	.0000000099
2	-.0000001484	.0000001412	-.0000000072
3	.0000000099	-.0000000072	.0000000053

3. Formulating Curve Fitting Problems

Most real-world computational problems involve nonlinear mathematical models at some step in the process. When this nonlinear model involves solving m nonlinear equations in n variables,

$$\underline{g}(\underline{p}) = \underline{0}$$

the problem originator usually linearizes the equations to

$$\underline{A}\underline{x} \approx \underline{b}$$

Here the solution vector $\underline{x}$ represents a change used to update the parameters, $\underline{p} \leftarrow \underline{p} + \underline{x}$. In particular, this indicates a preference for "small" solutions to $\underline{A}\underline{x} \approx \underline{b}$. Topics of this kind are discussed in Section 11., Ill-Posed Problems, Example, Use of Singular Value Analysis.

There is one class of problems where the model is truly a linear model from the beginning. This class is the (linear) curve fitting problem.

Suppose that a problem originator has a discrete set of data y_i at values t_i , $i = 1, \dots, m$, of the independent variable. The problem of linear least squares curve fitting is as follows:

Given a set of basis functions $\phi_j(t)$, $j = 1, \dots, n$. Find an appropriate set of coefficients x_j , $j = 1, \dots, n$, so that the function

$$f(t) \equiv \sum_{j=1}^n x_j \phi_j(t)$$

approximately fits the data y_i at $t = t_i$, $i = 1, \dots, m$. The approximation

we use is least squares. We choose the unknown components of the vector

$\underline{x} \equiv [x_1, \dots, x_n]^T$ so that

$$\sum_{i=1}^m \left(f(t_i) - y_i \right)^2 = \sum_{i=1}^m \left(\sum_{j=1}^n x_j \phi_j(t_i) - y_i \right)^2$$

is minimized.

This can be written as the m by n system of linear algebraic equations

$$\underline{Ax} \approx \underline{b}$$

The elements of the matrix A and vector $\underline{b}$ satisfy

$$a_{ij} = \phi_j(t_i), \quad j = 1, \dots, n; \quad i = 1, \dots, m$$

$$b_i = y_i, \quad i = 1, \dots, m$$

After this system has been formulated it can be solved using one of the subprograms listed in Table 1.

There are special classes of basis functions $\phi_j(t)$, $j = 1, \dots, n$, where the coefficients x_j can be obtained using more efficient and specialized techniques. Two such examples are the polynomial least squares fitting technique of Forsythe, Ref. [6], p. 43. The subprogram POLFIT() found in the Math. Subprogram Library, Ref. [4], performs this computation. The second example is the Fast Fourier Transform method (FFT), Ref. [7].

We now return to discussing the linear system derived from curve fitting. Many times the data should be weighted so the errors in each b_i are independent of errors in the remaining b_i , and are all about equal. One such weighting method consists of scaling a_{ij} , $j = 1, \dots, n$ and b_i by the reciprocal of the estimated (independent) uncertainty in b_i , $i = 1, \dots, m$. One then solves $\underline{Ax} \approx \underline{b}$.

In certain statistical applications the uncertainty or residual variance function of the curve is needed. This can be computed using the estimation formula, Ref. [11],

$$e^2(t) = \underline{\Phi}^T(t) C \underline{\Phi}(t)$$

Here $\underline{\Phi}(t) = [\phi_1(t), \dots, \phi_n(t)]^T$, while C is the n by n covariance matrix of the coefficients $\underline{x}$. This matrix C is available using the subprogram SLSEI(), noted in Table 1.

The function $e(t) \geq 0$ is often used to obtain the expected uncertainty of the fitted curve, $f(t)$. Necessary assumptions are that the errors in the data are normal, that the average of the data is zero and the uncertainty in each value is about equal. Use of the above mentioned weighting technique and inclusion of a constant as one of the $\phi_j(t)$ accomplishes this. The envelope of values

$$[f(t) - \alpha e(t), f(t) + \alpha e(t)], \quad \alpha \doteq 2.,$$

gives about a 95% confidence interval for all estimated curves of the given form that fit the data.

The dependence of the confidence interval as a function of α and the number of degrees of freedom $m - n$ is found in a t -table, Ref. [11], p. 305.

4. Fitting a Cubic Polynomial to Data

As a specific example of a curve fitting problem we will fit the discrete set of function samples, shown in Table 2, by a cubic polynomial.

t_i	10	20	30	40	50	60	70	80	90
$y(t_i)$	5.45	10.9	15.5	20.55	19.1	22.4	25.6	27.1	28.5

Table 2

This polynomial has the form

$$f(t) = x_1 + x_2 u + x_3 u^2 + x_4 u^3$$

In terms of our original notation,

$$m = 9, \quad n = 4,$$

and

$$\phi_j(t) = u^{j-1}, \quad \text{where } u = (t-\alpha)/\beta$$

with

$$\alpha = (b+a)/2 \quad \text{and} \quad \beta = (b-a)/2.$$

The values of a and b define, respectively, the left and right ends of the interval where the cubic polynomial $f(t)$ is to be evaluated. We take $a = 0$ and $b = 100$.

Notice that the variable u satisfies $-1 \leq u \leq 1$ for $a \leq t \leq b$. Use of this translated and rescaled independent variable results in a better-conditioned linear least squares problem than that obtained by representing $f(t) = y_1 + y_2 t + y_3 t^2 + y_4 t^3$.

The resulting m by n (9 by 4) least squares system $A\mathbf{x} \approx \mathbf{b}$ has its matrix elements and right-hand side entries generated as follows.

Row number i of A is generated by computing $u_i = (t_i - \alpha)/\beta$ and using the recurrence $a_{i1} = 1$, $a_{ij} = u_i a_{i,j-1}$, $j = 2, 3, 4$. The i^{th} component of the vector $\mathbf{b}$ is $y(t_i)$. The j^{th} component of $\mathbf{x}$ is the coefficient of u^{j-1} .

Generating and solving (not shown) this particular least squares problem yields the approximate solution

$$\mathbf{x} \approx [21.074, 10.524, -6.070, 6.455]^T$$

The residual vector length is ≈ 3.004 . A graphical plot of this cubic polynomial and the discrete data it fits is shown in Figure 1. In Section 10, Constrained Curve Fitting by a Cubic Polynomial, Example, we will impose constraints on the cubic polynomial that correspond to additional requirements placed on its shape for $a \leq t \leq b$ and its value at $t = a = 0$. The constrained cubic polynomial and the discrete data it fits is plotted in Figure 2.

For high-order polynomial fitting problems even the basis $\{u^{j-1}\}$ leads to poorly conditioned least squares problems. In this eventuality one should use the Tchebycheff basis, $\{T_j(u)\}$ to represent $f(t) = \sum_{j=0}^n c_j T_j(u)$, Ref. [13], p. 104-111.

5. Problems with Many Rows. Sequential Accumulation, Example

Occasionally there are least squares problems $A\tilde{x} \approx \tilde{b}$ that have significantly many more rows than columns in the matrix A . In this section we will suppose that it is possible to hold an n by n matrix in high-speed memory, but it is not possible or convenient to hold the entire m by n matrix A in high-speed memory.

We will present one method for generating a new least squares problem that requires about the same storage as an n by n matrix. The new problem also has the same set of solution vectors and residual vector length as the original system $A\tilde{x} \approx \tilde{b}$.

To this end we write the matrix A and vector $\tilde{b}$ in partitioned form

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_k \end{bmatrix}, \quad \tilde{b} = \begin{bmatrix} \tilde{b}_1 \\ \tilde{b}_2 \\ \vdots \\ \tilde{b}_k \end{bmatrix}$$

Each A_i is an m_i by n matrix, and each $\underline{b}_i$ is a vector of length m_i . The relation $m = m_1 + \dots + m_k$ holds, of course. The choice of the m_i is at the problem originator's discretion. These integers m_i can all be as small as 1. That choice permits the greatest economy of storage but it also leads to the greatest number of arithmetic operations. Remarks on the dependence of the total number of arithmetic operations on the m_i are given in Ref. [1], p. 211.

The algorithm for generating the least squares problem of reduced size constructs a sequence of upper triangular matrices $[R_i : \underline{d}_i]$, $i = 1, \dots, k$ such that the least squares problem

$$R_i \underline{x} \approx \underline{d}_i$$

has the same set of solutions and the same residual vector length as the least squares problem

$$\begin{bmatrix} A_1 \\ \vdots \\ A_i \end{bmatrix} \underline{x} \approx \begin{bmatrix} \underline{b}_1 \\ \vdots \\ \underline{b}_i \end{bmatrix}$$

Once the matrix $[R : \underline{d}] \equiv [R_k : \underline{d}_k]$ has been computed, the least squares problems

$$A \underline{x} \approx \underline{b} \quad \text{and} \quad R \underline{x} \approx \underline{d}$$

are equivalent in the following ways:

- the sets of solutions to both problems are identical
- the residual vector lengths of both solutions are identical
- any constraints present in the problem statement can be imposed on either system.

The process of computing $[R:d]$ is known as sequential accumulation. The significant idea that allows one to save storage is that for each i the matrix $[R_i:d_i]$ can be computed in the storage space previously occupied by

$$[\hat{A}_i:\hat{b}_i] \equiv \begin{bmatrix} R_{i-1:d_{i-1}} \\ A_i:b_i \end{bmatrix}$$

The matrix $[R_i:d_i]$ is the upper is the upper triangular factor resulting from triangularizing the matrix $[A_i:b_i]$ using orthogonal elimination matrices. The two most well-known such methods use respectively Householder and Givens transformations. See Ref. [1], p. 53 - 57, or p. 58 - 62 respectively. We will use Householder transformations for our illustrations here, but when each $m_i = 1$ the Givens transformation can be used to yield

The modified Givens transformation, Ref. [9], allows one to drop out unwanted rows of data that have already been accumulated. This feature of the modified Givens transformation can often be utilized to avoid storing the entire data matrix when removing these unwanted rows of data. See Table 1 for readily available codes implementing these various transformations.

Whatever the partitioning is that the problem originator chooses, the maximum number of rows of storage required is

$$M \equiv \max_j \left\{ m_j + \min[n+1, \sum_{i=1}^{j-1} m_i] \right\}$$

$$\leq n + 1 + \max_j m_j$$

The sequential accumulation process will take place in a computer FORTRAN array $W(*,*)$ having at least M rows and $n + 1$ columns. We use the notation

$W(i_1:i_2, j_1:j_2)$ to denote the subarray of $W(*,*)$ consisting of rows i_1 through i_2 and columns j_1 through j_2 . We use $W(i,j)$ to denote the (i,j) element of $W(*,*)$ or the start of a row vector or a column vector originating at $W(i,j)$.

Sequential Accumulation Using Householder Triangularization

$L = 0$

For $J = 1, \dots, k$

$IP = L + m_J$

Set $W(L+1:IP, 1:n+1) := [A_J: \underline{b}_J]$

Remark: Transfer the block of data $[A_J: \underline{b}_J]$ to the working array $W(*,*)$ in rows $L + 1, \dots, IP$, columns $1, \dots, n+1$.

For $I = 1, \dots, \min(IP-1, n+1)$

CALL H12(1, I, max(I, L)+1, IP, W(1, I), 1, t, 1, MDW, n-I+1)

CALL SCOPY(IP-I, 0.E0, 0, W(I+1, I), 1)

Remark: Construct the Householder transformation resulting in upper triangular form for rows and columns $1, \dots, I$. Copy zeroes into the column of the matrix where elimination of elements took place.

$L = \min(IP, n+1)$

End

The argument MDW in the CALL to H12() is the row dimension of the FORTRAN working array $W(*,*)$.

The user instructions for H12() and SCOPY() used in this algorithm are respectively noted and found in Table 1 and Refs. [9] or [4].

To illustrate the equivalence of the resulting upper triangular system or reduced problem $R\tilde{x} \approx \tilde{d}$ with the original system $A\tilde{x} \approx \tilde{b}$, we give a sample FORTRAN program computing the solution vector and residual vector length for

both systems. No constraints will be put on the solution, and a covariance matrix will be obtained in both cases.

Since the covariance matrix of the full problem is

$$C_1 \equiv \frac{\sigma^2}{m_A - n} (A^T A)^{-1}$$

while the covariance matrix of the reduced problem is

$$C_2 \equiv \frac{\sigma^2}{m - n} (A^T A)^{-1}$$

we must rescale C_2 to get

$$C_1 = s C_2, \quad s = (m - n) / (m_A - n)$$

The matrix A has dimension 50 by 5, so that $m_A = 50$ and $n = 5$. The function subprogram GEN(), Ref. [1], p. 277, is used to generate the matrix $[A:b]$. A noise level of 10^{-4} , loc. cit., is used.

Each partitioned submatrix A_i of A has dimension 5 by 5. Thus $k = 10$ and $m_i = 5$ $i = 1, \dots, k$. The minimum dimensions of the FORTRAN working array $W(*,*)$ are 11 by 6.

Generally, any of the various solution techniques noted in Table 1, such as SLSEI() or SVA(), can be used to solve the reduced problem $Rx \equiv d$. Thus this process of sequential accumulation can often be viewed as a first step to bring the problem size down to a point where the data all fit into high-speed memory.

```

C      SUBROUTINE EXMPL2
C      SAMPLE PROGRAM TO ILLUSTRATE THE
C      SEQUENTIAL ACCUMULATION OF LEAST
C      SQUARES PROBLEMS THAT HAVE MANY
C      MORE ROWS THAN COLUMNS.
C      THE FILES FXMATH AND FXIMSL MUST BE
C      DECLARED AS LIBRARIES IN THE JOB.

C      DIMENSION A(50,6),B(50),W(11,6)
C      DIMENSION X(5),WS(12),IW(5),IDOPE(3)

C      DEFINE THE ARRAY AND PROBLEM DIMENSIONS..
C      MDA=50
C      MDW=11
C      MA=50
C      N=5

C      GENERATE ON MA BY N LEAST SQUARES PROBLEM..
C      DUMMY=GEN(-1.E0)
C      AN=1.E-4
C      DO 20 I=1,MA
C      B(I)=GEN(AN)
C      DO 10 J=1,N
C      A(I,J)=GEN(AN)
10      20 CONTINUE

C      PERFORM THE SEQUENTIAL ACCUMULATION
C      ALGORITHM. THIS REDUCES AX=B TO AN
C      EQUIVALENT LEAST SQUARES PROBLEM RX=D
C      WITH M ROWS AND N COLS..
C      THE VALUE OF M IS DEFINED BY THE ALGORITHM
C      AS MAX(BLOCK SIZE)+N+1.

C      DEFINE BLOCK SIZE AND NUMBER OF BLOCKS..
C      MJ=5
C      K=10

C      SEQUENTIAL ACCUMULATION ALGORITHM..
C      L=0
C      DO 60 J=1,K
C      IP=MJ+L
C      MOVE BLOCK J OF DATA INTO
C      THE PROCESSING ARRAY..
C      DO 30 JJ=1,N
C      CALL SCOPY(MJ,A((J-1)*MJ+1,JJ),1,W(L+1,JJ),1)
30      CALL SCOPY(MJ,B((J-1)*MJ+1),1,W(L+1,N+1),1)
C      NB=MINO(N+1,IP-1)
C      IF(.NOT. NB.GT.0) GO TO 50
C      TRIANGULARIZE THE MATRIX IN THE
C      PROCESSING ARRAY..
C      DO 40 I=1,NB
C      CALL HIZ(1,I,MAX0(I,L)+1,IP,W(1,I),1,T,W(1,I+1),1,MDW,
1      N-I+1)
C

```

```

C      ZERO THE COLUMNS AS THEY ARE ELIMINATED..
C      CALL SCOPY(IP-1,0.E0,0,W(I+1,I),1)
40      CONTINUE
50      CONTINUE
      L=MIN0(N+1,IP)
60      CONTINUE
C
C      DEFINE NUMBER OF ROWS IN REDUCED PROBLEM..
C      M=L
C
C      SOLVE THIS M BY N SYSTEM USING SLSEI( )
C      WITH NO CONSIRAINTS..
C      ME=0
C      MG=0
C
C      COMPUTE A COVARIANCE MATRIX..
C      IDOPE(1)=1
C      CALL SLSEI(W,MDW,ME,M,MG,N,X,RNORME,RNORML,
1      MODE,WS,IW,IDOPE)
C
C      APPLY SCALE FACTOR TO THE COVARIANCE MATRIX
C      OF THE REDUCED PROBLEM..
C      T=FLOAT(M-N)/FLOAT(MA-N)
C      DO 70 J=1,N
70      CALL SSCAL(N,T,W(1,J),1)
      WRITE(6,1003)
      WRITE(6,1000) RNORML,MODE
      WRITE(6,1001)
C
C      PRINT THE SOLN VECTOR AND COVARIANCE
C      MATRIX WITH IMSL SUBPROGRAMS..
C      CALL USWTFV(1,N,1,1,X)
C      CALL USWTFM(1,0,0,N,N,MDW,W)
C
C      NOW SOLVE THE MA BY N FULL SYSTEM
C      TO SHOW EQUIVALENCE OF THE TWO PROBLEMS..
C
C      CALL SCOPY(MA,B,1,A(1,N+1),1)
C      IDOPE(1)=1
C      CALL SLSEI(A,MDA,ME,MA,MG,N,X,RNORME,RNORML,
1      MODE,WS,IW,IDOPE)
      WRITE(6,1000) RNORML,MODE
      WRITE(6,1002)
      CALL USWTFV(1,N,1,1,X)
      CALL USWTFM(1,0,0,N,N,MDA,A)
      RETURN
1000 FORMAT(24HRESIDUAL VECTOR LENGTH=,1PE12.5, /
1      22H OUTPUT VALUE OF MODE=,14,19H (MODE=0 IS: GOOD.))
1001 FORMAT(34H SOLUTION AND COVARIANCE MATRIX OF,/,
1      17H REDUCED PROBLEM.)
1002 FORMAT(34H SOLUTION AND COVARIANCE MATRIX OF,/,
1      14H FULL PROBLEM.)
1003 FORMAT(14H)
      END

```


RESIDUAL VECTOR LENGTH= 3.05464E-01
 OUTPUT VALUE OF MODE= 0 (MODE=0 IS GOOD.)
 SOLUTION AND COVARIANCE MATRIX OF
 REDUCED PROBLEM.

1	.9999852500	.6267856956	-.6268077697	.3731463991	-.3731739734
	1	2	3	4	5
1	.0000000008	-.0000005512	.0000005517	.0000005545	-.0000005542
2	-.0000005512	.0174638252	-.0174640960	-.0174746208	.0174747104
3	.0000005517	-.0174640960	.0174643678	.0174748957	-.0174749848
4	.0000005545	-.0174746208	.0174748957	.0174854584	-.0174855451
5	-.0000005542	.0174747104	-.0174749848	-.0174855451	.0174856326

RESIDUAL VECTOR LENGTH= 3.05464E-01
 OUTPUT VALUE OF MODE= 0 (MODE=0 IS GOOD.)
 SOLUTION AND COVARIANCE MATRIX OF
 FULL PROBLEM.

1	.9999852500	.6267856956	-.6268077697	.3731463991	-.3731739734
	1	2	3	4	5
1	.0000000008	-.0000005512	.0000005517	.0000005545	-.0000005542
2	-.0000005512	.0174638252	-.0174640960	-.0174746208	.0174747104
3	.0000005517	-.0174640960	.0174643678	.0174748957	-.0174749848
4	.0000005545	-.0174746208	.0174748957	.0174854584	-.0174855451
5	-.0000005542	.0174747104	-.0174749848	-.0174855451	.0174856326

6. Problems with Many Columns

In this section we deal with underdetermined systems of least squares equations $\underline{A}\underline{x} \approx \underline{b}$. The m by n matrix A has many more columns than rows. By this we mean that it is convenient to store $[A:b]$ plus about m^2 additional locations in high-speed memory. It is not convenient to store an n by n array in high-speed storage. (If plenty of storage is available put the data $[A:b]$ in an n by n working array and use the subprogram SVA(), for example).

The matrix A is post-multiplied by an orthogonal matrix Q so that

$$AQ = [L:O]$$

The matrix L above is m by m and lower triangular. This operation can be viewed as an orthogonal change of variables $\underline{x} = Q\underline{y}$. The vector $\underline{y}$ is then partitioned as

$$\underline{y} = \begin{bmatrix} \underline{u} \\ \underline{v} \end{bmatrix} \begin{matrix} \} m \\ \} n-m \end{matrix}$$

The system $\underline{A}\underline{x} \approx \underline{b}$ is then solved as an m by m least squares problem, using the subprograms SVA() or SLSEI(), for a solution $\underline{u} = \hat{\underline{u}}$. Then

$\underline{x} = Q \begin{bmatrix} \underline{u} \\ \underline{v} \end{bmatrix}$ is computed, where the vector $\underline{v}$ is arbitrary. The solution for $\hat{\underline{x}}$ of minimum length is obtained when $\hat{\underline{u}}$ is the solution of $\underline{L}\underline{u} \approx \underline{b}$ of minimum length and $\underline{v} = \underline{0}$.

We give no further algorithmic details for this problem except to point out that the matrix Q can be generated as a product of Householder transformations

$$Q = Q_1 \cdots Q_m$$

Each transformation Q_i zeros the entries in components $i+1, \dots, n$ of row number i of $AQ_1 \cdots Q_{i-1}$. The subprogram H12() documented and described in Ref. [1], p. 271, can be used for this purpose.

We complete this section with the following remarks.

- The solution of $\underline{L}\underline{u} = \underline{b}$, using subprogram SVA(), requires that the m by m upper triangular matrix L be copied to an m by m working storage array. This block of storage must be distinct from the block of storage containing the Householder transformation $Q_i, i=1, \dots, m$. This is due to the fact that the computation of AQ_i is usually organized so that the data stored to represent the Q_i overwrites that row of $AQ_1 \cdots Q_{i-1}$ that was eliminated. The subprogram SVA() requires a full m by m array to solve $\underline{L}\underline{u} = \underline{b}$.
- Another distinct method for obtaining the solution $\underline{x}$, of minimum length for consistent systems $A\underline{x} = \underline{b}$ is by a technique analogous to normal equations. Let $\underline{x} = A^T \underline{z}$. Solve

$$(AA^T)\underline{z} = P\underline{z} = \underline{b}$$

for $\underline{z} = \hat{\underline{z}}$. Then $\hat{\underline{x}} = A^T \hat{\underline{z}}$.

A Cholesky decomposition

$$P = LL^T$$

is computed for the m by m matrix P . The matrix L is m by m and lower triangular. The computational method given in Section 8., Normal Equations, Example can be used to compute the matrix L .

The solution vector is obtained by the following steps:

Solve $\underline{L}\underline{w} = \underline{b}$ for $\underline{w}$

Solve $\underline{L}^T \underline{z} = \underline{w}$ for $\underline{z}$

Compute $\underline{x} = \underline{A}^T \underline{z}$ for $\underline{x} = \hat{\underline{x}}$.

- The solution of underdetermined least squares problems $\underline{A}\underline{x} \approx \underline{b}$ with inequality constraints requires a more general algorithm than that used in the subprogram SLSEI(). Further details on methods and subprograms LSEI() or WNNLSE() for such problems will be found in the Sandia Laboratories technical report of Ref. [8].

7. Banded Least Squares Problems

Frequently the problem of least squares curve fitting to discrete data using piecewise defined basis functions that are zero outside some subinterval leads to banded least squares problems, Ref. [1], p. 212. An example of this type of problem occurs when fitting twice differentiable piecewise cubic polynomials (or cubic splines) to discrete data. Using the basis of B-splines, Ref. [10], one obtains a system in $n + 2$ unknowns $\underline{A}\underline{x} \approx \underline{b}$ for the coefficients, $\underline{x}$.

Here n is the number of breakpoints or knots of the cubic spline

$$\tau_1 < \tau_2 < \dots < \tau_n$$

On each interval $I_i = [\tau_i, \tau_{i+1})$, $1 \leq i < n - 1$, or $I_{n-1} = [\tau_{n-1}, \tau_n]$, there are m_i data points (t_j, y_j) , $\sum_{\ell < i} m_\ell < j \leq \sum_{\ell \leq i} m_\ell$, $i = 1, \dots, n-1$. When the data points are sorted by groups in I_i , $i = 1, \dots, n-1$, the matrix $\underline{A}$ can be written in partitioned form

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_{n-1} \end{bmatrix}$$

where each

$$A_i = \begin{bmatrix} \overbrace{0 : c_i}^{i-1} : \overbrace{0}^4 : \overbrace{0}^{n-i-1} \end{bmatrix}_{m_i}$$

Such least squares problems can be sequentially accumulated using a working storage array of dimension $5(\max_i m_i + 5)$. The subprograms BNDACC() and BND SOL(), documented in Ref. [1], p. 264, can be used for solving such systems.

Generating the values of the B-spline basis functions can be done using de Boor's package of subprograms, Ref. [10]. Usage of this package requires (for cubic splines) adjoining three additional breakpoints to each end, $\tau_j \leq \tau_1$, $j = -2, -1, 0$, and $\tau_l \geq \tau_n$, $l = n+1, n+2, n+3$, when the basis functions are twice-differentiable cubic spline functions.

The least squares problem associated with least squares fitting discrete data by B-splines can also be solved by forming normal equations $A^T A \underline{x} = A^T \underline{b}$ for this problem and using the Cholesky decomposition algorithm. One computes the upper triangular matrix R such that $R^T R = A^T A$ yields the $n + 2$ by $n + 2$ banded upper triangular matrix

$$R = \begin{bmatrix} x & x & x & x & 0 & & & 0 \\ 0 & x & x & x & x & & & \\ & & \ddots & & & & & \\ & & & & & x & x & x \\ 0 & & & & & & x & x \\ & & & & & & & x \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ (n+2)x(n+2) \end{matrix}$$

When R is of rank $n + 2$ the normal equations can be solved by solving the triangular systems

$$R^T \underline{d} = A^T \underline{b} \quad \text{for } \underline{d}$$

and

$$R \underline{x} = \underline{d} \quad \text{for } \underline{x}$$

See Ref. [10] for additional details.

8. Normal Equations Method, Example

Perhaps the most well-known method for solving least squares problems $A \underline{x} \approx \underline{b}$ is the method of normal equations, $A^T A \underline{x} = A^T \underline{b}$. This technique can be implemented based on the observation that since

$$\rho(\underline{x}) = \underline{r}^T \underline{r}, \quad \underline{r} = \underline{b} - A \underline{x},$$

we can write

$$\rho(\underline{x}) = (\underline{x}^T, -1) \tilde{P} \begin{pmatrix} \underline{x} \\ -1 \end{pmatrix}$$

where

$$\tilde{P} = [A : \underline{b}]^T [A : \underline{b}]$$

The $n + 1$ by $n + 1$ matrix $\tilde{P}$ is factored by the Cholesky method, Ref. [1], p. 124, so that

$$\tilde{P} = \tilde{R}^T \tilde{R}$$

where $\tilde{R}$ is the $n + 1$ by $n + 1$ upper triangular matrix

$$\tilde{R} = \begin{bmatrix} R : \underline{d} \\ \underline{0} : e \end{bmatrix}$$

When A has n linearly independent columns, $\rho(\underline{x}) = (\underline{Rx} - \underline{d})^T (\underline{Rx} - \underline{d}) + e^2$ is minimized when $\underline{x}$ is the solution of the upper triangular system $\underline{Rx} = \underline{d}$. The residual vector length is $|e|$ in this case.

The normal equations method outlined here does not have the numerical stability of the subprograms of Table 1. In fact, to solve the same class of problems as the SLSEI() subprogram, one must use twice the arithmetic precision for the normal equations method.

Nevertheless, this method is the fastest of the general methods for least squares problems, at least when there are sufficiently many more rows than columns, ($m > 4n/3$). The size of the computer program necessary to implement this is also small.

This and the algorithm itself is illustrated in the next sample FORTRAN program that we present here. Note that the problem solved is the same one as that used in Section 2., Standard Usage, Example.

The function subprogram SDOT() used in this sample program is described in Refs. [9] or [4]. It computes the dot product of single precision vectors, $w = \underline{x}^T \underline{y}$. The subprogram SCOPY() is also described in Refs. [9] or [4].

```

C      SUBROUTINE EXMPL3
C      A SAMPLE PROGRAM TO SHOW THE
C      METHOD OF NORMAL EQUATIONS
C      FOR SOLVING AN M BY N LEAST
C      SQUARES PROBLEM.
C
C      THIS METHOD IS THE FASTEST GENERAL
C      METHOD FOR THIS PROBLEM, BUT
C      DOES NOT HAVE THE NUMERICAL
C      STABILITY OF THE ORTHOGONAL ELIMINATION
C      METHODS.
C
C      THE FILES FXMATH AND FXIMSL MUST BE
C      DECLARED AS LIBRARIES IN THE JOB.
C
C      DIMENSION A(5,4),X(3),P(4,4)
C      MDA=5
C      MDP=4
C      N=3
C      NP1=N+1
C      M=5
C
C      DEFINE THE MATRIX AND RT-SIDE VECTOR..
C      DUMMY=GEN(-1.E0)
C      AN=1.E=4
C      DO 10 I=1,M
C      DO 10 J=1,NP1
C      A(I,J)=GEN(AN)*1.E-3
C
C      FORM THE NORMAL EQUATIONS FOR
C      THE LEAST SQUARES PROBLEM AX=B.
C      DO 20 I=1,NP1
C      DO 20 J=1,NP1
C      P(I,J)=SDOT(M,A(1,I),1,A(1,J),1)
C
C      COMPUTE THE CHOLESKY DECOMPOSITION
C      OF THE (UPPER TRIANGULAR PART OF THE)
C      PRODUCT (TRANPOSE OF (A;B))*(A;B).
C      DO 30 I=1,NP1
C      T=P(I,I)-SDOT(I-1,P(1,I),1,P(1,I),1)
C      IF(.NOT. T.LE.0.E0) GO TO 30
C      CALL SCOPY(NP1-I+1,0.E0;0,P(I,I),MDP)
C      GO TO 50
C      CONTINUE
C      T=SQRT(T)
C      P(I,I)=T
C      IF(I.EQ.NP1) GO TO 50
C      IP1=I+1
C      T=1./T
C      DO 40 J=IP1,NP1
C      P(I,J)=(P(I,J)-SDOT(I-1,P(1,I),1,P(1,J),1))*T
C      CONTINUE
C      CONTINUE
C
C      SOLVE FOR THE SOLUTION VECTOR
C
C      BY BACK SUBSTITUTION..
C      DO 70 II=1,N
C      I=N+1-II
C      X(I)=(P(I,NP1)-SDOT(II-1,P(I,I+1),MDP,X(I+1),1))/P(I,I)
C      WRITE(6,1000)
C      1000 FORMAT(53H1DATA MATRIX AND CHOLESKY FACTOR OF NORMAL EQUATIONS.)
C      CALL USWTFM(1,0,0,M,NP1,MDA,A)
C
C      FOR OUTPUT PURPOSES ONLY, ZERO
C      THE LOWER TRIANGULAR PART OF P(*,*).
C      DO 80 I=1,N
C      CALL SCOPY(NP1-I,0.E0,0,P(I+1,I),1)
C      CALL USWTFM(1,0,0,NP1,NP1,MDP,P)
C      WRITE(6,1001) P(NP1,NP1)
C      1001 FORMAT(24H0RESIDUAL VECTOR LENGTH=,1PE12.6,/,
C      16H SOLUTION VECTOR)
C      CALL USWTFV(1,N,1,1,X)
C      RETURN
C      END

```

DATA MATRIX AND CHOLESKY FACTOR OF NORMAL EQUATIONS.

	1	2	3	4
1	-.0450290000	-.0950157000	.3549807000	.3050304000
2	-.2449883000	-.2949600000	.1550121000	.1050234000
3	-.4449969000	-.4950019000	-.0449938000	-.0949810000
4	.3549863000	.3049974000	-.2450146000	-.2950151000
5	.1549558000	.1050169000	-.4449764000	-.4950052000

	1	2	3	4
1	.6403871424	.6579729203	-.2964878010	-.2789386315
2	0.0000000000	.1103624269	.1489908697	.2593882095
3	0.0000000000	0.0000000000	.5477201347	.5477863639
4	0.0000000000	0.0000000000	0.0000000000	.0000565859

RESIDUAL VECTOR LENGTH=5.658594E-05
SOLUTION VECTOR

1	-1.0001584950	1.0001531073	1.0001209180
---	---------------	--------------	--------------

9. Equality and Inequality Constraints, Example

There are often situations when the computed solution of an m_A by n least squares problem $A\underline{x} \approx \underline{b}$ is accurate but, as far as the user is concerned, is unacceptable. This is due to the fact that the problem originator is aware of other relations or constraints that an acceptable solution vector $\underline{x}$ satisfies. When these constraints can be defined and expressed as linear equality and inequality constraints, the least squares problem statement becomes

$$A\underline{x} \approx \underline{b}$$

subject to

$$E\underline{x} = \underline{f}$$

$$G\underline{x} \geq \underline{h}$$

In this form the problem can be solved using one of the subprograms SLSEI(), NNLS(), LSEI(), or WNNLSE().

The matrices E and G are respectively m_E by n and m_G by n . The relative sizes of m_E , m_A , m_G , and n are unimportant with the exception noted in Ref. [8] for the subprogram SLSEI().

The subprogram NNLS() considers the special case of nonnegativity constraints on all variables, $\underline{x} \geq 0$, and no equality constraints, or $m_E = 0$. A user with this particular problem is encouraged to use the NNLS subprogram, Ref. [4], because of its efficiency.

The subprogram LSEI() attacks the constrained least squares problem in the full generality stated above, Ref. [8]. This code is a relatively recent development and has not had the amount of usage of SLSEI() or NNLS(). The program is also quite lengthy.

The subprogram WNNLSE() attacks the problem where the inequality

constraints are nonnegativity constraints on variables x_i , $i = \ell+1, \dots, n$. Here ℓ can have any value $0 \leq \ell \leq n$.

For $\ell = 0$, all variables are constrained to be nonnegative. When $\ell = n$, no variables are constrained to be nonnegative. If only members of a selected subset of variables in a user's problem are constrained to be nonnegative, a preliminary interchange of variables will be required so that the constrained subset is written last.

Notice that the capabilities of WNNLSE() seem to be enough to solve the general problem. To see this, define m_G slack variables $\underline{w} \geq \underline{0}$ so that the inequalities becomes equality constraints

$$G\underline{x} - \underline{w} = \underline{h}$$

With this slack variable formulation, the problem statement becomes

$$[A:0] \begin{bmatrix} \underline{x} \\ \underline{w} \end{bmatrix} = \underline{b}$$

subject to

$$\begin{bmatrix} E:0 \\ G:-I \end{bmatrix} \begin{bmatrix} \underline{x} \\ \underline{w} \end{bmatrix} = \begin{bmatrix} \underline{f} \\ \underline{h} \end{bmatrix}$$

and

$$\underline{w} \geq \underline{0}$$

Identifying the vector $[\underline{x}^T, \underline{w}^T]^T$ with a new vector of length $n + m_G$ shows that WNNLSE() is applicable. Variables $n+1, \dots, n+m_G$ are constrained to be nonnegative, so in this case we have $\ell = n$.

This use of WNNLSE() and the slack variable formulation is not recommended generally. For small problems it won't make much difference

whether one uses LSEI() or WNNLSE(). For large problems with $m_E + m_A + m_G > n$, this formulation requires more working storage than usage of LSEI(). See Ref. [8] for more details. Also, WNNLSE() usually requires more computer time by about 25-50% in this context.

When LSEI() or SLSEI() determines that the inequality constraints $Gx \geq h$ are contradictory, the subprogram WNNLSE() can be used to solve the problem, using the slack variable formulation, in a least squares sense. Hence a solution is always obtained and this may prove of value to certain users. See Ref. [8] for more details about this.

We now turn to the solution of a specific constrained least squares problem. In Section 2., Standard Usage, Example the given problem had the unconstrained solution vector

$$\hat{x} = (-1.0002, 1.0002, 1.0001)^T$$

with residual vector length $= 5.60 \times 10^{-5}$.

Suppose that we want to constrain the components of the solution so that

$$x_1 + x_2 + x_3 = 1$$

and with bounds

$$-1. \leq x_i \leq 1., \quad i = 1, 2, 3$$

This additional equality constraint becomes the ($m_E=1$) equation

$$Ex = f$$

where

$$E = (1., 1., 1.) \quad \text{and} \quad f = (1.)$$

The bounds on the variables become the $m_G = 6$ inequalities

$$\underline{G}\underline{x} \geq \underline{h}$$

where

$$G = \begin{bmatrix} 1. & 0. & 0. \\ 0. & 1. & 0. \\ 0. & 0. & 1. \\ -1. & 0. & 0. \\ 0. & -1. & 0. \\ 0. & 0. & -1. \end{bmatrix}, \quad \underline{h} = \begin{bmatrix} -1. \\ -1. \\ -1. \\ -1. \\ -1. \\ -1. \end{bmatrix}$$

Notice that the three bounds $x_i \leq 1.$, $i = 1, 2, 3$, require scaling both sides of the inequalities by $-1.$ to get them expressed as $-x_i \geq -1.$, $i = 1, 2, 3$.

The following FORTRAN program formulates and solves this constrained least squares problem.

The constrained solution is

$$\underline{\tilde{x}} = (-1., 1., 1.)^T$$

with a residual vector length $\doteq 1.007 \times 10^{-4}$.

```

C      SUBROUTINE EXMPL4
C      A SAMPLE PROGRAM TO SHOW
C      THE USAGE OF SLSEI( ) IN SOLVING
C      LEAST SQUARES PROBLEMS WITH
C      EQUALITY OR INEQUALITY CONSTRAINTS.
C
C      THE CONSTRAINTS REQUIRED ON THE
C      SOLUTION ARE THAT THE VARIABLES
C      ADD UP TO 1., AND THE MAGNITUDE
C      OF EACH VARIABLE MUST NOT EXCEED 1.
C
C      THE FILES FXMATH AND FXIMSL MUST BE
C      DECLARED AS LIBRARIES IN THE JOB.
C      DIMENSION W(12,4),X(3),WS(46),IP(9),IDOPE(3)
C
C      MDW=12
C      MC=1
C      MA=5
C      MG=4
C      N=3
C
C      SET FLAG TO NOT COMPUTE A COVARIANCE MATRIX..
C      IDOPE(1)=2
C      NP1=N+1
C
C      DEFINE THE EQUALITY CONSTRAINT..
C      CALL SCOPY(NP1,1.E0,0,W,MDW)
C
C      DEFINE THE LEAST SQUARES EQUATIONS..
C      DUMMY=GEN(-1.E0)
C      AN=1.E-4
C      DO 10 I=1,MA
C          DO 10 J=1,NP1
10         W(I+ME,J)=GEN(AN)*1.E-3.
C
C      DEFINE THE INEQUALITY CONSTRAINTS..
C      DO 20 I=1,MG
20         CALL SCOPY(N,0.E0,0,W(ME+MA+I,1),MDW)
C      CALL SCOPY(N,1.E0,0,W(ME+MA+1,1),MDW+1)
C      CALL SCOPY(N,-1.E0,0,W(ME+MA+N+1,1),MDW+1)
C      CALL SCOPY(MG,-1.E0,0,W(ME+MA+1,N+1),1)
C
C      WRITE(6,1000)
C      CALL USWTFM(1,0,0,ME+MA+MG,N+1,MDW,W)
C
C      SOLVE THE CONSTRAINED LEAST SQUARES PROBLEM..
C      CALL SLSEI(W,MDW,ME,MA,MG,N,X,
1      RNORME,RNORML,MODE,WS,IP,IDOPE)
C
C      WRITE(6,1001) RNORML,MODE
C      CALL USWTFV(1,N,1,1,X)
C      RETURN
1000 FORMAT(58H1EQUALITY CONSTRAINT, (1), LEAST SQUARES EQUATIONS, (2-6
1),,, 36H0AND INEQUALITY CONSTRAINTS, (7-12).)
C
1001 FORMAT(24H0RESIDUAL VECTOR LENGTH=,1PE12.5,/,
1      22H OUTPUT VALUE OF MODE=,14,19H (MODE=0 IS GOOD.),/,
2      17H SOLUTION VECTOR.)
C      END

```


EQUALITY CONSTRAINT, (1), LEAST SQUARES EQUATIONS, (2-6),
AND INEQUALITY CONSTRAINTS, (7-12).

	1	2	3	4
1	1.0000000000	1.0000000000	1.0000000000	1.0000000000
2	-.0450290000	-.0950157000	.3549807000	.3050304000
3	-.2449883000	-.2949600000	.1550121000	.1050234000
4	-.4449969000	-.4950019000	-.0449938000	-.0949810000
5	.3549863000	.3049974000	-.2450146000	-.2950151000
6	.1549558000	.1050169000	-.4449764000	-.4950052000
7	1.0000000000	0.0000000000	0.0000000000	-1.0000000000
8	0.0000000000	1.0000000000	0.0000000000	-1.0000000000
9	0.0000000000	0.0000000000	1.0000000000	-1.0000000000
10	-1.0000000000	0.0000000000	0.0000000000	-1.0000000000
11	0.0000000000	-1.0000000000	0.0000000000	-1.0000000000
12	0.0000000000	0.0000000000	-1.0000000000	-1.0000000000

RESIDUAL VECTOR LENGTH= 1.00734E-04
OUTPUT VALUE OF MODE= 0 (MODE=0 IS GOOD.)
SOLUTION VECTOR.

1	-1.0000000000	1.0000000000	1.0000000000
---	---------------	--------------	--------------

10. Constrained Curve Fitting by a Cubic Polynomial, Example

Often constrained least squares problems arise from fitting discrete data by curves $f(t) = \sum_{j=1}^n x_j \phi_j(t)$ as was noted in Section 3., Formulating Curve Fitting Problems. These constrained problems appear when the problem originator wants to include additional information about the desired curve which is necessary in that context. When these constraints can be formulated as a finite number of equality and inequality constraints on $f(t)$ or its derivatives and integrals, the constrained least squares problem can be stated as

$$A\mathbf{x} = \mathbf{b}$$

subject to

$$E\mathbf{x} = \mathbf{f}$$

and

$$G\mathbf{x} \geq \mathbf{h}$$

The elements of the matrix A and the vector $\mathbf{b}$ are defined in Section 3. The rows of E and G are the set of values of the basis functions, or their derivatives and integrals, at the various points where these constraints are placed on $f(t)$ or its derivatives and integrals. The vectors $\mathbf{f}$ and $\mathbf{h}$ are the corresponding set of constraining values for the equality and inequality constraints on $f(t)$ or its derivatives and integrals.

We now discuss a specific example of a constrained curve fitting problem. In Section 4., Fitting a Cubic to Data, an example was presented where a cubic was fit to the discrete data found in Table 2. In this section we introduce additional problem-related requirements on the fitted curve that makes the unconstrained curve, plotted in Figure 1, unacceptable. Here is the scenario:

- The value of the curve that the data represents is equal to zero at $t = 0$.
- The curve is convex downward for all values of t , $0 \leq t \leq 100$.
- The problem originator has in mind using the fitted curve to extrapolate for values of t near $t = 100$.

The first condition translates into the mathematical statement

$$f(0) = 0$$

In terms of the coefficients of the cubic this becomes

$$\begin{aligned} f(0) &= x_1 + x_2(-1.) + x_3(-1.)^2 + x_4(-1.)^3 \\ &= x_1 - x_2 + x_3 - x_4 = 0 \end{aligned}$$

or the equality constraint equation

$$\underline{Ex} = [1., -1., 1., -1.] \underline{x} = [0] \equiv \underline{f}$$

The second condition regarding convexity of the curve is mathematically equivalent to requiring that

$$f''(t) \equiv \frac{d^2 f(t)}{dt^2} \leq 0, \quad \text{for } 0 \leq t \leq 100$$

Generally for other classes of functions this constraint might require that these inequalities be imposed at a grid of values in t . The grid size will depend on the fitted curve.

In the case of a cubic polynomial the realization of this constraint is much easier. This is due to the fact the $f''(t)$ is a linear function of t . Thus $f''(t) \leq 0$ for $0 \leq t \leq 100$ if and only if $f''(0) \leq 0$ and $f''(100) \leq 0$.

Using the chain rule for ordinary derivatives and the definition of

$u(t)$ as noted in Section 4. we have, since

$$u = (t-\alpha)/\beta, \quad f''(t) = \frac{d^2 f}{du^2} \left(\frac{du}{dt} \right)^2$$

Thus $f''(0) \leq 0$ and $f''(100) \leq 0$ if and only if

$$2x_3 + 6x_3(-1.)$$

$$= [0., 0., 2., -6.] \underline{x} \leq 0$$

$$\text{and } 2x_3 + 6x_3(1.)$$

$$= [0., 0., 2., 6.] \underline{x} \leq 0$$

Cancelling the common factor 2. and changing the sense of the inequalities gives the inequality constraints

$$\underline{G}\underline{x} = \begin{bmatrix} 0. & 0. & -1. & 3. \\ 0. & 0. & -1. & -3. \end{bmatrix} \underline{x} \geq \begin{bmatrix} 0. \\ 0. \end{bmatrix} \equiv \underline{h}$$

The following FORTRAN program fits the given discrete data of Table 2 by the cubic polynomial now constrained to satisfy the stated problem-dependent requirements. The coefficient values obtained are

$$\underline{x} \doteq (21.166, 12.581, -6.439, 2.146)^T$$

The residual vector length is $\doteq 3.284$. The extrapolated value is

$$f(100) = x_1 + x_2 + x_3 + x_4 \doteq 29.456$$

Note that the extrapolated value at $t = 100$ using the unconstrained curve is $\doteq 31.983$. The constrained curve $f(t)$ together with the discrete data it fits is plotted in Figure 2.

Figure 1. Unconstrained Cubic Polynomial Least Squares Fit to Discrete Data...

Figure 2. Constrained Cubic Polynomial Least Squares Fit to Discrete Data

```

SUBROUTINE EXPL5
C
C   A SAMPLE PROGRAM TO SHOW
C   THE USAGE OF SLSEI( ) TO FIT
C   A CUBIC POLYNOMIAL TO THE DISCRETE
C   DATA OF TABLE 1. CONSTRAINTS
C   ARE PUT ON THE DATA.
C   THE INTERVAL OF INTEREST IS FROM
C   T=A=0. TO T=B=100.
C   THE CONSTRAINTS ARE
C   (1) THE CURVE IS ZERO AT T=A.
C   (2) THE CURVE IS CONVEX DOWNWARD
C       FOR ALL VALUES OF T.
C
C   THE FILE FXMATH MUST BE
C   DECLARED AS A LIBRARY
C   IN THE JOB.
C   DIMENSION W(12,5),X(4),WS(26),IP(6),IDOPE(3),T(9),Y(9)
C
C   DATA STATEMENTS FOR THE DISCRETE POINTS..
C   DATA T/10.,20.,30.,40.,50.,60.,70.,80.,90./
C   DATA Y/5.45,10.9,15.5,20.55,19.1,22.4,25.6,27.1,28.5/
C
C   DEFINE THE INTERVAL WIDTH..
C   A=0.
C   B=100.
C
C   COMPUTE PARAMETERS TO CHANGE THE
C   INTERVAL FROM (A,B) TO (-1.,1.).
C   ALPHA=(B+A)/2.
C   BETA=(B-A)/2.
C
C   DEFINE THE MATRIX PROBLEM DIMENSIONS..
C   MDW=12
C   ME=1
C   MA=9
C   MG=2
C   N=4
C   NP1=N+1
C
C   DEFINE THE ARRAY ENTRIES FOR
C   THE EQUALITY CONSTRAINT..
C   W(1,1)=1.
C   W(1,2)=-1.
C   W(1,3)=1.
C   W(1,4)=-1.
C   W(1,5)=0.
C
C   DEFINE THE LEAST SQUARES
C   EQUATIONS..
C   DO 20 I=1,MA
C   U=(T(I)-ALPHA)/BETA
C   W(ME+I,NP1)=Y(I)
C   W(ME+I,1)=1.
C   DO 10 J=2,N
C
C   10      W(ME+I,J)=U*W(ME+I,J-1)
C   20      CONTINUE
C
C   DEFINE THE INEQUALITY
C   CONSTRAINTS..
C   M=ME+MA
C   DO 30 I=1,MG
C   30      CALL SCOPY(NP1,0.,0,W(M+I,1),MDW)
C   W(M+1,3)=-1.
C   W(M+2,3)=-1.
C   W(M+1,4)=3.
C   W(M+2,4)=-3.
C
C   SOLVE THE CONSTRAINED LEAST SQUARES
C   PROBLEM..
C   IDOPE(1)=2.
C   CALL SLSEI(W,MDW,ME,MA,MG,N,X,
C   1      RNORME,RNORML,MODE,WS,IP,IDOPE)
C   WRITE(6,1000)
C   1000  FORMAT(29H1CONSTRAINED CUBIC POLYNOMIAL,/,
C   1      34H FIT FOR DISCRETE DATA OF TABLE 2.)
C   WRITE(6,1001) RNORML,MODE
C   1001  FORMAT(24H0RESIDUAL VECTOR LENGTH=,1PE12.5,/,
C   1      22H OUTPUT VALUE OF MODE=,I4,
C   2      19H (MODE=0 IS GOOD.),/
C   3      33H SOLUTION VECTOR OF COEFFICIENTS.)
C   WRITE(6,1002)(X(I),I=1,N)
C   1002  FORMAT(4H01-4,2X,4F8.3)
C   RETURN
C   END

```

11. Ill-Posed Problems, Example (Use of Singular Value Analysis)

There are occasionally m by n least squares problems $A\tilde{x} \approx \tilde{b}$ whose solution is computed by a numerically stable method, such as shown in Table 1, and this solution is very large compared to the data. This situation is usually undesirable, particularly in the context of linearized approximations to nonlinear equations.

Usually this phenomenon of the large solution is difficult to understand by an examination of the data matrix $[A:\tilde{b}]$ per se.

A powerful method to analyze and compute acceptable approximate solutions to such least squares problems is based on the singular value decomposition of the matrix A , Ref. [1], p. 18, 117, 196. This decomposition is defined by the equation

$$A = USV^T$$

Additionally, the orthogonal matrices U and V are respectively m by m and n by n . The m by n matrix S is a rectangular diagonal matrix with ordered diagonal terms

$$s_{11} \geq s_{22} \geq \dots \geq s_{kk} \geq 0, \quad k = \min(m,n).$$

This decomposition of the matrix A leads to a simplification of the least squares problem

$$S\tilde{y} \approx \tilde{g}$$

where

$$\tilde{x} = V\tilde{y}$$

and

$$\tilde{g} = U^T \tilde{b}$$

The singular value analysis consists primarily of investigating the sequence

of solutions obtained by the closest rank i least squares approximation to A ,

$$\underline{y}_i = (g_1/s_{11}, \dots, g_i/s_{ii}, 0, \dots, 0)^T$$

$$\underline{x}_i = V\underline{y}_i$$

and the corresponding residual vector lengths

$$\|\underline{r}_i\| = \|\underline{b} - A\underline{x}_i\| = \left(\sum_{\ell=i+1}^m g_{\ell}^2 \right)^{\frac{1}{2}}$$

for $i = 1, \dots, r$, where $r = \text{rank of } A = \text{index of last } s_{ii} > 0$.

The subprogram SVA() computes and prints various quantities associated with the singular value analysis of the system $A\underline{x} \approx \underline{b}$.

The basic goal for the problem originator is to attempt to achieve some tradeoff between a minimum value of the residual vector length $\|\underline{r}_i\|$ and the solution vector length $\|\underline{x}_i\|$.

Another often-used method for achieving this above-mentioned tradeoff is to solve the augmented Levenberg-Marquardt regularized system

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

for positive values of the scalar λ . This method is often called "ridge regression," Ref. [1], p. 190. As λ increases the solution vector length and residual vector length respectively decrease and increase.

The subprogram SVA() prints out various quantities associated with the regularized or "ridge regression" system, too.

As an example we consider the application of singular value analysis

to the 8 by 5 matrix and third right hand side vector found in Ref.[12], p. 149. The matrix entry at row 3, column 4 was changed from -11. to 1. The right-hand side component number 1 was changed from 0. to 7. The matrix is printed on the listing.

We will now make a number of remarks about the SVA() generated printed output for this problem.

- The singular values are approximately 35.4, 19.8, 15.1, 6.3 and 10^{-12} .
- The first five components of the transformed right-hand side, g , are approximately -5.1, -3.8, -2.1, -18.3, 3.
- Because $s_5 \doteq (3.6 \times 10^{-14})s_1 \doteq (7 \times \text{machine precision})s_1$, the smallest singular value ($\doteq 10^{-12}$) must be regarded as essentially zero. Thus, we can only consider the approximate solutions $\underline{x}_i$, $i = 1, 2, 3, 4$. Use of $\underline{x}_5$, labeled SOLN 5, would be generally unreasonable because this solution is so large compared to the data.
- Using $\underline{x}_i$, $i = 1, 2, 3, 4$ as an approximate solution, the residual sum of squares starts at approximately 401. and consecutively is reduced to approximately 374.6, 360., 355.7, 21.6.
- Use of $\underline{x}_4$, labelled SOLN 4, gives

$$\underline{x}_4 \doteq (-0.59, 0.57, 0.54, 2.73, 0.27)^T$$

as an approximate solution. This results in $\|\underline{x}_4\| \doteq 2.9$ and $\|\underline{r}_4\| \doteq 4.6$.

- An approximate value of λ in the regularized or "ridge regression" problem that results in about the same solution and residual vector lengths is any value

$$0.125 \times 10^{-4} \leq \lambda \leq 0.026.$$

- The fifth column of the V matrix gives a vector that generates the null-space of the matrix A. The vector

$$\underline{v}_5 = (-0.1424, -0.2440, -0.3457, 0., 0.8948)^T$$

gives a linear combination of the columns of A that are essentially the zero vector. In particular the length of $\underline{Av}_5$ is approximately 10^{-12} , the smallest singular value.

There are often occasions when a user may want to suppress the printout of SVA(). One may compute the singular value decomposition of the system $\underline{Ax} = \underline{b}$ using the subprogram SVDRS(). Then an approximate solution vector $\underline{x}_i$ is obtained by choosing i so that $\|\underline{r}_i\|$ and $\|\underline{x}_i\|$ are both acceptably small.

Those ill-posed problems also having linear equality and inequality constraints can usually be handled by using "ridge regression" techniques and subprogram SLSEI(), or the more specialized subprograms LSEI() and WNNLSE() of Ref. [8].

```

C      SUBROUTINE EXMPL6
C      SAMPLE LEAST SQUARES PROBLEM
C      ANALYZED AND SOLVED USING
C      THE SINGULAR VALUE ANALYSIS
C      SUBPROGRAM SVA( ).
C      THIS EXAMPLE WAS TAKEN FROM
C      THE BOOK LINEAR ALGEBRA BY
C      WILKINSON AND REINSCH, P. 149.
C      THE MATRIX ENTRY AT ROW 3, COL 4 WAS
C      CHANGED FROM -11. TO 1.
C      THE RT-SIDE ENTRY IN ROW 1 WAS
C      CHANGED FROM 0. TO 7.
C
C      THE FILES FXMATH AND FXIMSL MUST BE
C      DECLARED AS LIBRARIES IN THE JOB.
C
C      DIMENSION A(8,6),S(15)
C      DATA NAMES /IH /
C      DATA A /22.,14.,-1.,-3.,9.,9.,2.,4.,
1      10.,7.,13.,-2.,8.,1.,-6.,5.,
2      2.,10.,-1.,13.,1.,-7.,6.,0.,
3      3.,0.,1.,-2.,-2.,5.,5.,-2.,
4      7.,8.,3.,4.,4.,-1.,1.,2.,
5      7.,1.,11.,4.,-6.,3.,12.,-5./
C
C      MDA=0
C      M=8
C      N=5
C      MDATA=8
C      ISCALE=1
C
C      PRINT OUT THE DATA MATRIX
C      AND RT-SIDE VECTOR..
C      WRITE(6,1000)
1000  FORMAT(27H1MATRIX AND RT-SIDE VECTOR.)
C      CALL USWTFM(1,0,0,M,N+1,MDA,A)
C
C      PRINT OUT THE ANALYSIS OF
C      THE SYSTEM USING SVA( ).
C      CALL SVA(A,MDA,M,N,MDATA,A(1,N+1),S,NAMES,ISCALE,D)
C      RETURN
C      END

```

MATRIX AND RT-SIDE VECTOR.

	1	2	3	4	5	6
1	22.0000000000	10.0000000000	2.0000000000	3.0000000000	7.0000000000	7.0000000000
2	14.0000000000	7.0000000000	10.0000000000	0.0000000000	8.0000000000	1.0000000000
3	-1.0000000000	13.0000000000	-1.0000000000	1.0000000000	3.0000000000	11.0000000000
4	-3.0000000000	-2.0000000000	13.0000000000	-2.0000000000	4.0000000000	4.0000000000
5	9.0000000000	8.0000000000	1.0000000000	-2.0000000000	4.0000000000	-6.0000000000
6	9.0000000000	1.0000000000	-7.0000000000	5.0000000000	-1.0000000000	3.0000000000
7	2.0000000000	-6.0000000000	6.0000000000	5.0000000000	1.0000000000	12.0000000000
8	4.0000000000	5.0000000000	0.0000000000	-2.0000000000	2.0000000000	-5.0000000000

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

M = 8, N = 5, MDATA = 8

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	COL 4	COL 5		
1	-8015.	1310.	-5033.	2587.	-1424.		
2	-4771.	2438.	7760.	-2263.	-2440.		
3	-1570.	-9160.	475.	-1204.	-3457.		
4	-636.	1151.	-3462.	-9289.	0.		
5	-3183.	-2666.	1495.	-671.	8948.		
INDEX	SING. VALUE	P COEF	RECIP. S.V.	G COEF	G**2	C.S.S.	N.S.R.C.S.S.
0						4.0100E+02	7.0799E+00
1	35.3946	-1.4522E-01	2.8253E-02	-5.1399E+00	2.6418E+01	3.7458E+02	7.3152E+00
2	19.7725	-1.9282E-01	5.0575E-02	-3.8126E+00	1.4536E+01	3.6005E+02	7.7465E+00
3	15.1141	-1.3836E-01	6.663E-02	-2.0912E+00	4.3732E+00	3.5567E+02	8.4341E+00
4	6.3115	-2.8960E+00	1.5444E-01	-1.8278E+01	3.3408E+02	2.1591E+01	2.3233E+00
5	.9816E-13	3.0726E+13	1.087E+13	3.0162E+00	9.0972E+00	1.2493E+01	2.0407E+00
INDEX	YNORM	RNORM	LOG10(YNORM)	LOG10(RNORM)			
0	D.	.20025E+02	-1000.00000	1.30157			
1	.14522E+00	.19354E+02	-.83799	1.28677			
2	.24139E+00	.18975E+02	-.61729	1.27818			
3	.27823E+00	.18859E+02	-.55560	1.27553			
4	.29093E+01	.46466E+02	.46379	.66713			
5	.30726E+14	.35346E+01	13.48751	.54834			

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)
.35395E+03	.18270E-02	.20004E+02	2.54894	-2.73826	1.30112
.52612E+02	.66451E-01	.19331E+02	1.72109	-1.17750	1.28625
.78205E+01	.11677E+01	.12026E+02	.89324	.06734	1.08012
.11625E+01	.28146E+01	.46852E+01	.06539	.44942	.67072
.17280E+00	.29071E+01	.46466E+01	-.76246	.46347	.66713
.25685E-01	.29092E+01	.46466E+01	-1.59031	.46378	.66713
.38180E-02	.29093E+01	.46466E+01	-2.41816	.46379	.66713
.56753E-03	.29093E+01	.46466E+01	-3.24601	.46379	.66713
.84360E-04	.29093E+01	.46466E+01	-4.07386	.46379	.66713
.12540E-04	.29093E+01	.46466E+01	-4.90171	.46379	.66713
.18640E-05	.29105E+01	.46466E+01	-5.72956	.46397	.66713
.27707E-06	.48310E+01	.46466E+01	-6.55741	.68403	.66713
.41185E-07	.17457E+03	.46466E+01	-7.38526	2.24198	.66713
.61219E-08	.78998E+04	.46466E+01	-8.21311	3.89762	.66713
.91000E-09	.35753E+06	.46466E+01	-9.04096	5.55331	.66713
.13527E-09	.16181E+08	.46466E+01	-9.86881	7.20901	.66713
.20107E-10	.73232E+09	.46465E+01	-10.69666	8.86470	.66713
.29888E-11	.33109E+11	.46445E+01	-11.52451	10.51995	.66694
.44426E-12	.14303E+13	.45567E+01	-12.35236	12.15541	.65865
.66037E-13	.21153E+14	.36574E+01	-13.18021	13.32537	.56317
.98161E-14	.30422E+14	.35347E+01	-14.00806	13.48319	.54836

SEQUENCE OF CANDIDATE SOLUTIONS, X

	SOLN 1	SOLN 2	SOLN 3	SOLN 4	SOLN 5
1	.11638931E+00	.91133923E-01	.16076627E+00	-.58838658E+00	-.43740366E+13
2	.69282995E-01	.22269671E-01	-.85096974E-01	.57035515E+00	-.74983484E+13
3	.22793224E-01	.19942123E+00	.19285394E+00	.54159688E+00	-.10622660E+14
4	.92383742E-02	-.12951348E-01	.34951001E-01	.27250000E+01	.31751525E+01
5	.46218317E-01	.97621238E-01	.76879936E-01	.27119779E+00	.27493944E+14

References

- [1] Lawson, C. L., Hanson, R. J., Solving Least Squares Problems. Prentice-Hall, Inc. (1974).
- [2] Wisniewski, J. A., and Hanson, R. J., "A Linear Algebraic Equation Polyalgorithm for the CDC 6600 and 7600. A User's Guide. Sandia Laboratories Technical Report SAND77-0871, (1977).
- [3] Watts, H. A., "Solving Linear Least Squares Problems Using SODS/SUDS/CODS." Sandia Laboratories Technical Report SAND77-0683, (1977).
- [4] Jones, R. E., Haskell, K., "Brief Instructions for Using MATHLIB. Version (7.2)." Sandia Laboratories Technical Report SAND77-1441, (1978).
- [5] IMSL Library 1 Ref. Manual. Vols. 1,2, Ed. 6, (July, 1977).
- [6] Shampine, L. F., and Allen, R. C., Jr., Numerical Computing: An Introduction, W. B. Saunders Co. (1973)
- [7] Bloomfield, P., Fourier Analysis of Time Series: An Introduction, John Wiley and Sons, (1976).
- [8] Haskell, K., Hanson, R. J., "An Algorithm for Linear Least Squares with Equality and Nonnegativity Constraints," Sandia Laboratories Technical Report SAND77-0552, (1978).

- [9] Lawson, C. L., Hanson, R. J., Krogh, F. T., Kincaid, D. R., "Basic Linear Algebra Subprograms for FORTRAN Usage. An Extended Report " Sandia Laboratories Technical Report SAND77-0829, (1977).
- [10] de Boor, C., "Package for Calculating with B-Splines," SIAM J. Numer. Anal., Vol. 14, No. 3, p. 441, June, (1977).
- [11] Draper, N. R., Smith, H., Applied Regression Analysis, John Wiley and Sons, (1966).
- [12] Wilkinson, J. H., Reinsch, C., Linear Algebra. Springer-Verlag, (1971)
- [13] Dahlquist, G., Björk, Å., Numerical Methods, Prentice-Hall, Inc., (1974).

Appendix User Instructions for Subprogram SLSEI()

SUBROUTINE SLSEI(W,MDW,ME,MA,MG,N,X,RNORME,RNORML,MODE,WS,IP,IDOPE)
 DIMENSION W(MDW,N+1),X(N),WS(2*ME+(MG+2)*(N+3)-4),IP(MG+N),IDOPE(3)

Written by R. J. Hanson. For further mathematical and algorithmic details see the Sandia Laboratories Tech. Rept. of Ref. [8]. The methods used in the tech. rept. refer to the book Solving Least Squares Problems, by C. L. Lawson and R. J. Hanson, Prentice-Hall, Inc. (1974).

Abstract

This subprogram solves a linearly constrained least squares problem with both equality and inequality constraints, and, if the user requests, obtains a covariance matrix of the solution parameters.

Suppose there are given matrices E, A and G of respective dimensions m_E by n , m_A by n and m_G by n , and vectors $\underline{f}$, $\underline{b}$ and $\underline{h}$ of respective lengths m_E , m_A and m_G . This subroutine solves the linearly constrained least squares problem

$$\underline{E}\underline{x} = \underline{f} \quad (E \text{ } m_E \text{ by } n) \quad (\text{Equations to be exactly satisfied})$$

$$\underline{A}\underline{x} \approx \underline{b} \quad (A \text{ } m_A \text{ by } n) \quad (\text{Equations to be approximately satisfied, least squares sense})$$

$$\underline{G}\underline{x} \geq \underline{b} \quad (G \text{ } m_G \text{ by } n) \quad (\text{Inequality constraints})$$

The inequalities $Gx \geq h$ mean that every component of the product Gx must be $\geq$ the corresponding component of $\underline{h}$.

In case the equality constraints cannot be satisfied, a generalized inverse solution residual vector length is obtained for $\underline{f} - E\underline{x}$. This is the minimal length possible for $\underline{f} - E\underline{x}$.

For the important special nonnegative constraint case where $m_E = 0$, $m_G = n$, $h = 0$, G is an n by n identity matrix, and no covariance matrix is desired, the user is advised to call the subprogram NNLS found in the Sandia Math. Subprogram Library.

Any values $m_E \geq 0$, $m_A \geq 0$, or $m_G \geq 0$ are permitted. The rank of the matrix E is estimated during the computation. We call this value KRANKE. It is an output parameter in IDOPE(1) defined below. Using a generalized inverse solution of $E\underline{x} = \underline{f}$, a reduced least squares problem with inequality constraints is obtained. The reduced least squares problem of dimension m_A by n -KRANKE must be of full column rank, however. The tolerances used in these tests for determining the rank of E and the rank of the reduced least squares problem are given in Ref. [8].

The user must dimension all arrays appearing in the call list: $W(MDW, n+1), X(n), WS(2m_E + (m_G+2)(n+3)-4), IP(m_G+n), IDOPE(3)$. This allows for a solution of a range of problems in the given working space. The dimension of $WS(*)$ given here is a necessary overestimate. Once that a particular problem has been run, the output parameter IDOPE(3) gives the actual dimension required.

The parameters for SLSEI are

Input:

$W(*,*)$, MDW, ME, MA, MG, N The array $W(*,*)$ is doubly subscripted with first dimensioning parameter equal to MDW. For this discussion let us call $M = m_E + m_A + m_G$. Then MDW must satisfy $MDW \geq M$. The condition $MDW < M$ is considered an error.

The array $W(*,*)$ contains the matrices and vectors

$\begin{pmatrix} E & \underline{f} \\ A & \underline{b} \\ G & \underline{h} \end{pmatrix}$

In rows and columns 1,...,m and 1,...,n+1 respectively.

The integers ME, MA, and MG are the respective matrix row dimensions of E , A and G , m_E , m_A , and m_G . Each matrix has $N = n$ columns.

IDOPE(1) An integer flag to indicate whether the covariance matrix of the solution parameters is desired. The only equality constraints used for this computation are those explicitly stated as such in the input to the subprogram. Use of this subprogram feature typically increases running time by 30-40 percent.

IDOPE(1) =

- 1 Compute the covariance matrix
- 2 Do not compute the covariance matrix

Output:

MODE INTEGER flag that indicates the subprogram status after completion. If MODE.GE.2, no solution has been computed.

MODE =

- 0 Both equality and inequality constraints are compatible and have been satisfied.
- 1 Equality constraints are contradictory. A generalized inverse solution of $Ex = f$ was used to minimize the residual vector length $\|f - Ex\|$. In this sense, the solution is still meaningful.
- 2 Inequality constraints are contradictory.
- 3 Both equality and inequality constraints are contradictory
- 4 The reduced least squares problem is rank deficient.

W(*,*) The array W(*,*) contains the n by n symmetric covariance matrix of the solution parameters, provided this was requested on input with IDOPE(1) = 1, and the integer output flag is returned with MODE = 0 or 1.

X(*), RNORME, RNORML The array X(*) contains the solution parameters provided the integer output flag MODE = 0 or 1. When MODE = 0 or 1, RNORME and RNORML respectively contain the residual vector euclidean lengths of $f - Ex$ and $b - Ax$. When MODE = 1 the equality constraint equations $Ex = f$ are contradictory, so RNORME.NE.0. The residual vector $f - Ex$ has minimal euclidean length. For MODE.GE.2, none of these parameters are defined.

IDOPE(*) An integer array with entries that provide rank and working array length information after completion.

IDOPE(1) = Rank of equality constraint matrix. Define this quantity as KRANKE.

IDOPE(2) = Rank of reduced least squares problem.

IDOPE(3) = The amount of storage in the working array WS(*) that was actually used by the subprogram. The formula given for the length of WS(*) is $2m_E + (m_G + 2)(n + 3) - 4$. This is an overestimate. To find the exact space required by the subprogram, available only after a given problem has been run at least once, substitute n-KRANKE for n and KRANKE for m_E in this formula.

Distribution:

1116 J. D. Plimpton
1223 R. B. Easterling
1223 I. J. Hall
1223 R. L. Iman
1223 R. R. Prairie
2610 R. E. Detry
2613 E. A. Aronson
2613 L. A. Bertram
2613 B. Marder
2613 K. Haskell (5)
2613 R. E. Jones
2613 M. R. Scott
2613 W. Vandevender
2613 H. A. Watts
5000 A. Narath
Attn: 5200 E. H. Beckner
5300 O. E. Jones
5400 A. W. Snyder
5700 J. H. Scott
5800 R. S. Claassen
5100 J. K. Galt
Attn: 5110 F. L. Vook
5130 G. A. Samara
5150 J. E. Schirber
5160 W. Herrmann
5120 G. J. Simmons
5121 H. T. Davis
5121 C. A. Morgan
5121 G. P. Steck
5121 R. J. Thompson
5122 D. E. Amos (2)
5122 R. J. Hanson (10)
5122 K. L. Hiebert
5122 B. L. Hulme
5122 L. F. Shampine
5122 J. A. Wisniewski

5162 D. Hicks
5231 F. Biggs
5231 P. J. McDaniel
5411 R. K. Cole
5411 S. L. Daniel
5411 A. W. Frazier
5732 L. C. Bartel
8325 R. E. Huddleston
8325 T. H. Jefferson
8325 T. A. Manteuffel
8325 D. Crawford
8327 J. F. Lathrop
8326 A. A. Aas
3141 C. A. Pepmueller (Actg.) (5)
3151 W. L. Garner (3)
For DOE/TIC (Unlim. Release)
3172-3 R. Campbell (25)
For DOE/TIC