A Parameter Estimation Subroutine Package
A Parameter Estimation Subroutine Package

G. J. Bierman
M. W. Nead

April 15, 1978

National Aeronautics and Space Administration
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California
The work described in this report was performed by the Systems Division of the Jet Propulsion Laboratory.
ACKNOWLEDGEMENT

The construction of this Estimation Subroutine Package (ESP) was motivated by an involvement with a particular problem; construction of fast, efficient and simple least squares data processing algorithms to be used for determining ephemeris corrections. Discussion with T. C. Duxbury led to the proposal of a subroutine strategy which would have great flexibility. The general utility of such a subroutine package was made evident by H. M. Koble and N. A. Mottinger who had a different but related problem that involved combining estimates from different missions. Thanks and credit are also due to our colleagues for experimenting with this package of subroutines and letting us benefit from their experience.
ABSTRACT

Linear least squares estimation and regression analyses continue to play a major role in orbit determination and related areas. In this report we document a library of FORTRAN subroutines that have been developed to facilitate analyses of a variety of estimation problems. Our purpose is to present an easy to use, multi-purpose set of algorithms that are reasonably efficient and which use a minimal amount of computer storage. Subroutine inputs, outputs, usage and listings are given, along with examples of how these routines can be used. The following outline indicates the scope of this report: Section I, introduction with reference to background material; Section II, examples and applications; Section III, a subroutine directory summary; Section IV, the subroutine directory user description with input, output and usage explained; and Section V, subroutine FORTRAN listings. The routines are compact and efficient and are far superior to the normal equation and Kalman filter data processing algorithms that are often used for least squares analyses.
CONTENTS

I. Introduction .............................................. 1

II. Applications and Examples ............................... 4

III. Subroutine Directory Summary .......................... 23

IV. Subroutine Directory User Description .................. 38

V. References ............................................... 84

VI. FORTRAN Subroutine Listings ............................ 85
I. Introduction

Techniques related to least squares parameter estimation play a prominent role in orbit determination and related analyses. Numerical and algorithmic aspects of least squares computation are documented in the excellent reference work by Lawson and Hanson, Ref. [1]. Their algorithms, available from the JPL subroutine library, Ref. [2], are very reliable and general. Experience has, however, shown that in reasonably well posed problems one can streamline the least squares algorithm codes and reduce both storage and computer times. In this report, we document a collection of subroutines most of which we have written that can be used to solve a variety of parameter estimation problems.

The algorithms for the most part involve triangular and/or symmetric matrices and to reduce storage requirements these are stored in vector form, e.g., an upper triangular matrix $U$ is written as

\[
\begin{bmatrix}
U_{11} & U_{12} & U_{13} & U_{14} \\
U_{22} & U_{23} & U_{24} & \text{etc.} \\
U_{33} & U_{34} \\
0 & & & U_{44}
\end{bmatrix}
= \begin{bmatrix}
U(1) & U(2) & U(4) & U(7) \\
U(3) & U(5) & U(8) & \text{etc.} \\
0 & & & U(10)
\end{bmatrix}
\]

Thus, the element from row $i$ and column $j$ of $U$, $i < j$, is stored in vector component $j(j-1)/2 + i$. We hasten to point out that the engineer, with few exceptions, need have no direct contact with the vector subscripting. By this we mean that the vector subscript related operations are internal to the subroutines, vector arrays transmitted from one
subroutine to another are compatible, and vector arrays displayed using the print subroutine TRIMAT appear in a triangular matrix format.

Aside: The most notable exception is that matrix problems are generally formulated using doubly subscripted arrays. Transforming a double subscripted symmetric or upper triangular matrix $A(\cdot,\cdot)$ to a vector stored form, $U(\cdot)$ is quite simply accomplished in FORTRAN via

$$
\begin{align*}
\text{IJ} &= 0 \\
\text{DO I J = 1,N} \\
\text{DO I J = 1,I} \\
\text{IJ} &= \text{IJ+1} \\
\text{U(IJ)} &= A(I,J)
\end{align*}
$$

Similarly, transforming an initial vector $D(\cdot)$ of diagonal positions of a vector stored form, $U(\cdot)$, is accomplished using

$$
\begin{align*}
\text{JJ} &= 0 \\
\text{DO J J = 1,N} \\
\text{JJ} &= \text{JJ+1} \\
\text{U(JJ)} &= D(J)
\end{align*}
$$

or

$$
\begin{align*}
\text{JJ} &= N^2(N+1)/2 \\
\text{DO J J = N,1,-1} \\
\text{U(JJ)} &= D(J) \\
\text{JJ} &= \text{JJ-J}
\end{align*}
$$

The conversion on the right has the modest advantage that $D$ and $U$ can share common storage (i.e., $U$ can overwrite $D$). These conversions are too brief to be efficiently used as subroutines. It seems that when such conversions are needed one can readily include them as in-line code.

End of Aside

This package of subroutines is designed, in the main, for the analysis of parameter estimation problems. One can, however, use it to solve problems that involve process noise and to map (time propagate) covariance or information matrix factors. In the case of mapping the storage savings associated with the use of vector stored triangular matrices is, to some extent, lost.
Mathematical background regarding Householder orthogonal transformations for least squares analyses and U-D matrix factorization for covariance matrix analyses are discussed in references [1] and [3].

Our plan is to illustrate, in Section II, with examples, how one can use the basic algorithms and matrix manipulation to solve a variety of important problems. The subroutines which comprise our estimation subroutine package are described in Section III, and detailed input/output descriptions are presented in Section IV.

Section V contains FORTRAN listings of the subroutines. There are several reasons for including such listings. Making these listings available to the engineer analyst allows him to assess algorithm complexity for himself; and to appreciate the simplicity of the routines he would otherwise to use as a black box. The routines use only FORTRAN IV and are therefore reasonably portable (except possibly for routines which involve alphanumeric inputs). When estimation problems arise to which our package does not directly apply (or which can be made to apply by an awkward concatenation of the routines) one may be able to modify the codes and widen still further the class of problems that can be efficiently solved.
II. APPLICATIONS AND EXAMPLES

Our purpose in this section is to illustrate, with a number of examples, some of the problems that can be solved using this ESP. The examples, in addition, serve to catalogue certain estimation techniques that are quite useful.

To begin, let us catalogue the subroutines that comprise the ESP:

<table>
<thead>
<tr>
<th>No.</th>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A2A1</td>
<td>Matrix A to matrix A1</td>
</tr>
<tr>
<td>2</td>
<td>COMBO</td>
<td>Combine R and A namelists</td>
</tr>
<tr>
<td>3</td>
<td>COVRHO</td>
<td>Covariance to correlation matrix, RHO</td>
</tr>
<tr>
<td>4</td>
<td>COV2RI</td>
<td>Covariance to R inverse</td>
</tr>
<tr>
<td>5</td>
<td>COV2UD</td>
<td>Covariance to U-D covariance factors</td>
</tr>
<tr>
<td>6</td>
<td>C2C</td>
<td>Permute the rows and columns of matrix C</td>
</tr>
<tr>
<td>7</td>
<td>INF2R</td>
<td>Information matrix to (triangular) R factor</td>
</tr>
<tr>
<td>8</td>
<td>HHPOST</td>
<td>Householder triangularization by post multiplication</td>
</tr>
<tr>
<td>9</td>
<td>PERMUT</td>
<td>Permute the columns of matrix A</td>
</tr>
<tr>
<td>10</td>
<td>PHIU</td>
<td>Multiplies a rectangular PHI matrix by the vector stored U matrix that has implicitly defined unit diagonal entries.</td>
</tr>
<tr>
<td>11</td>
<td>RA</td>
<td>R(upper triangular, vector stored)*A (rectangular)</td>
</tr>
<tr>
<td>12</td>
<td>RANK1</td>
<td>Updated U-D factors of a rank-1 modified matrix</td>
</tr>
<tr>
<td>13</td>
<td>RCOLRD</td>
<td>(SRIF)R colored noise time-update</td>
</tr>
<tr>
<td>14</td>
<td>RINCON</td>
<td>R inverse along with a condition number bounding estimate</td>
</tr>
<tr>
<td>15</td>
<td>RI2COV</td>
<td>R inverse to covariance</td>
</tr>
<tr>
<td>16</td>
<td>R2A</td>
<td>Triangular R to (rectangular stored) matrix A</td>
</tr>
<tr>
<td>17</td>
<td>R2RA</td>
<td>Transfer to triangular block of (vector stored) R to a triangular (vector stored) RA</td>
</tr>
<tr>
<td>18</td>
<td>RUDR</td>
<td>(SRIF)R to U-D covariance factors, or vice-versa</td>
</tr>
<tr>
<td>19</td>
<td>SFU</td>
<td>Sparse F matrix * vector stored U matrix with implicitly defined unit diagonal entries</td>
</tr>
<tr>
<td>20</td>
<td>TDHHT</td>
<td>Two dimensional Householder matrix triangularization</td>
</tr>
<tr>
<td>21</td>
<td>THH</td>
<td>Triangular vector stored Householder data processing algorithm</td>
</tr>
<tr>
<td>22</td>
<td>TTHH</td>
<td>Orthogonal triangularization of two triangular matrices</td>
</tr>
<tr>
<td>23</td>
<td>TWOMAT</td>
<td>Two dimensional labeled display of a vector stored triangular matrix</td>
</tr>
</tbody>
</table>
These routines are described in succeedingly more detail in sections IV, IV, and V. The examples to follow are chosen to demonstrate how these various subroutines can be used to solve orbit determination and other parameter estimation problems. It is important to keep in mind that these examples are not by any means all inclusive, and that the package of subroutines has a wide scope of applicability.

II.1 Simple Least Squares

Given data in the form of an overdetermined system of linear equations one may want a) the least squares solution; b) the estimate error covariance, assuming that the data has normalized errors; and c) the sum of squares of the residuals. The solution to this problem, using the ESP can be symbolically depicted as

\[ [A; z] \xrightarrow{THH} [R; z], e \]

Remarks: The array \([A; z]\) corresponds to the equations \(Ax = z-v, \ v \in N(0, I)\);
the array \([R; z]\) corresponds to the triangular data equation \(Rx = z-v, \ v \in N(0, I)\) and \(e = ||z-\hat{A}x||\)

\[ [R; z] \xrightarrow{UTINV} [R^{-1}; x] \]

Remark: \(x = R^{-1} \hat{z} \)

24) TZERO (T zero) Zero a horizontal segment of a vector stored triangular matrix
25) UDCOL (U-D colored) U-D covariance factor color update
26) UDMEAS (U-D measurement) U-D covariance factor measurement update
27) UD2COV (U-D to cov) U-D factors to covariance
28) UD2SIG (U-D to sig) U-D factors to sigmas
29) UTINV (U inverse) Upper triangular matrix inverse
30) UTIROW Upper triangular inverse, inverting only the upper rows
31) WGS (W G-S) U-D covariance factorization using a weighted Gram-Schmidt reduction
One may be concerned with the integrity of the computed inverse and the estimate. If one uses subroutine RINV instead of UTINV then in addition one obtains an estimate (lower and upper bounds) for the condition number $R$. If this condition number estimate is large the computed inverse and estimate are to be regarded with suspicion. By large, we mean considerable with respect to the machine accuracy (viz. on an 18 decimal digit machine numbers larger than $10^{12}$). Note that the condition number estimate is obtained with negligible additional computation and storage.

\[ [R^{-1}] \rightarrow [C] \]

**Remarks:** $C = R^{-1}R^{-T}$ = estimate error covariance. Some computation can be avoided in R12COV if only some (or all) of the standard deviations are wanted.

11.2 Least Squares With A Priori

If a priori information is given, it can be included as additional equations (in $\hat{V}_2$ A array) or used to initialize the $R$ array in subroutine TTH (see the subroutine argument description given in section IV). One is sometimes interested in seeing how the estimate and/or the formal statistics change corresponding to the use of different a priori conditions. In this case one should compute $[R;\hat{z}]$ as in case 11.1, and then include the a priori $[R;\hat{z}; o]$ using either subroutine TTH or subroutine TTHH when the a priori is diagonal or triangular, e.g.,

\[ [R;\hat{z}] \rightarrow TTHH \rightarrow [R;\hat{z}] \]

# The new result overwrites the old.
It is often good practice to process the data and form \([\bar{R}:\bar{z}]\) before including the a priori effects. When this is done one can analyze the effect of different a priori, \([R_0:z_0]\) without reprocessing the data.

If a priori is given in the form of an information matrix, \(A\), (as for example would be the case if the problem is being initialized with data processed using normal equation data accumulation *) then one can obtain \(R_0\) from \(A\) using \(\text{INF2R}\);

\[
\Lambda \xrightarrow{\text{INF2R}} R_0
\]

If there were a normal equation estimate term, \(z = A^Tb\), then \(z_0 = R_0^{-T}z\).

11.3 Batch Sequential Data Processing

Prime reasons for batch sequential data processing are that many problems are too large to fit in core, are too expensive in terms of core cost, and for certain problems it is desirable to be able to incorporate new data as it becomes available. Subroutines THH and UDMEAS are specially designed for this kind of problem. Both of these subroutines overwrite the a priori with the result which then acts as a priori for the next batch of data. If the data is stored on a file or tape as \(A_1, z_1, A_2, z_2, \ldots\) then the sequential process can be represented as follows:

**SRIF Processing**

a) Initialize \([R:z]\) with a priori values or zero

b) Read the next \([A:z]\) from the file

---

* i.e., solving \(Ax = b-v\) with normal equations, \(A^T\bar{x}_0 = A^Tb; \quad \Lambda = A^TA\) is the information matrix.

** The acronym SRIF represents Square Root Information Filter. The SRIF is discussed at length in the book by Bierman, ref. [3].
c) \[
\begin{pmatrix} 
[R; z] \\
[A; z] 
\end{pmatrix}
\xrightarrow{\text{THH}} 
\begin{pmatrix} 
[R; z]^* \\
[A; z]^* 
\end{pmatrix}
\]

d) If there is more data go back to b)

e) Compute estimates and/or covariances using UTINV and R12COV

(as in example II.1)

**U-D** Processing

a) Initialize \([U-D; x]\) with a priori U-D covariance factors and the initial estimate

b') Read the next \([A; z]\) scalar measurement from the file

c') \[
\begin{pmatrix} 
[U-D; x] \\
[A; z] 
\end{pmatrix}
\xrightarrow{\text{UDMEAS}} 
\begin{pmatrix} 
[U-D; x]^* \\
[A; z]^* 
\end{pmatrix}
\]

d') If there is more data go back to b')

e') Compute standard deviations or covariances using UD2SIG or UD2COV.

Note that subroutine THH is best (most efficiently) used with data batches of substantial size (say 5 or more) and that UDMEAS processes measurement vectors one component at a time. If the dimension of the state is small the cost of using either method is generally negligible. The UDMEAS subroutine is best used in problems where estimates are wanted with great frequency or where one wishes to monitor the effects of each update. In a given application one might choose to process data in batches for a while and during critical periods it may be

*The new result overwrites the old.

**U-D processing is a numerically stable algorithmic formulation of the Kalman filter measurement update algorithm, cf reference [3]. The estimate error covariance is used in its \(UDU^*\) factored form, where \(U\) is unit upper triangular and \(D\) is diagonal.
desirable to monitor the updating process on a point by point basis.

In cases such as this, one may use RUDR to convert a SRIF array to U-D
form or vice-versa.

Remarks: Another case where an R to U-D conversion can be useful occurs
in large order problems (with say 100 or more parameters) where after
data has been SRIF processed one wants to examine estimate and/or
covariance sensitivity to the a priori variances of only a few of the
variables. Here it may be more convenient to update using the UDMEAS
subroutine.

II.4 Reduced State Estimates and/or Covariances From a SRIF Array

Suppose, for example, that data has been processed and that we have a
triangular SRIF array [R;z] corresponding to the 14 parameter names, \( a_r, a_x, a_y, x, y, z, v_x, v_y, v_z, GM, CU41, LO41, CU43, LO43 \)
(constant spacecraft accelerations, position and velocity, target body gravitational constant,
and spin axis and longitude station location errors).

Let us ask first what would the computed error covariance be of
a model containing only the first 10 variables, i.e., by ignoring the
effect of the station location errors. One would apply UTINV and RI2COV
just as in example II.1, except here we would use \( N \) (the dimension of
the filter) = 10, instead of \( N=14 \).

Next, suppose that we want the solution and associated covariance
of the model without the 3 acceleration errors. One ESP solution is to
use
Remark: One could also have used subroutine COMBO, with the desired namelist as simply $a_r, a_x, a_y$. This would achieve the same $A$ matrix form.

Remark: $R$ here can replace the original $R$ and $z$.

Remarks: Here, use only $N=11$, i.e., 11 variables and the RHS. $x_{est}$ is the 11 state estimate based on a model that does not contain acceleration errors $a_r, a_x, or a_y$. Note how triangularizing the rearranged $R$ matrix produces the desired lower dimensional SRIF array; and this is the same result one would obtain if the original data had been fit using the 11 state model.

As the last subcase of this example suppose that one is only interested in the SRIF array corresponding to the position and velocity variables. The difference between this example and the one above is that here we want to include the effects due to the other variables.

* $z$ is often given the label RHS (right hand side)
One might want this sub-array to combine with a position-velocity SRIF array obtained from, say, optical data. One method to use would be,

\[
\begin{align*}
&[R : z] \xrightarrow{R2A} [R_A : z_A] \\
\text{INPUT NAMES:} & \quad \text{OUTPUT NAMES:} \\
&i_r, a_x, a_y, x, y, z, v_x, v_y, v_z, GM \quad x, y, z, v_x, v_y, v_z, GM \\
&\text{CU41, LO41, CU43, LO43, RHS} \quad \text{CU41, LO41, CU43, LO43, RHS}
\end{align*}
\]

**Remark:** The lower triangle starting with \( x \) is copied into \( R_A \).

\[
\begin{align*}
&[R_A : z_A] \xrightarrow{R2A} [A : z_A] \quad \text{(Reordering)} \\
\text{NAMES:} & \quad \text{GM, CU41, LO41, CU43, LO43,} \\
&x, y, z, v_x, v_y, v_z, RHS
\end{align*}
\]

\[
\begin{align*}
&[A : z_A] \xrightarrow{THH} [R_A : z_A] \quad \text{(Triangularizing)} \\
&[R_A : z_A] \xrightarrow{R2RA} [R : z_x] \quad \text{(Shifting array)} \\
\text{NAMES:} & \quad x, y, z, v_x, v_y, v_z, RHS
\end{align*}
\]

**Remark:** The lower right triangle starting with \( x \) is copied into \( R_x \).

We note that one could have elected to use COMBO in place of the first R2RA usage \( R2A \); this would have involved slightly more storage, but a lesser number of inputs. The sequence of operations is in this case,

\[
\begin{align*}
&[R : z] \xrightarrow{\text{COMBO}} [A : z] \\
\text{ORIGINAL NAMES} & \quad \text{DESIRED NAMES:} \quad x, y, z, v_x, v_y, v_z, RHS
\end{align*}
\]

**Remark:** By using COMBO the columns of \( [R : z] \) are ordered corresponding to the names \( a_r, a_x, a_y, \text{GM, CU41, LO41, CU43, and LO43,} \) followed by the desired names list.
Remark: The \([R;z]\) array that is output from this procedure is equivalent but different from the \([R;z]\) array that we began with.

Remark: As before, the lower right triangle starting with \(x\) is copied into \(R_x\).

To delete the last \(k\) parameters from a SRIF array, it is not necessary to use subroutines R2A and THH. The first \(N - k = N\) columns of the array already correspond to a square root information matrix of the reduced system. If estimates are involved one can simply move the \(z\) column left using:

\[
R \left( \frac{N(N+1)}{2} + 1 \right) = R(N(N+1)/2 + 1), \ i = 1, \ldots, k.
\]

Remark: We mention in passing that if one is only interested in estimates and/or covariances corresponding to the last \(k\) parameters then one can use R2RA to transform the lower right triangle of the SRIF array to an upper left triangle after which UTINV and R12COV can be applied.

II.5 Sensitivity, Perturbation, Computed Covariance and Consider Covariance Matrix Computation

Suppose that one is given a SRIF array

\[
\begin{bmatrix}
N_x & N_y & 1 \\
N_x & N_y & 1 \\
R_x & R_{xy} & z_x \\
0 & R_y & z_y
\end{bmatrix}
\]

\((\text{II.5a})\)
in which the \( y \) variables are to be considered. (One can, of course, using subroutines R2A and THH reorder and retriangularize an arbitrarily arranged SRIF array so that a given set of variables fall at the end.) For various reasons one may choose to ignore the \( y \) variables in the equation

\[
R_x \mathbf{x} + R_{xy} \mathbf{y} = \mathbf{z} - \nu_x, \quad \nu_x \sim N(0,1)
\]

and take as the estimate \( \mathbf{x} = R_x^{-1} \mathbf{z} \). It then follows that

\[
\mathbf{x} - \mathbf{x}_c = -R^{-1}_x R_{xy} \mathbf{y} - R^{-1}_x \nu_x,
\]

and from this one obtains

\[
\frac{\partial (\mathbf{x} - \mathbf{x}_c)}{\partial \mathbf{y}} = -R^{-1}_x R_{xy}
\]

(sensitivity of the estimate error to the unmodeled \( y \) parameters)

\[
\text{Pert} = \text{Sen} \ast \text{Diag}(\sigma_y(1), \ldots, \sigma_y(N_y))
\]

where \( \sigma_y(1), \ldots, \sigma_y(N_y) \) are a priori \( y \) parameter uncertainties.

(The perturbations are a measure of how much the estimate error could be expected to change due to the unmodeled \( y \) parameters.)

\[
P_{\text{con}} = R^{-1}_x R^{-T}_x + \text{Sen} \text{P}_y \text{Sen}^T
\]

\[
= P_c + (\text{Pert})(\text{Pert})^T \text{if } P_y \text{ is diagonal}^+^+
\]

where \( P_c \) is the estimate error covariance of the reduced model.

An easy way to compute \( P_c \), Pert and \( P_{\text{con}} \) is as follows: Use subroutine R2RA to place the \( y \) variable a priori \( [P_y^b(0): \mathbf{y}_0]^{++} \) into the lower right

\[
\text{Pert} = \text{Sen} P_y^b
\]

\( ^+^+ \)The a priori estimate \( \mathbf{y}_0 \) of consider parameters is generally zero.
corner of (II.5a), replacing \( R_y \) and \( z_y \), i.e.,

\[
\begin{bmatrix}
[R : z]
\end{bmatrix}
\xrightarrow{\text{R2RA}}
\begin{bmatrix}
R_x & R_{xy} & z_x \\
0 & P_{y}^{\dagger}(0) & \hat{y}_o
\end{bmatrix}
\]

Now apply subroutine UTIROW to this system (with a -1 set in the lower right corner)

\[
\begin{bmatrix}
R_x & R_{xy} & z_x \\
0 & P_{y}^{\dagger}(0) & \hat{y}_o \\
0 & 0 & -1
\end{bmatrix}
\xrightarrow{\text{UTIROW}}
\begin{bmatrix}
-1 & \text{Pert}^{**} & x_c \\
0 & P_{y}^{\dagger}(0) & \hat{y}_o \\
0 & 0 & -1
\end{bmatrix}
\]

Note that the lower portion of the matrix is left unaltered, i.e., the purpose of UTIROW is to invert a triangular matrix, given that the lower rows have already been inverted. From this array one can, using subroutine RI2COV, get both \( P_c \) and \( P_{\text{con}} \)

\[
[R_x^{-1}] \xrightarrow{\text{RI2COV}} [P_c] \quad \text{computed covariance}
\]

\[
[R_x^{-1} : \text{Pert}] \xrightarrow{\text{RI2COV}} [P_{\text{con}}] \quad \text{consider covariance}
\]

Suppose now that one is dealing with a U-D factored Kalman filter formulation. In this case estimate error sensitivities can be sequentially

\* To have estimates from the triangular inversion routines one sets a -1 in the last column (below the right hand side).

\** Strictly speaking this is not what we call the perturbation unless \( P_y(0) \) is diagonal.

14
calculated as each scalar measurement \( z = a_x^T x + a_y^T y + v \) is processed.

\[
Sen_j = Sen_{j-1} - K_j (a_x^T e_{j-1} + a_y^T)
\]

where \( Sen_{j-1} \) is the sensitivity prior to processing the \( j \)-th measurement, and \( K_j \) is the Kalman gain vector.

In this formulation one computes \( P_{\text{con}} \) in a manner analogous to that described in section 11.7;

Let \( \tilde{U}_l = U_j, \tilde{D}_1 = D_j \) (filter \( U-D \) factors)

\[
[s_1, \ldots, s_n] = Sen_j \text{ (estimate error sensitivities)}
\]

then recursively compute

\[
\begin{align*}
\tilde{U}_k \cdot \tilde{D}_k, s_k^2, s_k & \quad \text{RANK1} \quad \tilde{U}_{k+1} \cdot \tilde{D}_{k+1} \quad k = 1, \ldots, n_y
\end{align*}
\]

For the final \( \tilde{U}-\tilde{D} \) we have

\[
U_{j+1}^{\text{con}} = \tilde{U}_{n_y+1}, D_{j+1}^{\text{con}} = \tilde{D}_{n_y+1}
\]

If \( P_y(0) = U_D U^T \), instead of \( P_y(0) = \text{Diag} (d_1^2, \ldots, d_n^2) \), then in the \( U-D \) recursion one should replace the \( Sen_j \) columns by those of \( Sen_* U \) and \( d_j^2 \) should be replaced by the corresponding diagonal elements of \( D_y \).

II.6 Combining Various Data Sets

In this example we collect several related problems involving data sets with different parameter lists.

Suppose that the parameter namelist of the current data does not correspond to that of the a priori SRIF array. If the new data involves a permutation or a subset of the SRIF namelist, then an application of

\[
^+ K = g/\alpha \text{ where } g \text{ and } \alpha \text{ are quantities computed in subroutine UDMEAS.}
\]
subroutine PERMUT will create the desired data rearrangement. If the data involves parameters not present in the SRIF namelist then one could use subroutine R2A to modify the SRIF array to include the new names and then if necessary use PERMUT on the data, to rearrange it compatibly.

Suppose now that two data sets are to be combined and that each contains parameters peculiar to it (and of course there are common parameters). For example let data set 1 contain names ABC and data set 2 contain names DEB. One could handle such a problem by noting that the list ABCDE contains both name lists. Thus one could use subroutine PERMUT on each data set comparing it to the master list ABCDE, and then the results could be combined using subroutine THH. An alternative automated method for handling this problem is to use subroutine COMBO with data set 1 (assuming it is in triangular form) and namelist 2. The result would be data set 1 in double subscripted form and arranged to the namelist ACDEB (names A and C are peculiar to data set 1 and are put first). Having determined the namelist one could apply subroutine PERMUT to data set 2 and give it a compatible namelist ordering.

The process of increasing the namelist size to accommodate new variables can lead to problems with excessively long namelists, i.e., with high dimension. If it is known that a certain set of variables will not occur in future data sets then these variables can be eliminated and the problem dimension reduced. To eliminate a vector y from a SRIF array, first use subroutine R2A to put the y names first in the namelist; then use subroutine THH to retriangularize and finally use subroutine R2RA to put the y independent subarray in position for further use; viz.
The rows \([R:R_y:R_x:z_y]\) can be used to recover a \(y\) estimate (and its covariance) when an estimate for \(x\) (and its covariance) are determined. (See example 11.4).

Still another application related to the combining of data sets involves the combining of SRIF triangular data arrays. One might encounter such problems when combining data from different space missions (that involve common parameters) or one might choose to process data of each type or tracking station separately and then combine the resulting SRIF arrays. Triangular arrays can be combined using subroutine THH, assuming that subroutines R2A, THH and R2RA have been used previously to formulate a common parameter set for each of the sub problems.

II.7 Batch Sequential White Noise

It is not uncommon to have a problem where each data set contains a set of parameters that apply only to that set and not to any other, viz. the data is of the form

\[ A_j x + B_j y_j = z_j - v_j \quad j = 1, \ldots, N \]

where there is generally a priori information on the vector \(v_j\) variables. Rather than form a concatenated state vector composed of \(x, y_1, \ldots, y_N\) which might create a problem involving exhorbitant amounts of storage and computation we solve the problem as follows. Apply subroutine THH to \([B_1:A_1:z_1]\), with the corresponding \(R\) initialized with the \(y_1\) a priori. The resulting SRIF array is of the form

\[ [R:R_y:R_x:z_x] \]
Copy the top \( N \) rows if one will later want an estimate or covariance of the \( y \) parameters. Apply subroutine TZERO to zero the top \( N \) rows and using subroutine R2RA set in the \( y \) a priori*. This SRIF array is now ready to be combined with the second set of data \([B_2; A_2; z_2]\) and the procedure repeated.

A somewhat analogous situation is represented by the class of problems that involve noisy model variations, i.e., the state at step \( j+1 \) satisfies

\[
x_{j+1} = x_j + G_j w_j
\]

where matrix \( G_j \) is defined so that \( w_j \) is independent of \( x_j \) and \( w_j \sim N(0, Q_j) \).

Models of this type are used to reflect that the problem at hand is not truly one of parameter estimation, and that some (or all) of the components vary in a random (or at least unknown) manner that is statistically bounded. To solve this problem in a SRIF formulation suppose that a priori for \( x_j \) and \( w_j \) are written in data equation form (cf ref. [3]),

\[
R_j x_j = z_j - v_j \quad \mid v_j \sim N(0, I)
\]

\[
Q_j^{1/2} w_j = 0 - v_j^{(w)} \quad \mid v_j^{(w)} \sim N(0, I_{n_w})
\]

where \( Q_j^{1/2} \) is a Cholesky factor of \( Q_j \) that is obtainable from COV2R1. Combining these two equations with the one for \( x_{j+1} \) gives

*In this example it is assumed that all of the \( y \) variables have the same dimension. This assumption, though not essential, simplifies our description of the procedure.
where $Q_{w} = w_{j}$. This is the equation to be triangularized with subroutine THH, i.e.,

\[
\begin{bmatrix}
I_{n_{w}} & 0 \\
-R_{j}G_{j}Q_{j}^{T} & R_{j}
\end{bmatrix}
\begin{bmatrix}
\hat{w}_{j} \\
x_{j+1}
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
z_{j}
\end{bmatrix}
- 
\begin{bmatrix}
\nu_{j}^{(w)} \\
\nu_{j}
\end{bmatrix}
\]

When the problem is arranged so that $Q_{j}$ is diagonal one can reduce storage and computation. Incidentally, the form of this algorithm allows one to use singular $Q_{j}$ matrices.

When the a priori for $x_{j}$ and $Q_{j}$ are given in U-D factored form, one can obtain the U-D factors for $x_{j+1}$ as follows:

Let $Q_{j} = u^{(q)} D^{(q)} (u^{(q)})^{T}$ (use COV2UD if necessary)

Set $\bar{G} = G_{j} u^{(q)} = \{g_{1}, \ldots, g_{n_{w}}\}$, $D^{(q)} = \text{Diag}(d_{1}, \ldots, d_{n_{w}})$

Apply subroutine RANK1 $n_{w}$ times, with $\bar{U}_{0} = \bar{U}_{j}$, $\bar{D}_{0} = D_{j}$

\[
(\bar{U}-\bar{D})_{k}; d_{k}, \bar{z}_{k} \xrightarrow{\text{RANK1}} (\bar{U}-\bar{D})_{k+1}
\]

i.e.

\[
(\bar{U}_{k} \bar{D}_{k} \bar{U}_{k}^{T} + d_{k} \bar{s}_{k} \bar{s}_{k}^{T} = \bar{U}_{k+1} \bar{D}_{k+1} \bar{U}_{k+1}^{T})
\]

Then $U_{j+1} = \bar{U}_{n_{w}}$, $D_{j+1} = \bar{D}_{n_{w}}$.
Certain filtering problems involve dynamic models of the form

\[ x_{j+1} = \phi_j x_j + G_j w_j \]

Given an estimate for \( x_j \), \( \hat{x}_j \), the predicted estimate for \( x_{j+1} \), denoted \( \tilde{x}_{j+1} \) is simply

\[ \tilde{x}_{j+1} = \phi_j \hat{x}_j \]

The U-D factors of the estimate error corresponding to the estimate \( \tilde{x}_{j+1} \) can be obtained using the weighted Gram-Schmidt triangularization subroutine

\[ [\phi_j, U_j : C]; \text{Diag}(D_j, D_j(q)) \xrightarrow{WGS} (\tilde{U}_{j+1} - \tilde{D}_{j+1}) \]

Subroutine PHIU can be used to construct \( \phi_j \hat{U}_j \). Note that this matrix multiplication updates the estimate too, because it is placed as an attached column to the \( U \) matrix.

When the \( w \) and associated \( x \) terms correspond to a colored noise model, \( \phi_{j+1} \approx \phi_j \hat{x}_j + w_j \), then it is easier and more efficient to use the colored noise update subroutine UDCOL. Note that here too the estimate is updated by the subroutine calculation because the estimate is an attached column of \( U \).

II.8 Miscellaneous Uses of the Various ESP Subroutines

In certain parameter analyses we may want to reprocess a set of data suppressing different subsets of variables. In this case the original data should be left unaltered and subroutine A2A1 used to copy \( A \) into \( A_1 \), which then can be modified as dictated by the analysis.

Covariance analysis subroutines are initialized using a covariance matrix from a different problem (or a different phase of the same problem). In such cases it may be necessary to permute, delete or insert rows and columns into the covariance matrix; and that can be achieved using subroutine C2C.

If a priori for the problem at hand is given as a covariance matrix then one can compute the corresponding SRIF or U-D initialization using

\[ x(j+1|j) = \phi_j x(j|j) \]

In statistical notation that is commonly used, one writes
subroutine COV2RI or COV2UD. Of course, if the covariance is diagonal the appropriate R and U-D factors can be obtained more simply. To convert a priori given in the form of an information matrix to a corresponding SRIF matrix one applies subroutine INF2R. To display covariance results corresponding to the SRIF or U-D filter one can use subroutines UTINV, RI2COV and UD2COV. The vector stored covariance results can be displayed in a triangular format using subroutine TWOMAT.

Parameter estimation does not, in the main, involve matrix multiplication. Certain applications, such as coordinate transformations and time propagation are important enough to warrant inclusion in the ESP. For that reason we have included RA (to post multiply a square root information matrix) and PHIU to premultiply a U-covariance factor). Certain time propagation problems involve sparse transition matrices, and for this we have included the subroutine SFU. Other special matrix products involving triangular matrices were not included because we have had no need for other products to date), and they are generally not lengthy or complicated to construct. We illustrate this point by showing how to compute \( z = Rx \) where \( R \) is a triangular vector stored matrix and \( x \) is an N vector,

\[
\begin{align*}
II & = 0 \\
& \text{DO 2 I=1,N} \\
SUM & = 0. \quad \text{@SUM is Double Precision} \\
II & = II + I \quad \text{@II=(I,I)} \\
IK & = II \\
& \text{DO 1 K=1,N} \\
SUM & = SUM + R(IK) \times (K) \quad \text{@IK=(I,K)} \\
& 1 \quad IK = IK + K \\
& 2 \quad z(I) = SUM \quad \text{@z can overwrite x if desired}
\end{align*}
\]
Note that the II and IK incremental recursions are used to circumvent the $N(N+1)/2$ calculations of $IK = K(K-1)/2 + I$. 
III. SUBROUTINE DIRECTORY SUMMARY

1. A2A1 - (A to Al)

Reorders the columns of a rectangular matrix A, storing the result in matrix Al. Columns can be deleted and new columns added. Zero columns are inserted which correspond to new column name entries. Matrices A and Al cannot share common storage.

**Example III.1**

\[
\begin{align*}
\text{A} & \quad \text{B} \quad \text{C} \\
1 & 5 & 9 \\
2 & 6 & 10 \\
3 & 7 & 11 \\
4 & 8 & 12 \\
\end{align*}
\]

\[
\begin{align*}
\text{Al} & \quad \text{B} \quad \text{F} \quad \text{G} \quad \text{C} \quad \text{H} \\
5 & 0 & 0 & 9 & 0 \\
6 & 0 & 0 & 10 & 0 \\
7 & 0 & 0 & 11 & 0 \\
8 & 0 & 0 & 12 & 0 \\
\end{align*}
\]

The new namelist (BFGCH) contains F, G and H as new columns and deletes the column corresponding to name \(a\).

**Example III.2**

Suppose one is given an observation data file with regression coefficients corresponding to a state vector with components say, \(x, y, z, v_x', v_y', v_z\) and station location errors. Suppose further, that the vector being estimated has components \(a_x^+, a_y^+, a_z^+\), \(x, y, z, v_x, v_y, v_z\), GM and station location errors. A2A1 can be used to reorder the matrix of regression coefficients to correspond to the state being estimated. Zero coefficients are set in place for the accelerations and GM which are not present in the original file.

\[
\text{in track and cross track accelerations}
\]
2. **COMBO** - (combine R and A namelists)

The upper triangular vector stored matrix R has its columns permuted and is copied into matrix A. The names associated with R are to be combined with a second namelist.

The namelist for A is arranged so that R names not contained in the second list appear first (left most). These are then followed by the second list. Names in the second list that do not appear in the R namelist have columns of zeros associated with them.

**Example III.3**

```
<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>B</th>
<th>E</th>
<th>α</th>
<th>F</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>
```

R-Vector stored A-Double subscripted

A principal application of this subroutine is to the problem of combining equation sets containing different variables, and automating the process of combining name lists.

3. **COVRHO** - (covariance to correlation matrix)

A vector stored correlation matrix, RHO, is computed from an input positive semi-definite vector stored matrix, P. Correlations corresponding to zero diagonal covariance elements are zero. To economize on storage the output RHO matrix can overwrite the input P matrix. The principal function of correlation matrices is to expose strong pairwise component correlations (|RHO(IJ)|.LE.1, and near unity in magnitude). It is sometimes erroneously assumed that numerical ill-conditioning
of the covariance matrix can be determined by inspecting the correlation matrix entries. While it is true that RHO is better conditioned than is the covariance matrix, it is not true that inspection of RHO is sufficient to detect numerical ill-conditioning. For example, it is not at all obvious that the following correlation matrix has a negative eigenvalue.

\[
\begin{bmatrix}
1. & 0.49999 & 0.49999 \\
0.49999 & 1. & -0.49999 \\
0.49999 & -0.49999 & 1.
\end{bmatrix}
\]

4. **COV2RI** - (Covariance to R inverse)

An input positive semi-definite vector stored matrix P is replaced by its upper triangular vector stored Cholesky factor S, \( P = S S^T \). The name RI is used because when the input covariance is positive definite, \( S = R^{-1} \).

5. **COV2UD** - (Covariance to U-D factors)

An input positive semi-definite vector stored matrix P is replaced by its upper triangular vector stored U-D factors. \( P = U D U^T \).

6. **C2C** - (C to C)

Reorders the rows and columns of a square (double subscripted) matrix C and stores the result back in C. Rows and columns of zeros are added when new column entries are added.

**Example III.4**

\[
\begin{bmatrix}
A & B & \Gamma \\
A & 1 & 4 & 7 \\
B & 2 & 5 & 8 \\
\Gamma & 3 & 6 & 9
\end{bmatrix} \rightarrow \begin{bmatrix} \Gamma & P & B & Q \\
9 & 0 & 6 & 0 \\
0 & 0 & 0 & 0 \\
8 & 0 & 5 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

Names P and Q have been added and name A deleted. An important application of this subroutine is to the rearranging of covariance matrices.
7. **INF2R** - (Information matrix to R)

Replaces a vector stored positive semi-definite information matrix \( \Lambda \) by its lower triangular Cholesky factor \( R^T; \Lambda = RR^T \). The upper triangular matrix \( R \) is in the form utilized by the SRIF algorithms. The algorithm is designed to handle singular matrices because it is a common practice to omit a priori information on parameters that are either poorly known or which will be well determined by the data.

8. **HHPOST** - (Householder orthogonal triangularization by post multiplication)

The input, double subscripted, rectangular matrix \( W(M.N) (M.LE.N) \) is triangularized, and overwritten, by post-multiplying it by an implicitly defined orthogonal transformation, i.e.

\[
[ W ]T \rightarrow [ 0 \backslash S ]
\]

This subroutine is used, in the main, to retriangularize a mapped covariance square root and to include in the effects of process noise (i.e. \( W = [\Phi * P^{1/2} : BQ^{1/2}] \)) and to compute consider covariance matrix square roots (i.e. \( W = [P^{1/2}_{\text{computed}}: \text{Sen} * P^{1/2}_y] \)).

9. **PERMUT**

Reorders the columns of matrix \( A \), storing the result back in \( A \). This routine differs from A2A1 principally in that here the result overwrites \( A \). PERMUT is especially useful in applications where storage is at a premium or where the problem is of a recursive nature.

10. **PHIU** - (PHI (rectangular) * U(unit upper triangular))

\[
[ \text{PHI} ]\begin{bmatrix} \text{U} \end{bmatrix} = [ \text{PHIU} ]
\]

The matrices PHI and PHIU are double subscripted, and U is vector subscripted with implicitly defined unit diagonal elements. It is not
necessary to include trailing columns of zeros in the PHI matrix; they are accounted for implicitly. To economize on storage the output PHIU matrix can overwrite the input PHI matrix. For problems involving sparse PHI matrices it is more efficient to use the sparse matrix multiplication subroutine, SFU. When the last column of U contains the estimate, \( x \), the last column of W represents the mapped elements of PHI*\( x \). The principal use of this subroutine is the mapping of covariance U factors, where \( P = UDU^T \), and estimates.

11. **RA** - (R(triangular) * A(rectangular))

\[
\begin{pmatrix}
R \\
0
\end{pmatrix} \star \begin{bmatrix}
A \\
I
\end{bmatrix} = \begin{bmatrix}
RA
\end{bmatrix}
\]

Square root information matrix mapping involves matrix multiplication of the form indicated in the figure, i.e. with the bottom portion of A only implicitly defined as a partial identity matrix. Features of this subroutine are that the resulting RA matrix can overwrite the input A, and one can compute RA based on a trapezoidal input R matrix (i.e. only compute part of R*A).

12. **RANK1** - (U-D covariance factor rank 1 modification)

Computes updated U-D factors corresponding to a rank 1 matrix modification; i.e., given U-D, a scalar \( c \), and vector \( v \), \( \tilde{U} \) and \( \tilde{D} \) are computed so that \( \tilde{U} \tilde{D} \tilde{U}^T = UDU^T + cvv^T \). Both \( c \) and \( v \) are destroyed during the computation, and the resultant (vector stored) U-D array replaces the original one. Uses for this routine include (a) adding process noise effects to a U-D factored Kalman filter; (b) computing consider covariances (cf Section 11.5); (c) computing "actual" covariance factors resulting from the use of suboptimal Kalman filter gains; and (d) adding measurements to a U-D factored information matrix.
13. **RCOLRD** - (colored noise inclusion into the SRIF)

Includes colored noise time updating into the square root information matrix. It is assumed that the deterministic portion of the time update has been completed, and that only the colored noise effects are being incorporated by this subroutine. The algorithm used is Bierman's colored noise one-component-at-a-time update, cf ref. [3], and updates the SRIF array corresponding to the model

\[
\begin{bmatrix}
  x_1 \\
  p \\
  x_2
\end{bmatrix}
= 
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & M & 0 \\
  0 & 0 & I
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  p \\
  x_2
\end{bmatrix}
+ 
\begin{bmatrix}
  0 \\
  w_j \\
  0
\end{bmatrix}
\]

M is diagonal and \( w_j \in N(0, Q) \). Auxiliary quantities, useful for fixed interval smoothing, are also generated.

14. **RINCON** - (R inverse with condition number bound, CNB)

Computes the inverse of an upper triangular vector stored matrix \( R \) using back substitution. To economize on storage the output result can overwrite the input matrix. A Frobenius bound (CNB) for the condition number of \( R \) is computed too. This bound acts as both an upper and a lower bound, because \( CNB/N \leq \text{condition number} \leq CNB \). When this bound is within several orders of magnitude of the machine accuracy the computed inverse is not to be trusted, (viz if \( CNB > 10^{-15} \) on an 18 decimal digit machine \( R \) is ill-conditioned).

15. **RI2COV** - (RI to covariance)

This subroutine computes sigmas (standard deviations) and/or the covariance of a vector stored upper triangular square root covariance matrix, \( \text{RINV} \) (SRIF inverse). The result, stored in \( \text{COVOUT} \) (covariance output) is also vector stored. To economize on storage, \( \text{COVOUT} \) can overwrite \( \text{RINV} \).
16. **R2A - (R to A)**

The columns of a vector stored upper triangular matrix $R$ are permuted and variables are added and/or deleted. The result is stored in the double subscripted matrix $A$. In other respects the subroutine is like $A2A1$.

**Example III.5**

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>E</th>
<th>F</th>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>22</td>
<td>22</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>16</td>
<td>24</td>
<td>24</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>18</td>
<td>26</td>
<td>26</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>28</td>
<td>28</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>30</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$R$ is vector stored as $R = (2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30)$ with namelist $(\alpha, B, C, D, E)$ associated with it. Names $\alpha$ and $D$ are not included in matrix $A$, and a column of zeros corresponding to name $F$ is added.

One trivial, but perhaps useful, application is to convert a vector stored matrix to a double subscripted form.\[ R2A \] is used most often when one wants to rearrange the columns of a SRIF array so that reduced order estimates, sensitivities, etc. can be obtained; or so that data sets containing different parameters can be combined.

\[ \text{\[R2A\]} \]

\[ \text{\[see also the aside in the introduction} \]

\[ \text{\[\text{\[R2A\]} \text{\[see also the aside in the introduction} \]}} \]

29
17. **R2RA** - (Triangular block of R to triangular block of RA)

A triangular portion of the vector stored upper triangular matrix R is put into a triangular portion of the vector stored matrix RA. The names corresponding to the relocated block are also moved. R can coincide with RA.

**Examples III.6**

Note that an upper left triangular submatrix can slide to any lower position along the diagonal, but that a submatrix moving up must go to the upper leftmost corner. Upper shifting is used when one is interested in that subsystem; and the lower shifting is used, for example, when inserting a priori information for consider analyses.
18. **RUDR** - (SRIF R converted to U-D form or vice versa)

A vector stored SRIF array is replaced by a vector stored U-D form or conversely. A point to be noted is that when data is involved the right side of the SRIF data equation transforms to the estimate in the U-D array.

19. **SFU** - (Sparse F*U(Unit upper triangular))

\[
\begin{bmatrix}
\text{Sparse F} \\
\end{bmatrix}
\begin{bmatrix}
U \\
\end{bmatrix} = 
\begin{bmatrix}
F \\
U \\
\end{bmatrix}
\]

A sparse F matrix, with only its nonzero elements recorded, multiplies U which is vector stored with implicit unit diagonal entries. When the input F is sparse this routine is very efficient in terms of storage and computation. When the last column of U contains the estimate, x, the last column of FU represents elements of the mapped estimate F*x.

20. **TDHHT** - (Two dimensional Householder Triangularization)

Implicitly defined Householder orthogonal transformations are used to triangularize an input two dimensional rectangular array, S(M,N). Computation can be reduced if S starts partially triangular;

\[
S = \begin{bmatrix}
\end{bmatrix}
\]

Further, the algorithm implementation is such that (a) maximum triangularization is achievable

when M.LT.N

\[
S \rightarrow \begin{bmatrix}
\end{bmatrix}
\]
and finally when an intermediate form is desired

\[
S \rightarrow \begin{bmatrix}
0 \\
JSTOP
\end{bmatrix}
\]

This subroutine can be used to compress overdetermined linear systems of equations to triangular form (for use in least squares analyses). The application, that we have in mind, of this subroutine, is to the matrix triangularization of a "mapped" square root information matrix. This subroutine overlaps to a large extent the subroutine THH which utilizes vector stored, single subscripted, matrices. This latter routine when applicable is more efficient. The triangularization is adapted from ref. [1].

21. **THH** - (Triangular Householder data packing)

An upper triangular vector stored matrix R is combined with a rectangular doubly subscripted matrix A by means of Householder orthogonal transformations. The result overwrites R, and A is destroyed in the process. This subroutine is a key component of the square root information sequential filter, cf ref. [3].

\[
\begin{bmatrix}
R \\
A
\end{bmatrix} \rightarrow_{THH} \begin{bmatrix}
R \\
0^*
\end{bmatrix}
\]

*The elements are not explicitly set to zero.*
22. **TTHH** - (Two triangular arrays are combined using Householder orthogonal transformations)

This subroutine combines two single subscripted upper triangular SRIF arrays, R and RA using Householder orthogonal transformations. The result overwrites R.

\[
\begin{bmatrix}
R \\
RA
\end{bmatrix}
\overset{TTHH}{\rightarrow}
\begin{bmatrix}
R \\
0^\dagger
\end{bmatrix}
\]

23. **TWOMAT** - (Two dimensional print of a triangular matrix)

Prints a vector stored upper triangular matrix, using a matrix format.

**Example III.7**

\[
R(10) = (2, 4, 6, 8, 10, 12, 14, 16, 18, 20)
\]

with associated namelist \((A, B, C, D)\) is printed as

\[
\begin{array}{cccc}
A & B & C & D \\
2 & 4 & 8 & 14 \\
6 & 10 & 16 \\
12 & 18 \\
20
\end{array}
\]

(The numbers are printed as 7 columns of 8 significant floating point digits or 12 columns of 5 significant floating point digits.)

To appreciate the importance of this subroutine compare the vector \(R(10)\) with the double subscript representation.

\[\text{The elements are not explicitly set to zero.}\]
24. **TZERO** - (Zero a horizontal segment of a vector stored upper triangular matrix)

Upper triangular vector stored matrix R has its rows between ISTART and IFINAL set to zero.

**Example III.8**

To zero rows 2 and 3 of R(15) of example III.5

R(15) = (2,4,6,8,10,2,14,16,18,20,22,24,26,28,30) is transformed to

R(15) = (2,4,0,8,0,0,14,0,20,22,0,0,28,30) i.e.,

\[
\begin{bmatrix}
2 & 4 & 8 & 14 & 22 \\
0 & 6 & 0 & 16 & 24 \\
0 & 0 & 12 & 18 & 26 \\
0 & 0 & 0 & 20 & 28 \\
0 & 0 & 0 & 0 & 30
\end{bmatrix}
\rightarrow
\begin{bmatrix}
2 & 4 & 8 & 14 & 22 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 20 & 28 \\
0 & 0 & 0 & 0 & 30
\end{bmatrix}
\]

R-vector stored \hspace{1cm} R-vector stored

25. **UDCOL** - (U-D covariance factor colored noise update)

This subroutine updates the U-D covariance factors corresponding to the model

\[
\begin{bmatrix}
x_1 \\
p \\
x_2_{j+1}
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 \\
0 & M & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
p \\
x_2_j
\end{bmatrix} +
\begin{bmatrix}
0 \\
\nu_j \\
0
\end{bmatrix}
\]

where \(M\) is diagonal and \(\nu_j \sim \mathcal{N}(0,Q)\). The special structure of the transition and process noise covariance matrices is exploited, cf Bierman, [3].
26. **UDMEAS** - (U-D Measurement Update)

Given the U-D factors of the a priori estimate error covariance and the measurement, \( z = Ax + \nu \) this routine computes the updated estimate and U-D covariance factors, the predicted residual, the predicted residual variance, and the normalized Kalman gain. This is Bierman's U-D measurement update algorithm, cf [3].

27. **UD2COV** - (U-D factors to covariance)

The input vector stored U-D matrix (diagonal D elements are stored as the diagonal entries of U) is replaced by the covariance \( P \), also vector stored, \( P = U D U^T \). \( P \) can overwrite \( U \) to economize on storage.

28. **UD2SIG** - (U-D factors to sigmas)

Standard deviations corresponding to the diagonal elements of the covariance are computed from the U-D factors. This subroutine, a restricted version of UD2COV can print out the resulting sigmas and a title. The input U-D matrix is unaltered.

29. **UTINV** - (Upper triangular matrix inversion)

An upper triangular vector stored matrix RIN (R in) is inverted and the result, vector stored, is put in ROUT (R out). ROUT can overwrite RIN to economize on storage. If a right hand side is included and the bottommost tip of RIN has a -1 set in then ROUT will have the solution in the place of the right hand side.
30. **UTIROW** - (Upper triangular inversion, inverting only the upper rows)

\[
\begin{bmatrix}
R_x & R_{xy} \\
0 & R^{-1}_y
\end{bmatrix}
\rightarrow
\begin{bmatrix}
R^{-1}_x & -R^{-1}_x R_{xy} R^{-1}_y \\
0 & R^{-1}_y
\end{bmatrix}
\]

An input vector stored \( R \) matrix with its lower left triangle assumed to have been already inverted is used to construct the upper rows of the matrix inverse of the result. The result, vector stored, can overwrite the input to economize on storage.

If the columns comprising \( R_{xy} \) represent consider terms then taking \( R^{-1}_y \) as the identity gives the sensitivity on the upper right portion of the result. If \( R^{-1}_y = \text{Diag}(\sigma_y, \ldots, \sigma_{n_y}) \) then the upper right portion of the result represents the perturbation. Note that if \( z \) (the right hand side of the data equation) is included in \( R_{xy} \) then taking the corresponding \( R^{-1}_y \) diagonal as \(-1\) results in the filter estimate appearing as the corresponding column of the output array. When \( n_y \) is zero this subroutine is algebraically equivalent to **UTINV**. The subroutines differ when a zero diagonal is encountered. **UTINV** gives the correct inverse for the columns to the left of the zero element, whereas **UTIROW** gives the correct inverse for the rows below the zero element.
31. **WGS - (Weighted Gram-Schmidt U-D matrix triangularization)**

An input rectangular (possibly square) matrix \( W \) and a diagonal weight matrix, \( D_w \), are transformed to \( (U-D) \) form; i.e.,

\[
SD_wW^T = UDU^T
\]

where \( U \) is unit upper triangular and \( D \) is diagonal. The weights \( D_w \) are assumed nonnegative, and this characteristic is inherited by the resulting \( D \).
IV. SUBROUTINE DIRECTORY USER DESCRIPTION

1. A2A1 (A to A1)

**Purpose**

To rearrange the columns of a namelist indexed matrix to conform to a desired namelist.

```call a2a1(a,ia,ir,la,nama,a1,ial,la1,nama1)```

**Argument Definitions**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(IR,LA)</td>
<td>Input rectangular matrix</td>
</tr>
<tr>
<td>IA</td>
<td>Row dimension of A, IA.GE.IR</td>
</tr>
<tr>
<td>IR</td>
<td>Number of rows of A that are to be arranged</td>
</tr>
<tr>
<td>LA</td>
<td>Number of columns in A; this also represents the number of parameter names associated with A</td>
</tr>
<tr>
<td>NAMA(LA)</td>
<td>Parameter names associated with A</td>
</tr>
<tr>
<td>A1(IR,LA1)</td>
<td>Output rectangular matrix</td>
</tr>
<tr>
<td>IAL</td>
<td>Row dimension of A1, IAL.GE.IR</td>
</tr>
<tr>
<td>LA1</td>
<td>Number of columns in A1; this also represents the number of parameter names associated with A1</td>
</tr>
<tr>
<td>NAMA1(LA1)</td>
<td>Input list of parameter names to be associated with the output matrix A1</td>
</tr>
</tbody>
</table>

**Remarks and Restrictions**

A1 cannot overwrite A. This subroutine can be used to add on columns corresponding to new names and/or to delete variables from an array.

**Functional Description**

The columns of A are copied into A1 in an order corresponding to the NAMA1 parameter namelist. Columns of zeros are inserted in those A1 columns which do not correspond to names in the input parameter namelist NAMA.
2. COMBO (Combine parameter namelists)

Purpose

To rearrange a vector stored triangular matrix and store the result in matrix A. The difference between this subroutine and R2A is that there the namelist for A is input; here it is determined by combining the list for R with a list of desired names.

CALL COMBO (R,L1,NAM1,L2,NAM2,A,IA,LA,NAMA)

Argument Definitions

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(L1*(L1+1)/2)</td>
<td>Input vector stored upper triangular matrix</td>
</tr>
<tr>
<td>L1</td>
<td>No. of parameters in R (and in NAM1)</td>
</tr>
<tr>
<td>NAM1(L1)</td>
<td>Names associated with R</td>
</tr>
<tr>
<td>L2</td>
<td>No. of parameters in NAM2</td>
</tr>
<tr>
<td>NAM2(L2)</td>
<td>Parameter names that are to be combined with R (NAM1 list); these names may or may not be in NAM1</td>
</tr>
<tr>
<td>A(L1,LA)</td>
<td>Output array containing the rearranged R matrix L1.LE.IA</td>
</tr>
<tr>
<td>IA</td>
<td>Row dimension of A</td>
</tr>
<tr>
<td>LA</td>
<td>No. of parameter names in NAMA, and the column dimension of A. LA = L1 + L2 - No. names common to NAM1 and NAM2; LA is computed and output</td>
</tr>
<tr>
<td>NAMA(LA)</td>
<td>Parameter names associated with the output A matrix; consists of names in NAM1 which are not in NAM2, followed by NAM2</td>
</tr>
</tbody>
</table>

Remarks and Restrictions

The column dimension of A is a result of this subroutine.

To avoid having A overwrite neighboring arrays one can bound the column dimension of A by L1 + L2.
**Functional Description**

First the NAM1 and NAM2 lists are compared and the names appearing in NAM1 only have their corresponding R column entries stored in A (e.g. if NAM1(2) and NAM1(6) are the only names not appearing in the NAM2 list then columns 2 and 6 of R are copied into columns 1 and 2 of A). The remaining columns of A are labeled with NAM2. The A namelist is recorded in NAMA. The NAM1 list is compared with NAM2 and matching names have their R column entries copied into the appropriate columns of A. NAM2 entries not appearing in NAM1 have columns of zero placed in A.
3. COVRHO (Covariance to correlation matrix, RHO)

Purpose

To compute the correlation matrix RHO from an input covariance matrix COV. Both matrices are upper triangular, vector stored and the output can overwrite the input.

CALL COVRHO(COV,N,RHO,V)

Argument Definitions

- COV(N*(N+1)/2) Input vector stored positive semi-definite covariance matrix
- N Model dimension, N.GE.1
- RHO(N*(N+1)/2) Output vector stored correlation matrix
- V(N) Work vector

Remarks

No test for non-negativity of the input matrix is made. Correlations corresponding to negative or zero diagonal entries are set to zero, as is the diagonal output entry.

Functional Description

\[ V(I) = \frac{1}{\sqrt{COV(I,I)}} \text{ if } COV(I,I) > 0 \text{ and } 0 \text{ otherwise} \]

\[ RHO(I,J) = COV(I,J) \times V(I) \times V(J) \]

The subroutine employs, however, vector stored COV and RHO matrices.
4. COV2RI (Covariance to Cholesky Square Root, RI)

**Purpose**

To construct the upper triangular Cholesky factor of a positive semi-definite matrix. Both the input covariance and the output Cholesky factor (square root) are vector stored. The output overwrites the input. Covariance (input) = (CF)*(CF)**T

(output CF = Rinverse). If the input covariance is singular, the output factor has zero columns.

```CALL COV2RI(CF,N)```

**Argument Definitions**

- **CF(N*(N+1)/2)**: Contains the input vector stored covariance matrix (assumed positive definite) and on output it contains the upper triangular Cholesky factor.

- **N**: Dimension of the matrices involved, N.GE.2

**Remarks and Restrictions**

No check is made that the input matrix is positive semi-definite. Singular factors (with zero columns) are obtained if the input is (a) in fact singular, (b) ill-conditioned, or (c) in fact indefinite; and the latter two situations are cause for alarm. Case (c) and possibly (b) can be identified by using R12COV to reconstruct the input matrix.

**Functional Description**

An upper triangular Cholesky reduction of the input matrix is implemented using a geometric algorithm described in Ref. [3].

\[
CF(\text{input}) = CF(\text{output})^T \cdot CF(\text{output})^T
\]

At each step of the reduction diagonal testing is used and negative terms are set to zero.
5. COV2UD (Covariance to UD factors)

Purpose

To obtain the U-D factors of a positive semi-definite matrix.

The input vector stored matrix is overwritten by the output U-D factors which are also vector stored.

CALL COV2UD(U,N)

Argument Definitions

U(N*(N+1)/2) Contains the input vector stored covariance matrix; on output it contains the vector stored U-D covariance factors.

N Matrix dimension, N,GE.2

Remarks and Restrictions

No checks are made in this routine to test that the input U matrix is positive semi-definite. Singular results (with zero columns) are obtained if the input is (a) in fact singular, (b) ill-conditioned, or (c) in fact indefinite; and the latter two situations are cause for alarm. Case (c) and possibly case (b) can be identified by using UD2-COV to reconstruct the input matrix. Note that although indefinite matrices have U-D factorizations, the algorithm here applies only to matrices with non-negative eigenvalues.

Functional Description

An upper triangular U-D Cholesky factorization of the input matrix is implemented using a geometric algorithm described in Ref. [3].

\[ U(\text{input}) = U^TDU^T \]

U-D overwrites the input U at each step of the reduction diagonal testing is used to zero negative terms.
6. C2C (C to C)

Purpose

To rearrange the rows and columns of C, from NAM1 order to NAM2 order. Zero rows and columns are associated with output defined names that are not contained in NAM1.

CALL C2C(C,IC,L1,NAM1,L2,NAM2)

Argument Definitions

C(L1,L1) Input matrix
IC Row dimension of C
IC.GE.L = MAX(L1,L2)
L1 No. of parameter names associated with the input C
NAM1(L) Parameter names associated with C on input. (Only the first L1 entries apply to the input C)
L2 No. of parameter names associated with the output C
NAM2(L2) Parameter names associated with the output C

Remarks and Restrictions

The NAM2 list need not contain all the original NAM1 names and L1 can be .GE. or .LE. L2. The NAM1 list is used for scratch and appears permuted on output. If L2.GT.L1 the user must be sure that NAM1 has L2 entries available for scratch purposes.

Functional Description

The rows and columns of C and NAM1 are permuted pairwise to get the names common to NAM1 and NAM2 to coalesce. Then the remaining rows and columns of C(L2,L2) are set to zero.
7. HHPOST (Householder Post Multiplication Triangularization)

Purpose
To employ Householder orthogonal transformations to triangularize
an input rectangular $W$ matrix by post multiplication, i.e.
$$
\begin{bmatrix}
W
\end{bmatrix}^T = \begin{bmatrix} 0 \\ S \end{bmatrix}
$$
This algorithm is employed in various covariance square root updates.

CALL HHPOST(S,W,MROW,NROW,NCOL,V)

Argument Definitions

- $S(NROW*(NROW+1)/2)$: Output upper triangular vector stored square root matrix
- $W(NROW,NCOL)$: Input rectangular square root covariance matrix ($W$ is destroyed by computations)
- $MROW$: Maximum row dimension of $W$
- $NROW$: Number of rows of $W$ to be triangularized and the dimension of $S$ ($NROW \geq 2$)
- $NCOL$: Number of columns of $W$ ($NCOL \geq NROW$)
- $V(NCOL)$: Work vector

Functional Description

Elementary Householder transformations are applied to the rows of $W$
in much the same way as they are applied to obtain subroutine TH9. The
orthogonalization process is discussed at length in the books by Lawson
and Hanson [1] and Bierman [3].
8. **INF2R** (Information matrix to R)

**Purpose**

To compute a lower triangular Cholesky factorization of an input positive semi-definite matrix. The result transposed, is vector stored; this is the form of an upper triangular SRIF matrix.

```
CALL INF2R(R,N)
```

**Argument Definitions**

- \( R(N*(N+1)/2) \) Input vector stored positive semi-definite (information) matrix; on output it represents the transposed lower triangular Cholesky factor (i.e. the SRIF R matrix)
- \( N \) Matrix dimension, \( N \geq 2 \)

**Remarks and Restrictions**

No checks are made on the input matrix to guard against negative eigenvalues of the input, or to detect ill-conditioning. Singular output matrices have one or more rows of zeros.

**Functional Description**

A Cholesky type lower triangular factorization of the input matrix is implemented using the geometric formulation described in Ref. [3].

\[
R(\text{input}) = (R(\text{output}))^T \ast [R(\text{output})]^{-1}
\]

At each step of the factorization diagonal testing is used to zero columns corresponding to negative entries. The result is vector stored in the form of a square root information matrix as it would be used for SRIF analyses.
9. **PERMUT** (Permute A)

**Purpose**

To rearrange the columns of a namelist indexed matrix to conform to a desired namelist. The resulting matrix is to overwrite the input.

```
CALL PERMUT(A,IA,IR,L1,NAM1,L2,NAM2)
```

**Argument Definitions**

- **A(IR,L)**: Input rectangular matrix, \( L = \max(L1,L2) \)
- **IA**: Row dimension of \( A \), \( IA \geq IR \)
- **IR**: Number of rows of \( A \) that are to be rearranged
- **L1**: Number of parameter names associated with the input \( A \) matrix
- **NAM1(L)**: Parameter names associated with \( A \) on input (only the first \( L1 \) entries apply to the input \( A \))
- **L2**: Number of parameter names associated with the output \( A \) matrix
- **NAM2**: Parameter names associated with the output \( A \)

**Remarks and Restrictions**

This subroutine is similar to \( A2A1 \); but because the output matrix in this case overwrites the input there are several differences. The \( NAM1 \) vector is used for scratch, and on output it contains a permutation of the input \( NAM1 \) list. The user must allocate \( L = \max(L1,L2) \) elements of storage to \( NAM1 \). The extra entries, when \( L2 > L1 \), are used for scratch.

**Functional Description**

The columns of \( A \) are rearranged, a pair at a time, to match the \( NAM2 \) parameter namelist. The \( NAM1 \) entries are permuted along with the columns, and this is why \( \dim(NAM1) \) must be larger than \( L1 \) (when \( L2>L1 \)). Columns of zeroes are inserted in \( A \) which correspond to output names that do not appear in \( NAM1 \).
10. PHIU (PHI-rectangular matrix PHI-tiangular U-upper triangular vector stored)

Purpose

To multiply a rectangular two dimensional matrix PHI by a unit upper triangular vector stored matrix U, and store the result in PHIU. The PHI matrix can overwrite PHI to economize on storage.

\[
\begin{bmatrix} \text{PHI} \end{bmatrix} \begin{bmatrix} U \end{bmatrix} = \begin{bmatrix} \text{PHIU} \end{bmatrix}
\]

CALL PHIU(PHI,MAXPHI,JRPHI,JCPHI,U,N,PHIU,MPHJU)

Argument Definitions

PHI(IRPHI,JCPHI) Input rectangular matrix IRPHI.LE.MAXPHI
MAXPHI Row dimension of PHI
IRPHI number of rows of PHI
JCPHI number of columns of PHI
U(N*(N+1)/2) unit upper triangular vector stored matrix
N U-matrix dimension, JCPHI.LE.N
PHIU(IRPHI,N) output result PHI*U,PHIU can overwrite PHI
MPHIU row dimension of PHIU

Remarks and Restrictions

If JCPHI.LT.N it is assumed that there are implicitly defined trailing columns of zeros in PHI. The unit diagonal entries of U are implicit, i.e. the diagonal U entries are not explicitly used.

Functional Description

PHIU = PHI*U
11. RA (R-upper triangular*A-rectangular)

Purpose

To post multiply a vector stored triangular matrix, R, by a rectangular matrix A, and if desired to store the result in A.

\[
\begin{bmatrix} R \end{bmatrix} \begin{bmatrix} A \end{bmatrix} = \begin{bmatrix} RA \end{bmatrix}
\]

CALL RA(R,N,A,MAXA,IA,JA,RA,MAXRA,IRA)

Argument Definitions

- **R(N*(N+1)/2)**: upper triangular, vector stored input
- **N**: order of R
- **A(IA,JA)**: Input rectangular right multiplier matrix
- **MAXA**: Row dimension of input A matrix
- **IA**: Number of rows of A that are input
- **JA**: Number of columns of A
- **RA(IRA,JA)**: Output resulting rectangular matrix
  - RA can overwrite A
- **MAXRA**: Row dimension of RA
- **IRA**: Number of rows in the output result (IRA .LE. MAXRA)

Functional Description

The first IRA rows of the product R*A are computed using the vector stored input matrix R, and the output can, if desired, overwrite the input A matrix. When N.GT.IA (i.e. there are more columns of R than rows of A) then it is assumed that the bottom N-IA rows of A are implicitly defined as a partial identity matrix, i.e.

\[
A = \begin{bmatrix}
-(\text{Input}) & \cdot & \cdot \\
0 & \cdot & \cdot \\
\end{bmatrix}_{IA} \quad \begin{bmatrix}
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
\end{bmatrix}_{N-IA}
\]
12. RANK1 (Stable U-D rank one update)

Purpose

To compute the (updated) U-D factors of $UDU^T + Cvv^T$.

CALL RANK1(UIN,UOUT,N,C,V)

Argument Definitions

UIN(N*(N+1)/2)  Input vector stored positive semi-definite U-D array (with the D entries stored on the diagonal of U)

UOUT(N*(N+1)/2)  Output vector stored positive (possibly) semi-definite U-D result, UOUT=UIN is allowed.

N  Matrix dimension, N.GE.2

C  Input scalar, which should be non-negative. C is destroyed by the algorithm.

V(N)  Input vector for the rank one modification. V is destroyed by the algorithm.

Remarks and Restrictions

If C negative is used the algorithm is numerically unstable, and the result may be numerically unreliable. Singular U matrices are allowed, and these can result in singular output U Matrices. The code switches from a 1-multiply to a 2-multiply mode at a key place, based upon a 1/16 comparison of input to output D values. Also, there is provision made to supply a machine accuracy epsilon when single precision is specified.

Functional Description

This rank one modification is based on a result published by Agee and Turner (1972), White Sands Missile Range Tech. Report No. 38 and improved on using a numerical stabilization idea due to Gentlemen (1973). The algorithm is derived in the chapter,
13. **RCOLRD** (Colored noise time update of the SRIF R matrix)

**Purpose**

To include colored noise time updating into the square root information matrix. It is assumed that the deterministic portion of the time update has been completed, and that only the colored noise effects are being incorporated by this subroutine.

**CALL** RCOLRD(S,MAXS,IRS,JCS,NPSTRT,NP,EM,RW,ZW,V,SGSTAR)

**Argument Definitions**

- **S**<sub>(IRS,JCS)</sub>: Input rectangular portion of the square root information matrix corresponding to the nonconstant parameters. It is assumed that estimates are included, i.e. the last column represents the "right hand side", Z. (but see JCS description). S also houses the time updated array, and if there is smoothing there are NP extra rows adjoined to S.

- **MAXS**: Row dimension of S. If smoothing calculations are to be included then MAXS.GE.IRS+NP.

- **IRS**: The number of rows of S, i.e. the number of nonconstant parameters (including colored noise variables). IRS.GE.2

- **JCS**: The number of columns of S. If the vector ZN is zero, then the right hand side of transformed estimates need not be included.

- **NPSTRT**: Location of the first colored process noise variable.

- **NP**: The number of colored noise variables contiguous to and following the first.

- **EM**(NP): Vector of exponential colored noise multipliers \( EM = \exp\left(-\frac{DT}{TAU}\right) \)

- **RW**(NP): Vector of positive reciprocal colored process noise standard deviations, i.e. \( p_{j+1} = \exp\left(-\frac{DT}{\tau}\right) \cdot p_j + \omega_j, \text{ RW } = 1/\omega \)
$ZW(NP)$ Vector of normalized process noise a priori estimates. $ZW$ is generally zero.

$V(IRS)$ Work vector.

$SGSTAR(NP)$ Vector of smoothing coefficients. Needed only if smoothing is to be done.

**Remarks and Restrictions**

There are three lines of code associated with smoothing, and these are commented out of the nominal case. Therefore, if smoothing is contemplated the comments must be removed. The vector $SGSTAR$ is involved only with smoothing. Last note: for smoothing, be sure that $S$ has $NP$ extra rows to house the smoothing coefficients.

The $ZW$ vector is generally zero. If $ZW = 0$ one has the option of doing covariance only analyses and the last column of $S$ (the right hand side of normalized estimates) can be omitted.

Because of the large number of arguments appearing in this subroutine, and because almost all of them are constant (i.e. with succeeding calls only $S$, and possible EM, RW, ZW and SGSTAR change) for a given problem, it is suggested that one a) introduce COMMON, b) use this as an internal subroutine, or c) write in-line code.

**Functional Description**

The model is

$$\begin{bmatrix}
x_1 \\
p \\
x_2 \\
\end{bmatrix}_{j+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & I \\ \end{bmatrix} \begin{bmatrix}
x_1 \\
p \\
x_2 \\
\end{bmatrix}_j + \begin{bmatrix} 0 \\ w_j \\ 0 \\ \end{bmatrix}_{NPSTRT-1}$$

$$\begin{bmatrix}
x_1 \\
p \\
x_2 \\
\end{bmatrix}_{j+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & I \\ \end{bmatrix} \begin{bmatrix}
x_1 \\
p \\
x_2 \\
\end{bmatrix}_j + \begin{bmatrix} 0 \\ w_j \\ 0 \\ \end{bmatrix}_{NPSTRT-1+NP}$$

where $M$ is diagonal, with $NP$ non-negative entries and $w_j$ is a white noise process with $w_j \in N(\bar{\sigma}, Q), Q = R_w^{-1} R_w^{-T}$. The algorithm is based on Bierman's one component-at-a-time SRIF time update which economizes

When smoothing is contemplated, there is output a vector $\sigma^*(NP)$ and a matrix $S^*(NP,N+1)$; $S^*$ occupies the bottom $NP$ rows of the output $S$ matrix. Smoothed estimates of the $p$ terms can be obtained from the $\sigma^*$ and $S^*$ terms as follows:

Let $X^*$ be the previously computed estimates of the $N$ filter parameters, then for $J = NP, NP-1, \ldots 1$ recursively compute

$$X^*(\text{MSTRT} + J-1) := (S^*(J, N+1) - \sum_{K=1}^{N} S^*(J,K)X^*(K))/\sigma^*(J)$$

Note that the symbol "\:=I1 means is replaced by, so that the old values of $X^*$, on the right side, are over-written by the new smoothed colored noise estimates. Smoothed covariances can be obtained from the $S^*$ and $\sigma^*$ terms as well, but we do not go into detail here; the reader is directed to chapter 10 of the Bierman reference.
14. RINCON (R inverse with condition number bound)

Purpose

To compute the inverse of an upper triangular vector stored triangular matrix, and an estimate of its condition number.

```
CALL RINCON(RIN,N,ROUT,CNB)
```

Argument Definitions

- **RIN(N*(N+1)/2)**: Input vector stored upper triangular matrix
- **N**: Matrix dimension, N.GE.2
- **ROUT(N*(N+1)/2)**: Output vector stored matrix inverse (RIN = ROUT is permitted)
- **CNB**: Condition number bound. If \( \kappa \) is the condition number of RIN, then \( \frac{CNB}{N.\LE.\kappa.\LE. CNB} \)

Remarks and Restrictions

The condition number bound, CNB serves as an estimate of the actual condition number. When it is large the problem is ill-conditioned.

Functional Description

The matrix inversion is carried out using a triangular back substitution. If any diagonal element of the input R matrix is zero the condition number computation is aborted. When the first zero occurs at diagonal k the matrix inversion is carried out only on the first k-1 columns. The condition number bound is computed as follows:

\[
F.NORM \, R = \sum_{J=1}^{NTOT} R(J)^2
\]

\[
F.NORM \, R^{-1} = \sum_{J=1}^{NTOT} R^{-1}(J)^2
\]
where $NTOT = \frac{N(N+1)}{2}$ is the number of elements in the vector stored triangular matrix. The condition number bound, $CNB$, is given by

$$CNB = (F.NORM R \ast F.NORM R^{-1})^{1/2}$$

$F.NORM$ is the Frobenius norm, squared. The inequality

$$CNB/N \leq \text{condition number } R \leq CNB$$

is a simple consequence of the Frobenius norm inequalities given in Lawson-Hanson "Solving Least Squares," page 234.
15. RI2COV (RI Triangular to covariance)

Purpose

To compute the standard deviations, and if desired, the covariance matrix of a vector stored upper triangular square root covariance matrix. The output covariance matrix, also vector stored, can overwrite the input.

CALL RI2COV(RINV,N,SIG,COVOUT,KROW,KCOL)

Argument Definitions

RINV(N*(N+1)/2) Input vector stored upper triangular covariance square root (RINV=Inverse is the inverse of the SRIF matrix).

N Dimension of the RINV matrix

SIG(N) Output vector of standard deviations

COVOUT(N*(N+1)/2) Output vector stored covariance matrix (COVOUT = RINV is allowed)

. GT.0 Computes the covariance and sigmas corresponding to the first KROW variables of the RINV matrix

. LT.0 Computes only the sigmas of the first (KROW) variables of the RINV matrix.

. EQ.0 No covariance, but all sigmas (e.g. use all N rows of RINV)

KCOL Number of columns of COVOUT that are computed, If KCOL.LE.0, then KCOL = KROW.

Remarks and Restrictions

Replacing N by |KROW| corresponds to computing the covariance of a lower dimensional system.

Functional Description

COVOUT=RI2V*RI2V**T
16. R2A (R to A)

**Purpose**

To place the upper triangular vector stored matrix R into the matrix A and to arrange the columns to match the desired NAMA parameter list. Names in the NAMA list that do not correspond to any name in NAMR have zero entries in the corresponding A columns.

```
CALL R2A(R,LR,NAMR,A,IA,LA,NAMA)
```

**Argument Definitions**

- **R(LR*(LR+1)/2)**: Input upper triangular vector stored array
- **LR**: No. of parameters associated with R
- **NAMR(LR)**: Parameter names associated with R
- **A(LR,LA)**: Matrix to house the rearranged R matrix
- **IA**: Row dimension of A, IA.GE.LR.
- **LA**: No. of parameter names associated with the output A matrix.
- **NAMA(LA)**: Parameter names for the output A matrix.

**Functional Description**

The matrix A is set to zero and then the columns of R are copied into A.
17. R2RA (Permute a subportion R_A of a vector stored triangular matrix)

Purpose

To copy the upper left (lower right) portion of a vector stored upper triangular matrix R into the lower right (upper left) portion of a vector stored triangular matrix RA.

CALL R2RA(R,NR,NAM,RA,NRA,NAMA)

Argument Definitions

<table>
<thead>
<tr>
<th>R(NR*(NR+1)/2)</th>
<th>Input vector stored upper triangular matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>NR</td>
<td>Dimension of vector stored R matrix†</td>
</tr>
<tr>
<td>NAM(NR)</td>
<td>Names associated with R.</td>
</tr>
<tr>
<td>RA(NRA*(NRA+1)/2)</td>
<td>Output vector stored upper triangular matrix</td>
</tr>
<tr>
<td>NRA</td>
<td>If NRA = 0 on input, then NAMA(1) should have the first name of the output namelist. In this case the number of names in NAMA, NRA, will be computed. The lower right block of R will be the upper left block of RA. If NRA = last name of the upper left block that is to be moved then this upper block is to be moved to the lower right corner of RA. When used in this mode NRA=NR on output†.</td>
</tr>
<tr>
<td>NAMA(NRA)</td>
<td>Names associated with RA. Note that NRA used here denotes the output value of NRA.</td>
</tr>
</tbody>
</table>

Remarks and Restrictions

RA and NAMA can overwrite R and NAM. The meaning of the NRA = 0 option is clarified by the following example:

INPUT
NR = 5
NAM = 'A', 'B', 'C', 'D', 'E'
NRA = 0
NAMA(1) = 'C'

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>INPUT</th>
<th>R</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>22</td>
<td></td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>16</td>
<td>24</td>
<td></td>
<td></td>
<td>20</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>18</td>
<td>26</td>
<td></td>
<td>30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td>RA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† see the concluding paragraph of Remarks and Restrictions
When NRA = 0 and NAMA(1) = 'C' we are asking that the lower triangular portion of R, beginning at the column labeled C, be moved to form the first (in this case 3) columns of RA. Incidentally, RA could have additional columns; these columns and their names would be unaltered by the subroutine.

The meaning of the other NRA option is illustrated by the following example:

```
INPUT
NR = 5
NAM = ['A', 'B', 'C', 'D', 'E'
NRA = 'C'
R

OUTPUT
NR = 5
NAMA(3-5) = ['A', 'B', 'C'
RA
```

When NRA = 'C' we are asking that the upper left block of R, up to the column labeled C, be moved to the lower right portion of RA and the corresponding names be moved too. If RA overwrites R, as in the example, then the first two rows of R remain unchanged and since NAMA overwrites NAM, the labels of the first two columns remain unaltered.

The remark that NRA=NR on output means, in this example, that the column with name C in R is moved over to column 5. If one wanted to slide the upper left triangle corresponding to names ABC of R to columns 7-9 of an RA matrix (of unspecified dimension, ≥ 9), then one should set NR=9 in the subroutine call. Thus NR, when used in this sliding down the diagonal mode, does not represent the dimension of R; but indicates how far the slide will be.
18. RUDR  (R to U-D or U-D to R)

Purpose

To transform an upper triangular vector stored SRIF array to U-D form or vice versa.

CALL RUDR(RIN,N,ROUT,IS)

Argument Definitions

RIN(NBAR*(NBAR+1)/2)  Input upper triangular vector stored SRIF or U-D array; NBAR = ABS(N) + 1

ROUT(NBAR*(NBAR+1)/2)  Output upper triangular vector stored U-D or SRIF array (RIN = ROUT is permitted)

N  Matrix dimension, N.GT.0 represents an R to U-D conversion and N.LT.0 represents a U-D to R conversion. ABS(N).GE.2

IS  If IS = 0 the input array is assumed not to contain a right side (or an estimate), and IS = 1 means an appropriate additional column is included. In the IS = 0 case the last column of RIN is ignored and NBAR = ABS(N) is used.

Subroutine used: RINCON

Functional Description

Consider the N>0 case. RIN = R is transformed to ROUT = R inverse using subroutine RINCON with dimension N+IS. If IS = 1 the subroutine sets RIN((N+1)(N+2)/2) = -1, so that the N+1st column of ROUT will be the X estimate followed by -1. R^(-1) = UD^1/2 so that the diagonals are square root scaled U columns. This information is used to construct the U-D array which is written in ROUT.

If N<0 the input is assumed to be a U-D array. This array is converted to ROUT = UD^1/2 and then using RINCON, R is computed and stored in ROUT. If IS = 1 the U-D matrix is assumed augmented by X (estimate), and on output the right side term of the SRIF array is obtained. When IS = 1, the initial value of RIN((N+1)(N+2)/2) is restored before exiting the subroutine.
19. **SFU** (Sparse F * unit upper triangular U)

**Purpose**

To efficiently form the product \( F^*U \) so that only the nonzero elements of \( F \) are employed and so that the structure of the \( U \) matrix is utilized (upper triangular with implicit unit diagonal elements). When \( F \) is sparse there are significant savings in storage and computation. Not that since we deal only with the nonzero elements of \( F \) we are saved the time associated with computing unnecessary \( F \) matrix element addresses.

```
CALL SFU(FEL, IROW, JCOL, NF, U, N, FU, MAXFU, IFU, JDIAG)
```

**Argument Definitions**

- **FEL(NF)**: Values of the non-zero elements of the \( F \) matrix
- **IROW(NF)**: Row indices of the \( F \) elements
- **JCOL(NF)**: Column indices of the \( F \) elements
  \[ F(IROW(K), JCOL(K)) = FEL(K) \]
- **NF**: The number of non-zero elements of the \( F \) matrix
- **U(N*(N+1)/2)**: Upper triangular, vector stored matrix with implicitly defined unit diagonal elements. Note that \( U(JJ) \) terms are not, in fact, unity.
- **N**: Dimension of the \( U \) matrix
- **FU(IFU,N)**: The output result
- **MAXFU**: Row dimension of the \( FU \) matrix
- **IFU**: Number of rows in \( FU \). \( IFU \leq MAXFU \), and \( IFU \geq 1 \); i.e. \( FU \) must have at least as many rows as does \( F \). Additional rows of \( FU \) could correspond to zero rows of \( F \).
- **JDIAG(N)**: Diagonal element indices of a vector stored upper triangular matrix, i.e. \( JDIAG(K) = K*(K+1)/2 - JDIAG(K-1) + K \).
Example:

\[ F(3,12) \text{ with: } F(1,1) = .9, \ F(2,2) = .8, \ F(3,3) = 1.1, \]
\[ F(1,7) = 1.7, \ F(2,8) = -2.8 \text{ and } F(3,11) = 3.11. \]

In this case \( F \) has \( \text{NF} = 6 \) (nonzero elements); and one may take

\[
\begin{align*}
\text{IROW}(1) &= 1 & \text{JCOL}(1) &= 1 & \text{FEL}(1) &= .9 \\
\text{IROW}(2) &= 2 & \text{JCOL}(2) &= 2 & \text{FEL}(2) &= .8 \\
\text{IROW}(3) &= 3 & \text{JCOL}(3) &= 3 & \text{FEL}(3) &= 1.1 \\
\text{IROW}(4) &= 1 & \text{JCOL}(4) &= 7 & \text{FEL}(4) &= 1.7 \\
\text{IROW}(5) &= 2 & \text{JCOL}(5) &= 8 & \text{FEL}(5) &= -2.8 \\
\text{IROW}(6) &= 3 & \text{JCOL}(6) &= 11 & \text{FEL}(6) &= 3.11 \\
\end{align*}
\]

**Remarks and Restrictions**

Comments regarding increased efficiency are included in the code.

**Functional Description**

We write

\[
F = \sum_{i,j} F_{ij} e_i e_j^T
\]

where \( e_i \) is the \( i \)-th unit vector. Then

\[
F_U = \sum_{ij} F_{ij} e_i (e_j^T U)
\]

The code is based on this equation.
20. **TDHHT** (Two dimensional Householder triangularization)

**Purpose**

To transform a two dimensional rectangular matrix to a triangular, or partially triangular form by Householder orthogonal matrix pre-multiplication. This subroutine can be used to compress overdetermined linear systems to triangular (double subscripted form) in much the same way as does the subroutine **THH** (which outputs a vector subscripted triangular result). For recursive applications **THH** is computationally more efficient and requires less storage.

The chief application, that we have in mind, for this subroutine is to the matrix triangularization of "mapped" square root information matrices of the form $S(m,n)$ with $m$ less than $n$.

```
CALL TDHHT(S,MAXS,IRS,JCS,JSTART,JSTOP,V)
```

**Argument Definitions**

- **S** (IRS, JCS) Input (possibly partially) triangular matrix. The output (possibly partially) triangular result overwrites the input.
- **MAXS** Row dimension of S matrix
- **IRS** Number of rows in S (IRS.LE.MAXS), and IRS.GE.2.
- **JCS** Number of columns in S
- **JSTART** Index of first column to be triangularized. If JSTART.LT.1 then it is assumed that the triangularization starts at column 1.
- **JSTOP** Index of last column to be triangularized. When JSTOP is not between max(1,JSTART) and JCS then the triangularization is carried out as far as possible (i.e. to IRS if S has less rows than columns, or to JCS if it has more rows than columns).
- **V** (IRS) Work vector
Remarks and Restrictions

The indices JSTART and JSTOP are input for efficiency purposes. When it is known that the input matrix is partially triangular one can by-pass the corresponding (initial) Householder reduction steps. Further, for certain applications it is not necessary to totally triangularize the input array. For example if \( S(m,n) \) and \( m \) is less than \( n \), the system is in triangular form after only \( m \) elementary Householder reduction steps, i.e

\[
T \left[ \begin{array}{c}
S \\
\end{array} \right] \xrightarrow{m} \begin{array}{c}
0 \\
n \\
m \\
\end{array}
\]

The code is set up so that it defaults to the largest possible upper triangularization.

Functional Description

The dotted portion of the matrix and the block of zeros are not employed at all in the computations. The input matrix is transformed to (possibly partially) triangular form by premultiplication by a sequence of elementary Householder orthogonal transformations.
The method is described fully in the books by Lawson and Hanson - Solving Least Squares Problems, and in Bierman - Factorization Methods for Discrete Sequential Estimation.
Purpose

To compute \([R; z]\) such that

\[
T \begin{bmatrix}
R & z \\
A & z
\end{bmatrix} = \begin{bmatrix}
\hat{R} & \hat{z} \\
0 & 0
\end{bmatrix}
\]

This is the key algorithm used in the square root information batch sequential filter.

\[
\text{CALL THH}(R, N, A, IA, M, RSOS, NSTRT)
\]

Argument Definitions

- **R**\((N^2(N+3)/2)\)
  - Input upper triangular vector stored square root information matrix. If estimates are involved RSOS.GE.0 and \(R\) is augmented with the right hand side (stored in the last \(N\) locations of \(R\)). If RSOS.LT.0 only the first \(N^2(N+1)/2\) locations of \(R\) are used. The result of the subroutine overwrites the input \(R\).

- **N**
  - Number of parameters

- **A**\((M, N+1)\)
  - Input measurement matrix. The \(N+1\)st column is only used if RSOS.GE.0, in which case it represents the right side of the equation \(v + AX = z\). \(A\) is destroyed by the algorithm, but it is not explicitly set to zero.

- **IA**
  - Row dimension of \(A\)

- **M**
  - The number of rows of \(A\) that are to be combined with \(R\) (\(M\LE.\IA\))

- **RSOS**
  - Accumulated residual root sum of squares corresponding to the data processed prior to this time. On exit RSOS represents the updated root sum of squares of the residuals \(\sum_{i=1}^{n} z_i \hat{z}_i - A \hat{X}_i^\text{est} \|^2 \), summed over the old and new data. It also includes the a priori term.
\[ \| \mathbf{R} \mathbf{x}_{\text{est}} - z_0 \|^2 \]. Because RSOS cannot be used if data, \( z \), is not included we use RSOS.LT.0 to indicate when data is not included.

\( \text{NSTART} \)

First column of the input \( \mathbf{A} \) matrix that has a nonzero entry. In certain problems, especially those involving the inclusion of a priori statistics, it is known that the first \( \text{NSTART}-1 \) columns of \( \mathbf{A} \) all have zero entries. This knowledge can be used to reduce computation. If nothing is known about \( \mathbf{A} \), then \( \text{NSTART} \leq 1 \) gives a default value of 1, i.e. it is assumed that \( \mathbf{A} \) may have nonzero entries in the very first column.

**Remarks and Restrictions**

It is trivial to arrange the code so that \( \mathbf{R} \) output need not overwrite the input \( \mathbf{R} \). This was not done because, in the author's opinion, there are too few times when one desires to have \( \text{ROUT} \neq \text{RIN} \).

**Functional Description**

Assume for simplicity that \( \text{NSTART} = 1 \). Then at step \( j, j = 1, \ldots, N \) (or \( N+1 \) if data is present) the algorithm implicitly determines an elementary Householder orthogonal transformation which updates row \( j \) of \( \mathbf{R} \) and all the columns of \( \mathbf{A} \) to the right of the \( j \)th. At the completion of this step column \( j \) of \( \mathbf{A} \) is in theory zero, but it is not explicitly set to zero. The orthogonalization process is discussed at length in the books by Lawson and Hanson - *Solving Least Squares Problems* and Bierman - *Factorization Methods for Discrete Sequential Estimation*. 
22. TTHH  (Two triangular matrix Householder reduction)

**Purpose**

To combine two vector stored upper triangular matrices, R and RA by applying Householder orthogonal transformations. The result overwrites R.

\[
\begin{bmatrix}
R \\
\vdots \\
RA
\end{bmatrix}
\xrightarrow{TTHH}
\begin{bmatrix}
R \\
\vdots \\
Q
\end{bmatrix}
\]

**Argument Definitions**

- **R** (\(N*(N+1)/2\))
  - Input vector stored upper triangular matrix, which also houses the result
- **RA** (\(N*(N+1)/2\))
  - Second input vector stored upper triangular matrix. This matrix is destroyed by the computation.
- **N**
  - Matrix dimension
  
  - N less than zero is used to indicate that R and RA have right sides (|N|+1 columns) and have dimension \(|N|*(|N|+3)/2\).

**Remarks and Restrictions**

- RA is theoretically zero on output, but is not set to zero.
23. TWOMAT (Triangular matrix print)

Purpose

To display a vector upper triangular matrix in a two dimensional triangular format. Precision output corresponds to a 7 column 8 digit, double precision format. Compact output corresponds to a 12 column, 5 digit single precision format.

CALL TWOMAT(A,N,LEN,CAR,TEXT,NCHAR,NAMES)

Argument Definitions

A(N*N+1)/2 Vector stored upper triangular matrix (DP)
N Dimension of A
LEN Column format (7 or 12 columns). When LEN is different from 7 or 12 the print defaults to 12 columns.
CAR(N) Parameter names (alphanumeric) associated with A. When NAMES is false, CAR is not used.
TEXT(NCHAR) An array of field data characters to be printed as a title preceding the matrix
NCHAR Number of characters (including spaces) that are to be printed in text(). ABS(NCHAR) LE.114. If NCHAR is negative there is no page eject before printing. NCHAR positive results in a page eject so that the print starts on a fresh page.
NAMES A logical flag. If true then the names of the parameters are used as labels for the rows and columns. If false the output labels default to numerical values.

Remarks and Restrictions

Using NCHAR nonnegative, and starting the print at the top of a new page makes it easier to locate the printed result and is
especially recommended when dealing with large dimensioned arrays. Page economy can, however, be achieved using the NCHAR negative option. In this case the print begins on the next line. The alphanumerics in this routine make it machine dependent; it is arranged for implementation on a UNIVAC 1108.
24. TZERO (Triangular matrix zero)

Purpose
To zero out rows IS(Istart) to IF(Ifinal) of the vector stored upper triangular matrix R.

CALL TZERO(R,N,IS,IF)

Argument Definition

\[ R(\frac{N(N+1)}{2}) \quad \text{Input vector stored upper triangular matrix} \]

\[ N \quad \text{Row dimension of vector stored matrix} \]

\[ IS \quad \text{First row of } R \text{ that is to be set to zero} \]

\[ IF \quad \text{Last row of } R \text{ that is to be set to zero} \]

Functional Description

\[ R(\text{input}) \quad \rightarrow \quad R(\text{output}) \]

\[ \text{IS} \quad \text{0} \quad \text{IF} \]
25. **UDCOL** (U-D covariance factor colored noise time update)

**Purpose**

To time update the U-D covariance factors so as to include the effects of colored noise variables.

```
CALL UDCOL(U,N,KS,NCOLOR,V,EM,Q)
```

**Argument Definitions**

- **U(N*(N+1)/2)**: Input vector stored U-D covariance factors. The updated result resides here on output.
- **N**: Filter matrix dimension. If the last column of U houses the filter estimates, then \( N = \text{number filter variables} + 1 \).
- **KS**: Location of the first colored noise variable (\( KS \geq 1 \) AND \( KS \leq N \))
- **NCOLOR**: The number of colored noise variables contiguous to the first, including the first (\( NCOLOR \geq 1 \))
- **V(KS-1+NCOLOR)**: Work vector (\( (KS-1+NCOLOR) \leq N \))
- **EM(NCOLOR)**: Input vector of colored noise mapping terms (unaltered by program)
- **Q(NCOLOR)**: Input vector of process noise variances (unaltered by program)

**Remarks and Restrictions**

When estimates are involved they are appended as an additional column to the U-D matrix. When the subroutine is applied to the augmented matrix the estimates are correctly updated. When the colored noise terms are not contiguously located one can fill in the gaps with unit EM terms and corresponding zero Q elements. It is preferable, however, to apply the subroutine repeatedly to the individual contiguous groups.
Functional Description

The model equation corresponding to the time update of this subroutine is

$$
\begin{bmatrix}
  x \\
  p_j \\
  y_j
\end{bmatrix}
= \begin{bmatrix}
  I & 0 & 0 \\
  0 & M & 0 \\
  0 & 0 & I
\end{bmatrix}
\begin{bmatrix}
  x \\
  p_j + I \\
  y_j
\end{bmatrix}
$$

where $M$ is diagonal, with NP terms, and $w_j \sim N(0,Q)$ where $Q$ is diagonal with NP terms. The output $U-D$ array associated with this time update equation satisfies

$$
UDU^T_{\text{(output)}} = \psi UDU^T_{\psi} + QBQ^T
$$

where $\psi$ and $B$ are as above. The algorithm for obtaining $U-D$ (output) is the Bierman-Thornton one-component-at-a-time update described in Bierman - Factorization Methods for Discrete Sequential Estimation", Academic Press (1977), pp 147-148.
26. UDMEAS (U-D measurement update)

**Purpose**

Kalman filter measurement updating using Bierman's U-D measurement update algorithm, c: 1975 CONF. DEC. CONTROL paper. A scalar measurement \( z = A^T x + v \) is processed, the covariance U-D factors and estimate (when included) are updated, and the Kalman gain and innovations variance are computed.

```
CALL UDMEAS(U,N,R,A,F,G,ALPHA)
```

**Argument Definitions**

**INPUTS**

- **U**(N*(N+1)/2): Upper triangular vector stored input matrix. D elements are stored on the diagonal. The U vector corresponds to an a priori covariance. If state estimates are involved the last column of U contains X. In this case \( \text{Dim } U = (N+1)*(N+2)/2 \) and on output \( U(N+1)*(N+2)/2 = z-A^T*X(a \text{ priori est}) \).
- **N**: Dimension of state vector, \( N \geq 2 \)
- **R**: Measurement variance
- **A**(N): Vector of Measurement coefficients; if data then \( A(N+1) = z \)
- **F**(N): Input work vector. To economize on storage \( F \) can overwrite \( A \)
- **ALPHA**: If ALPHA.LT.zero no estimates are computed (and X and z need not be included).

**OUTPUTS**

- **U**: Updated vector stored U-D factors. When ALPHA (input) is nonnegative the \( (N+1) \)st column contains the updated estimate and the predicted residual.
- **ALPHA**: Innovations variance of the measurement residual.
- **F**: Contains \( U^*T*A(input) \) and when ALPHA(input) is nonnegative \( F(N+1) = (z-A^*T*X(a \text{ priori est}))/\text{ALPHA}. \)

75
G(N) \quad \text{Vector of unweighted Kalman gains,}\n\quad K = G/\text{ALPHA}

**Remarks and Restrictions**

One can use this algorithm with R negative to delete a
previously processed data point. One should, however, note that
data deletion is numerically unstable and sometimes introduces
numerical errors.

The algorithms holds for R = 0 (a perfect measurement) and
the code has been arranged to include this case. Such situations
arise when there are linear constraints and in the generation of
certain error "budgets".

**Functional Description**

The algorithm updates the columns of the U-D matrix, from
left to right, using Bierman's algorithm, see Bierman's
"Factorization Methods for Discrete Sequential Estimation,"
27. **UD2COV**  *(U-D factor to covariance)*

**Purpose**

To obtain a covariance from its U-D factorization. Both matrices are vector stored and the output covariance can overwrite the input U-D array. U-D and P are related via $P = UDU^T$.

```
CALL UD2COV(UIN, POUT, N)
```

**Argument Definitions**

- **UIN**($N*(N+1)/2$)  
  Input vector stored U-D factors, with $D$ entries stored on the diagonal.

- **POUT**($N*(N+1)/2$)  
  Output vector stored covariance matrix (POUT = UIN is permitted).

- **N**  
  Dimension of the matrices involved (N.GE.2)
28. UD2SIG  (U-D factors to sigmas)

Purpose

To compute variances from the U-D factors of a matrix.

CALL UD2SIG(U,N,SIG,TEXT,NCT)

Argument Definitions

U(N*(N+1)/2)  Input vector stored array containing the U-D factors. The D (diagonal) elements are stored on the diagonal of U.

N         Dimension of the U matrix (N.GE.2)

SIG(N)  Output vector of standard deviations

TEXT ()  Output label of field data characters, which precedes the printed vector of standard deviations.

NCT      Number of characters of text, 0.LE.NCT.LE.126. If NCT = 0, no sigmas are printed, i.e. nothing is printed.

Remarks and Restrictions

The user is cautioned that the text related portion of this subroutine may not be compatible with other computers. The changes that may be involved are, however, very modest.

Functional Description

If U and V are represented as doubly subscripted matrices then

\[ \text{SIG}(J) = \left( \text{D}(J,J) + \sum_{K=J+1}^{N} \text{D}(K,K)[\text{U}(J,K)]^2 \right)^{1/2} \]

If NCT GT 0 a title is printed, followed by the sigmas.
29. UTINV (Upper triangular matrix inverse)

Purpose

To invert an upper triangular vector stored matrix and store the result in vector form. The algorithm is so arranged that the result can overwrite the input.

```
CALL UTINV(RIN,N,ROUT)
```

Argument Definitions

- **RIN(N*(N+1)/2)**: Input vector stored upper triangular matrix
- **N**: Matrix dimension
- **ROUT(N*(N+1)/2)**: Output vector stored upper triangular matrix inverse (ROUT = RIN is permitted)

Remarks and Restrictions

Ill conditioning is not tested, but for nonsingular systems the result is as accurate as is the full rank Euclidean scaled singular value decomposition inverse. Singularity occurs if a diagonal is zero. The subroutine terminates when it reaches a zero diagonal. The columns to the left of the zero diagonal are, however, inverted and the result stored in ROUT.

This routine can also be used to produce the solution to \( RX = Z \). Place \( Z \) in column \( N+1 \) (viz. \( RIN(N*(N+1)/2+1) = Z(1) \), etc.), define \( RIN((N+1)(N+2)/2) = -1 \) and call the subroutine using \( N+1 \) instead of \( N \). On return the first \( N \) entries of column \( N+1 \) contain the solution (e.g. \( ROUT(N*(N+1)/2+1) = X(1) \), etc.). When only the estimate is needed, then it is more efficient to use the code described in section to II.8 to obtain \( X \), directly.
Because matrix inversion is numerically sensitive we recommend using this subroutine only in double precision.

**Functional Description**

The matrix inversion is accomplished using the standard back substitution method for inverting triangular matrices, cf. the book references by Lawson and Hanson, [1] or Bierman [3].
30. UTIROW  (Upper triangular inverse, inverting only the upper rows)

Purpose

To compute the inverse of a vector stored upper triangular matrix, when the lower right corner triangular inverse is given.

CALL UTIROW(RIN,N,ROUT,NRY)

Argument Definitions

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIN(N*(N+1)/2)</td>
<td>Input vector stored upper triangular matrix. Only the first N - NRY rows are altered by the algorithm.</td>
</tr>
<tr>
<td>N</td>
<td>Matrix dimension.</td>
</tr>
<tr>
<td>ROUT(N*(N+1)/2)</td>
<td>Output vector stored upper triangular matrix inverse. On input the lower NRY dimensional right corner contains the given (known) inverse. This lower right corner matrix is left unchanged. (ROUT = RIN is permitted.)</td>
</tr>
<tr>
<td>NRY</td>
<td>Number of rows, starting at the bottom, that are assumed already inverted.</td>
</tr>
</tbody>
</table>

Remarks and Restrictions

The purpose of this subroutine is to complete the computation of an upper triangular matrix inverse, given that the lower right corner has already been inverted. Part of the input, the rows to be inverted, are inserted via the matrix RIN. The portion of the matrix that has already been inverted is entered via the matrix ROUT. It may seem odd that part of the input matrix is put into RIN and part into ROUT. The reasoning behind this decision is that RIN represents the input matrix to be inverted (it just happens that we do not make use of the lower right triangular entries); ROUT represents the inversion result, and therefore that portion of the inversion that is given should be entered in this array.
Ill conditioning is not tested, but for nonsingular systems the result is accurate. Singularity halts the algorithm if any of the first N-NRY diagonal elements is zero. If the first zero encountered moving up the diagonal (starting at N-NRY) is at diagonal j then the rows below this element will be correctly represented in ROUT.

To generate estimates do the following: put N+1 into the matrix dimension argument; in the first N-NRY rows of the last column of RIN put the right hand side elements of the equation \( R_x + R_{xy} y = z_x \) (i.e., \( R_x \), \( R_{xy} \), and \( z_x \) make up the first N-NRY rows of RIN); in the next NRY entries of ROUT, beginning in the (N-NRY+1)st element, put \( y_{est} \) (i.e., \( R_y^{-1} \) and \( y_{est} \) make up rows N-NRY+1,...,N of ROUT); and ROUT((N+1)(N+2)/2) = -1. On output, the last column of ROUT will contain \( x_{est} \), \( y_{est} \) and -1.

When NRY = 0 this algorithm is equivalent to subroutine UTINV.

**Functional Description**

The matrix inversion is accomplished using the standard back substitution method. The computations are arranged row-wise, starting at the bottom (from row N-NRY, since it is assumed that the last NRY rows have already been inverted).
31. WGS (Weighted Gram-Schmidt matrix triangularization)

Purpose

To compute a vector stored U-D array from an input rectangular matrix $W$, and a diagonal matrix $D_w$ so that $W D_w^T = U D U^T$.

CALL WGS($W$, IMAXW, IW, JW, DW, U, V)

Argument Definitions

$W(IW, JW)$  
Input rectangular matrix, destroyed by the computations

IMAXW  
Row dimension of input $W$ matrix, IMAXW.GE.IW

IW  
Number of rows of $W$ matrix, dimension of $U$

JW  
Number of columns of $W$ matrix

DW(JW)  
Diagonal input matrix; the entries are assumed to be nonnegative. This vector is unaltered by the computations

$U(IW*(IW+1)/2)$  
Vector stored output U-D array

$V(JW)$  
Work vector in the computation

Remarks and Restrictions

The algorithm is not numerically stable when negative $D_w$ weights are used; negative weights are, however, allowed. If $JW$ is less than $IW$ (more rows than columns), the output U-D array is singular; with $IW-JW$ zero diagonal entries in the output $U$ array.

Functional Description

A $D_w$-orthogonal set of row vectors, $\phi_1, \phi_2, \ldots, \phi_{IW}$, are constructed from the input rows of the $W$ matrix, i.e., $W = U \phi$, $\phi_d w^T = D$.

The construction is accomplished using the modified Gram-Schmidt orthogonal construction (see refs. [1] or [3]). This algorithm is reputed to have excellent numerical properties. Note that the $\phi$ vectors are not of interest in this routine, and they are overwritten.

The $V$ vector used in the program houses vector $IW-j+1$ of $\phi$ at step $j$ of the algorithm. The fact that the computed $\phi$ vectors may not be $D$ orthogonal is of no import in regard to the $U$ and $D$ computed results.
References


V. FORTRAN Subroutine Listings

The subroutines use only FORTRAN IV, and are therefore essentially portable. The one notable exception is subroutine TWOMAT, which prints triangular, vector stored matrices. It employs FORTRAN V FORMAT statements and six character UNIVAC alphanumeric wordlength, and thus is UNIVAC dependent. Subroutine UD2SIG also involves text, and it too is therefore to some extent machine dependent. Comment statements appear occasionally to the right of the FORTRAN code, and are preceded by a "@" symbol. The subroutine user can, if necessary, transfer or remove such program commentary.

All of the subroutines employ "implicit double precision" statements. They are, however, constructed so as to operate in single precision, and the user has only to omit or comment out the implicit statements. If the subroutines are to be used in double precision on a machine that does not have the implicit FORTRAN option one should explicitly declare all of the non-integer variable names appearing in the programs as double precision variables.

If these subroutines are to be used in production code and computational efficiency is of major concern one should replace the somewhat lengthy subroutine argument lists by introducing COMMON, and including those terms in the COMMON that are redundantly computed with each subroutine call.
SUBROUTINE A2A1 (A*IA,IR,LA,NAMA,1,IA1,LA1,NAMA1) C

SUBROUTINE TO REARRANGE THE COLUMNS OF A(IR,LA), IN NAMA ORDER AND PUT THE RESULT IN A1(IR,LA1) IN NAMA1 ORDER. ZERO COLUMNS ARE INSERTED IN A1 CORRESPONDING TO THE NEWLY DEFINED NAMES.

A(IR,LA) INPUT RECTANGULAR MATRIX
IA ROW DIMENSION OF A, IR, LE, IA
IR NO. OF ROWS OF A THAT ARE TO BE REARRANGED
LA NO. COLUMNS IN A, ALSO THE NO. OF PARAMETER NAMES ASSOCIATED WITH A
NAMA(LA) PARAMETER NAMES ASSOCIATED WITH A
A1(IR,LA1) OUTPUT RECTANGULAR MATRIX
A AND A1 CANNOT SHARE COMMON STORAGE
IA1 ROW DIMENSION OF A1, IR, LE, IA1
LA1 NO. COLUMNS IN A1, ALSO THE NO. OF PARAMETER NAMES ASSOCIATED WITH A1
NAMA1(LA1) INPUT LIST OF PARAMETER NAMES TO BE ASSOCIATED WITH THE OUTPUT MATRIX A1

COGNIZANT PERSONS: G. J. BIERMAN/M. W. NAFA (JPL, SEP. 1976)

DIMENSION A(IA1), NAMA(1), A1(IA1,1), NAMA1(1)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)

ZERO=0.
DO 100 J=1,LA1
DO 60 I=1,LA
   IF (NAMA(I),EQ,NAMA1(J)) GO TO 80
60 CONTINUE
DO 70 K=1,IR
70 A1(K,J)=ZERO \& ZERO COL. CORRESP. TO NEW NAME
   GO TO 100
80 DO 90 K=1,IR
90 A1(K,J)=A(K,I) \& COPY COL. ASSOC. WITH OLD NAMF
100 CONTINUE

RETURN
END
SUBROUTINE COMBO (R,L1,NAM1,L2,NAM2,A,IA,LA,NAM,)  

TO REARRANGE A VECTOR STORED TRIANGULAR MATRIX AND STORE 

THE RESULT IN MATRIX A. THE DIFFERENCE BETWEEN THIS SUB- 

ROUTINE AND R2A IS THAT THERE THE NAMELIST FOR A IS INPUT. 

HERE IT IS DETERMINED BY COMBINING THE LIST FOR R WITH 

A LIST OF DESIRF NAMES.

R(L1*(L1+1)/2) INPUT VECTOR STORED UPPER TRIANGULAR MATRIX 

L1 NO. OF PARAMETERS IN R (AND IN NAM1)  

NAM1(L1) NAMES ASSOCIATED WITH R 

L2 NO. OF PARAMETERS IN NAM2 

NAM2(L2) PARAMETER NAMES THAT ARE TO BE COMBINED WITH R 

(NAM1 LIST). THESE NAMES MAY OR MAY NOT BE IN 

NAM1. 

A(L1*LA) OUTPUT ARRAY CONTAINING THE REARRANGED 

R MATRICES, L1,LE,TA. 

IA ROW DIMENSION OF A 

LA NO. OF PARAMETER NAMES IN NAM2, AND THE 

COLUMN DIMENSION OF A, LA=L1+L2=NO. NAMES 

COMMON TO NAM1 AND NAM2. LA IS COMPUTED AND 

OUTPUT. 

NAM(LA) PARAMETER NAMES ASSOCIATED WITH THE OUTPUT A 

MATRIX. CONSISTS OF NAMES IN NAM1 WHICH ARE 

NOT IN NAM2 FOLLOWED BY NAM2. 

COGNIZANT PERSONS: G.J.BIERMAN/M.W.NFAN (JPL/SPF, 1976) 

IMPLICIT DOUBLE PRECISION (A-H,O-7) 

DIMENSION R(1), A(IA*1), NAM1(1), NAM2(1), NAM(1)

C

ZERO=0.0  
K=1  
DO 100 I=1,L1  
   DO 50 J=1,L2  
      IF (NAM1(I)*EQ 'NAM2(J)') GO TO 100  
   CONTINUE  
   NAM(K)=NAM1(I)  
   J=I*(I-1)/2  
   DO 60 L=1,L1  
      A(L*K)=R(JJ+L)  
      IF (L*EQ.L1) GO TO 100  
      IP1 = I+1  
      DO 70 L=IP1,L1  
      A(L*K) = ZERO  
   60 CONTINUE  
   K=K+1  
100 CONTINUE  

C NAM(NAM) UNIQUE TO NAM1 ARE NOW IN NAM  

DO 200 J=1,L2  
   DO 150 I=1,L1  
      IF (NAM2(J)*EQ 'NAM1(I)') GO TO 170  
   CONTINUE  
   NAM(K)=NAM2(J)  
   DO 160 L=1,L1  
   A(L*K)=ZERO  
200 CONTINUE  

C
C NAMES UNIQUE TO NAM2 ARE NOW IN NAMA
GO TO 190
170 NAMA(K)=NAM2(J)
C LOCATE DIAGONAL OF PRECEDING COLUMN
    JJ=I*(I-1)/2
    DO 180 L=1,I
    180 A(L+K)=R(JJ+L)
    IF (I.EQ.L1) GO TO 190
    IP1=I+1
    DO 185 L=IP1,L1
    185 A(L+K)=ZERO
    190 K=K+1
    200 CONTINUE
    IA=K-1
C NAMES MUTUAL TO NAM1 AND NAM2 ARE NOW IN NAMA
RETURN
END
SUBROUTINE COVRHO(COV,N,RHO,V)
C TO COMPUTE THE CORRELATION MATRIX RHO, FROM AN INPUT COVARIANCE
C MATRIX COV. BOTH MATRICES ARE UPPER TRIANGULAR VECTOR STORED.
C THE CORRELATION MATRIX RESULT CAN OVERWRITE THE INPUT COVARIANCE
C THE CORRELATION MATRIX RESULT CAN OVERWRITE THE INPUT COVARIANCE
C INPUT VECTOR STORED POSITIVE SEMI-DEFINITE
C COV(N*(N+1)/2) INPUT VECTOR STORED POSITIVE SEMI-DEFINITE
C N NUMBER OF PARAMETERS, N.GE.1
C RHO(N(N+1)/2) OUTPUT VECTOR STORED CORRELATION MATRIX.
C RHO(IJ)=COV(IJ)/(SIGMA(I)*SIGMA(J))
C V(N) WORK VECTOR

C COGNIZANT PERSONS: G.J.BIERMAN/M.W.HEAD (JPL,FEB,1978)
C
C IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C DIMENSION COV(1), RHO(1), V(1)
C
C ONE=1.00
C Z=0.00

C JJ=0
DO 10 J=1,N
  JJ=JJ+J
  V(J)=Z
  IF (COV(JJ).GT.Z) V(J)=ONE/SRT(COV(JJ))
10 CONTINUE

C **** SOME MACHINES REQUIRE DSQRFT FOR DOUBLE PRECISION

C 10 CONTINUE
C
C IJ=0
DO 20 J=1,N
  S=V(J)
  DO 20 I=1,J
    IJ=IJ+1
20   RHO(IJ)=COV(IJ)*S*V(I) RETURN
END
SUBROUTINE COV2R(U,N)

TO CONSTRUCT THE UPPER TRIANGULAR CHOLESKY FACTOR OF A
POSITIVE SEMI-DEFINITE MATRIX. BOTH THE INPUT COVARIANCE
AND THE OUTPUT CHOLESKY FACTOR (SQUARE ROOT) ARE VECTOR
STORED. THE OUTPUT OVERWRITES THE INPUT.

COVARIANCE(INPUT)=U*U**T (U IS OUTPUT).

IF THE INPUT COVARIANCE IS SINGULAR THE OUTPUT FACTOR HAS
ZERO COLUMNS.

U(N*(N+1)/2) CONTAINS THE INPUT VECTOR STORED COVARIANCE
MATRIX (ASSUMED POSITIVE DEFINITE) AND ON OUTPUT
IT CONTAINS THE UPPER TRIANGULAR SQUARE ROOT
FACTOR.

N DIMENSION OF THE MATRICES INVOLVED

COGNIZANT PERSONS: G.J.BIERMAN/W.M.NEAD (JPL, FFM, 1977)

IMPLICIT DOUBLE PRECISION (A-H.O-Z)

DIMENSION U(1)

ZERO=0.0
ONE=1.
JJ=N*(N+1)/2

DO 5 J=1,N
  IF (U(JJ)**T*ZERO) U(JJ)=ZERO
  ELSE U(JJ)=SQRT(U(JJ))
  IF (U(JJ)*GT*ZERO) ALPHA=ONF/U(JJ)
  KK=0
  JJN=JJ-J
  JM1=J-1
  DO 3 K=1,JM1
     U(JJN+K)=ALPHA*U(JJN+K)
     S=U(JJN+K)
     DO 1 I=1,K
        U(KK+I)=U(KK+I)-S*I(I+J)
     3  KK=KK+K
     4  JJ=JJ+1
  5  IF (U(1)**T*ZERO) U(1)=ZERO
      U(1)= SQRT(U(1))

RETURN
END
SUBROUTINE COV2(J=N)
  TO OBTAIN THE U-D FACTORS OF A POSITIVE SEMI-DEFINITE MATRIX.
  THE INPUT VECTOR STORED MATRIX IS OVERWRITTEN BY THE OUTPUT
  U-D FACTORS WHICH ARE ALSO VECTOR STORED.
  U(N(N+1)/2) CONTAINS INPUT VECTOR STORED COVARIANCE MATRIX.
  ON OUTPUT IT CONTAINS THE VECTOR STORED U-D
  COVARIANCE FACTORS.
  N MATRIX DIMENSION: N\geq 2
  SINGULAR INPUT COVARIANCES RESULT IN OUTPUT MATRICES WITH ZERO
  COLUMNS
  COGNIZANT PERSONS: G.J. RIERMAN/R.A. JACORSON (JPL, FEB. 1977)

IMPLICIT DOUBLE PRECISION (A-H,O-Z)

DIMENSION U(1)

Z=0.0D0
ONE=1.0D0
NONE=1

JJ=N*(N+1)/2
NP2=N+2
DO 50 L=2,N
  J=NP2-L
  ALPHA=Z
  IF (U(JJ).GE.Z) GO TO 10
  WRITE (6,100) J+U(JJ)
  U(JJ)=Z

10  IF (U(JJ).GT.Z) ALPHA=ONE/U(JJ)
  JJ=JJ-J
  KK=0
  KJ=JJ
  JM1=J-1
  DO 40 K=1,JM1
    KJ=KJ+1
    BETA=U(KJ)
    U(KJ)=ALPHA*U(KJ)
    IH=JJ
    IK=KK
    DO 30 I=1,K
      IK=IK+1
      J=IJ+1
      U(IK)=U(IK)-BETA*U(IJ)
    30    KK=KK+K
    CONTINUE
    IF (U(1).GE.Z) GO TO 60
    WRITE (6,100) NONE, U(1)
    U(1)=Z
  60  RETURN

C
100 FORMAT (140,20X,4 AT STEP, 14,4 DIAGONAL ENTRY = *F12.4)
END
SUBROUTINE C2C (C, IC, L1, NAM1, L2, NAM2)

SUBROUTINE TO REARRANGE THE ROWS AND COLUMNS OF MATRIX C(L1+1:L1+L2) IN NAM1 ORDER AND PUT THE RESULT IN NAM2 ORDER. ZERO COLUMNS AND ROWS ARE ASSOCIATED WITH OUTPUT DEFINED NAMES THAT ARE NOT CONTAINED IN NAM1.

C(L1+1:L1) INPUT MATRIX
IC ROW DIMENSION OF C, IC+GF.L=MAX(L1+L2)
L1 NO. OF PARAMETER NAMES ASSOCIATED WITH C ON INPUT. (ONLY THE FIRST L1 ENTRIES APPLY TO THE INPUT C)
NAM1(L) PARAMETER NAMES ASSOCIATED WITH C ON INPUT. (ONLY ASSOCIATED WITH OUTPUT C)
L2 NO. OF PARAMETER NAMES ASSOCIATED WITH THE OUTPUT C
NAM2(L2) PARAMETER NAMES ASSOCIATED WITH THE OUTPUT C

Cognizant Persons: G.J. Hufnagel/M.W. Head (JPL, Sept. 1976)

IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION C(IC+1), NAM1(L), NAM2(L)

ZERO=0.
L=MAX(L1,L2)
IF (L.LT.L1) GO TO 5
NM=L1+1
DO 1 K=NM+1
1 NAM1(K)= ZERO
5 DO 10 J=1:L2
10 IF (NAM1(I).EQ.NAM2(J)) GO TO 30
10 CONTINUE
GO TO 90
30 IF (I.EQ.J) GO TO 90
40 CONTINUE
DO 80 K=1:L1
40 H=C(K,J) C(K,J)=C(K,I)
50 C(K,I)=H
DO 80 K=1:L2
80 H=C(J,K) C(J,K)=C(I,K)
80 CONTINUE
DO 90 K=1:L2
90 NM=NAM1(K) NAM2(K)=NAM1(J)
90 CONTINUE

Find NAM2 NAMES NOT IN NAM1 AND SET CORRESPONDING ROWS AND COLUMNS TO ZERO

DO 120 J=1:L2
120 IF (NAM1(I).EQ.NAM2(J)) GO TO 120
120 CONTINUE
DO 110 K=1:L2
110 C(J,K)=ZERO

110     C(K-J)=ZERO
120     CONTINUE
C
    RETURN
END
SUBROUTINE WHPLOT(S,W,MROW,NROW,NCOL,V)

TRIANGULARIZES RECTANGULAR W BY POST MULTIPLYING IT BY AN
ORTHOGONAL TRANSFORMATION T. THE RESULT IS IN S

S(NROW*(NROW+1)/2) OUTPUT UPPER TRIANGULAR VECTOR STORED SORT
COVARIANCE MATRIX
W(NROW,NCOL) INPUT RECTANGULAR SORT COVARIANCE MATRIX
(W IS DESTROYED BY COMPUTATIONS)
MROW ROW DIMENSION OF W
NROW NUMBER OF ROWS OF W TO BE TRIANGULARIZED
AND THE DIMENSION OF S (NROW,GT,1)
NCOL NUMBER OF COLUMNS OF W (NCOL,GE,NROW)
V(NCOL) WORK VECTOR

COGNIZANT PERSONS: G.J.RIFRAN M.W. NEAD (JPL NOV, 1977)

IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DOUBLE PRECISION SUM,BETA
DIMENSION S(1),W(MROW,NCOL),V(NCOL)

ZERO=0.00
ONE=1.0D0

JCOL=NCOL
NSYM=NRW*(NRW+1)/2
JC=NRW+2
DO 150 L=2,NROW
IRW=JCOL-L
SUM=ZERO
DO 1 K=1,JC
V(K)=W(IRW,K)
SUM=SUM+V(K)**2
100 SUM=DSQRT(SUM)
IF (V(JCOL)*GT,ZERO) SUM=SUM .Q DIAGONAL ENTRY (JCOL,JCOL)

C S(NSYM)=SUM
NSYM=NSYM-IROW
V(JCOL)=V(JCOL)-SUM
IF (SUM,NE,ZERO) BETA=ONE/((SUM*V(JCOL))
T=ORTHOG, TRANS,)=I-BETA*V*V**T
IRW=IRW-I
JCOL=JCOL-1
DO 140 I=IRW+1,IRW
SUM=ZERO
DO 110 K=1,JCOL
SUM=SUM+V(K)*W(I,K)
110 SUM=BETA*SUM
DO 120 K=1,JCOLM1
120 W(I,K)=W(I,K)-SUM*V(K)
140 S(NSYM+1)=W(I,IRW)+SUM*V(IRW)
150 JCOL=JCOLM1

JC=NCOL-NROW+1
SUM=ZERO
DO 160 J=1,JC
160 SUM=SUM+W(I,J)**2
S(1)=DSQRT(SUM)
C
RETURN
END
SUBROUTINE INF2R (R,N)

TO CHOLFSKY FACTOR AN INFORMATION MATRIX

COMPUTES A LOWER TRIANGULAR VECTOR STORED CHOLFSKY FACTORIZATION
OF A POSITIVE SEMI-DEFINITE MATRIX. R= R(T*TR), R UPPER TRIANGULAR.
Both matrices are vector stored and the result overwrites the input

R(N*(N+1)/2) ON INPUT THIS IS A POSITIVE SEMI-DEFINITE
(INFORMATION) MATRIX, AND ON OUTPUT IT IS THE
TRANSPOSED LOWER TRIANGULAR CHOLFSKY FACTOR. IF THE
INPUT MATRIX IS SINGULAR THE OUTPUT MATRIX WILL
HAVE ZERO DIAGONAL ELEMENTS.

N DIMENSION OF MATRICES INVOLVED, N.GE.2

Cognizant person: G. J. Riemann/J.W. Nif~O (JPL, Feb, 1977)

IMPLICIT DOUBLE PRECISION (A-H,O-Z)

DIMENSION R(1)

Z=0.D0
ONE=1.D0
JJ=0
NN=N*(N+1)/2
NM=1

DO 10 J=1,NM
JJ=JJ+J
IF (R(JJ).GE.2) GO TO 5
WRITE (6,20) J,R(JJ)
R(JJ)=Z
5 R(JJ)=SQRT(R(JJ))

ALPHA=Z
IF (R(JJ).GT.2) ALPHA=ONE/R(JJ)
JK=NN+J
JP=J+1
JIS=JK
NPJP=JP+1

DO 10 L=JP1,N
K=NPJP-L
JK=JK+K
R(JK)=ALPHA*R(JK)
RETA=R(JK)
KI=NN+K
JI=JIS
NPK=K+1
DO 10 M=NPK,N
I=NPK-M
KI=KI+I
JI=JI+I
10 CONTINUE

97
10 \text{R(KI)} = \text{R(KI)} - \text{R(JI)} \times \text{BETA}

\text{C}
\text{IF} (\text{R(NN)} \geq \text{Z}) \text{GO TO 15}
\text{WRITE (6,20) N,R(NN)}
\text{R(NN)} = \text{Z}
\text{15 R(NN) = SQRT(R(NN))}
\text{RETURN}

\text{C}
\text{20 FORMAT (1H0,20X,14,4,DIAGONAL ENTRY =**E12.4,} \text{1**, IT IS RESET TO ZERO")}
\text{END}
SUBROUTINE PERMIT (A, IA, IR, L1, NAM1, L2, NAM2)

SUBROUTINE TO REARRANGE PARAMETERS OF A(1R, L1), NAM1 ORDER TO A(1R, L2), NAM2 ORDER. ZERO COLUMNS ARE INSERTED CORRESPONDING TO THE NEWLY DEFINED NAMES.

A(1R, L) INPUT RECTANGULAR MATRIX, L = MAX(L1, L2)
IA ROW DIMENSION OF A, IA+GF+IR
IR NUMBER OF ROWS OF A THAT ARE TO BE REARRANGED
L1 NUMBER OF PARAMETER NAMES ASSOCIATED WITH THE INPUT MATRIX
NAM1(L) PARAMETER NAMES ASSOCIATED WITH A ON INPUT (ONLY THE FIRST L1 ENTRIES APPLY TO THE INPUT A) NAM1 IS DESTROYED BY PERMIT
L2 NUMBER OF PARAMETER NAMES ASSOCIATED WITH THE OUTPUT MATRIX
NAM2 PARAMETER NAMES ASSOCIATED WITH THE OUTPUT A

COGNIZANT PERSONS: G. J. TRIFMAN/M. W. NFAD (JPL, SEPT. 1976)

IMPLICIT DOUBLE PRECISION (A-H, O-Z)
DIMENSION A(IA, 1), NAM1(1), NAM2(1)

ZERO = 0, L = MAX(L1, L2)
IF (L.LE.L1) GO TO 50
NM = L1 + 1
NO 40 K = NM + L
40 DO 100 J = 1, L2
   DO 60 I = 1, L1
      IF (NAM1(I).EQ.NAM2(J)) GO TO 65
   60 CONTINUE
   GO TO 100
65 CONTINUE
   IF (I.EQ.J) GO TO 100
   DO 70 K = 1, IR
      Q INTERCHANGE Cols I AND J
      W = A(K, J)
      A(K, J) = A(K, I)
      A(K, I) = W
      NM = NAM1(I)
      NAM1(I) = NAM1(J)
      NAM1(J) = NM
   70 CONTINUE
   REPEAT TO FILL NEW Cols
   DO 200 J = 1, L2
      DO 160 I = 1, L1
         IF (NAM1(I).EQ.NAM2(J)) GO TO 200
      160 CONTINUE
      DO 170 K = 1, IR
      170 A(K, J) = ZERO
   200 CONTINUE
RETURN
END
SUBROUTINE PHIU(PHI,MAXPHI,IRPHI,ICPHI,U,N,PHIU,MPHIU)

C THIS SUBROUTINE COMPUTES W=PHIU WHERE PHI IS A RECTANGULAR MATRIX PHIU=U*PHI
WITH IMPLICITLY DEFINED COLUMNS OF TRAILING ZEROS AND U IS A VECTOR STORED UPPER TRIANGULAR MATRIX

C PHI(IRPHI,ICPHI) INPUT RECTANGULAR MATRIX PHI, ROW DIMENSION OF PHI IRPHI.LE.MAXPHI
C MAXPHI NO. ROWS OF PHI
C IRPHI NO. COLS OF PHI
C U(N+(N+1)/2) UPPER TRIANGULAR VECTOR STORED MATRIX N DIMENSION OF U MATRIX (ICPHI.LE.N)
C PHIU(IRPHI,N) OUTPUT RESULT OF PHI*U, PHIU CAN OVERWRITE PHI
C MPHIU ROW DIMENSION OF PHIU

C COGNIZANT PERSONS: G.J.BIEPMAN/M.W.NEAL (JPL, FFR.197A)

IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION PHI(MAXPHI,IRPHI),U(1)*ICPHI,MPHIU(1)
DOUBLE PRECISION SUM

C DO 10 I=1,IRPHI
10 PHIU(I,1)=PHI(I,1)

C NP2=N+2
KJS=N*(N+1)/2
DO 40 L=2,N
J=NP2-L
KJS=KJS-J
JM1=J-1
DO 30 J=1,IRPHI
SUM=PHIU(J,J)
IF (J.LE.ICPHI) 60 TO 15
SUM=0.D0
30 CONTINUE

C 15 DO 20 K=1,JM1
20 SUM=SUM+PHIU(K,K)*U(KJS+K)
30 PHIU(I,J)=SUM
40 CONTINUE

RETURN
END
SUBROUTINE RA (R,N,MAXA,IA,J,RA,M,A,NRA)

C TO COMPUTE RA=R*A

WHERE R IS UPPER TRIANGULAR VECTOR SUBSCRIPTED AND OF DIMENSION N, A HAS JA COLUMNS AND IA ROWS, IF IA.LT.JA THEN THE BOTTOM JA-IA ROWS OF A ARE ASSUMED TO BE IMPLICITLY DEFINED AS THE

BOTTOM JA-IA ROWS OF THE JA DIMENSION IDENTITY MATRIX, ONLY NRA ROWS OF THE PRODUCT R*A ARE COMPUTED.

C

R=UPPER TRIANGULAR VECTOR STORED INPUT MATRIX C N DIMENSION OF R C A(IA,J) INPUT RECTANGULAR MATRIX C MAXA ROW DIMENSION OF A C IA NUMBER OF ROWS IN THE A MATRIX (IA.LT.MAXA) C JA NUMBER OF COLUMNS IN THE A MATRIX C RA(NRA,J) OUTPUT RESULTING RECTANGULAR MATRIX C R=A IS ALLOWED C MAXRA ROW DIMENSION OF PA C NRA NUMBER OF ROWS OF THE PRODUCT R*A THAT ARE COMPUTED (NRA.LT.MAXRA)

C COGNIZANT PERSONS: G.J.BERMAN/M.W.NEAD (JPL, FER, 1978)

C IMPLICIT DOUBLE PRECISION (A-H,P-O-Z)

C DIMENSION R(1),A(MAXA,1),RA(MAXRA,1)

C DOUBLE PRECISION SUM

C

IJ=IJ+(IA+I)/2 @ IJ=JJ(I)

C DO 30 J=1,JA

II=0 @ TO BE REMOVED IF JJ(I) IS USED

DO 20 I=1,NRA

II=II+1 @ II=II+J(J)

IT IS MORE EFFICIENT TO USE A PRESTORRED VECTOR OF DIAGONALS WITH JJ(I)=I*(I+1)/2, AND TO SET II=JJ(I) AND IJ=J(J)

C

SUM=0.00

IF (I.GT.IA) GO TO 15

IK=II

DO 10 K=1,IA

SUM=SUM+R(IK)*A(K,J)

10 IK=IK+K

15 IF (J.GT.IA) AND (I.LT.J) SUM=SUM+R(IJ+1)

C

20 RA(I,J)=SUM

30 IF (J.GT.IA) IJ=IJ+J @ IJ=JJ(J)

C

RETURN

END

101
SUBROUTINE RANK1 (UIN*UOUT,N*C,V)

STABLE U-D FACTOR RANK 1 UPDATE

(UOUT)\*DOUT(\*UOUT)\*T=(UIN)\*DIN(\*UIN)\*T+C\*V\*V\*T

UIN(N*(N+1)/2) INPUTCTOR STORED POSITIVE SEMI-DEFINITE U-D
ARRAY, WITH D ELEMENTS STORED ON THE DIAGONAL
UOUT(N*(N+1)/2) OUTPUT VECTOR STORED POSITIVE (POSSIBLY) SEMI-
DEFINITE U-D RESULT. UOUT=UIN IS PERMITTED.
N MATRIX DIMENSION, N>GE.2
C INPUT SCALAR. SHOULD BE NON-NEGATIVE
C IS DESTROYED DURING THE PROCESS
V(N) INPUT VECTOR FOR RANK ONE MODIFICATION.
V IS DESTROYED DURING THE PROCESS

COGNIZANT PERSONS: G.J.RIERMANN/M.W.JEDEAD (JPL,SEPT.1977)

IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION UIN(1), UOUT(1), V(1)
DOUBLE PRECISION ALPHA, BETA, S, N, EPS, TST

DATA EPS=0.DO/, TST=0.62500/
IN SINGLE PRECISION EPSILON IS MACHINE ACCURACY

TST=1/16 IS USED FOR RANK1 ALGORITHM SWITCHING

Z=0.DO
JJ=N*(N+1)/2
IF (C\*GT\*Z) GO TO 4
DO 1 JJ=1,JJ
1 UOUT(J)=UIN(J)
RETURN

4 NP2=N+2
DO 70 JJ=1,N
J=NP2-J
S=V(J)
BETA=C\*S
D=UIN(JJ)+BETA*S
IF (D\*GT.EPS) GO TO 30
IF (D\*GE.Z) GO TO 10
5 WRITE (6,100)
RETURN

10 JJ=J-J
WRITE (6,110)
DO 20 K=1,J
20 UOUT(JJ+K)=Z
GO TO 70
30 BETA=BE\*A/D
ALPHA=UIN(JJ)/D
C=ALPHA*C
UOUT(JJ)=D
JJ=JJ-J
JMI=J-1
IF (ALPHA.LT.TST) GO TO 50
DO 40 I=1,JM1
   V(I)=V(I)-S*UIN(JJ+I)
40   UOUT(JJ+I)=RETA*V(I)+UIN(JJ+I)
GO TO 70
50 DO 60 I=1,JM1
   D=V(I)-S*UIN(JJ+I)
   UOUT(JJ+I)=ALPHA*UIN(JJ+I)+RETA*V(I)
60   V(I)=D
70 CONTINUE
C
UOUT(1)=UIN(1)+C*V(1)**2
RETURN
C
100 FORMAT (1H0,10X,'** ERROR RETURN DUE TO A COMPUTED NEGATIVE
       INPUTED DIAGONAL IN RANK 1 ** **)
110 FORMAT (1H0,10X,'** NOTE: U-D RESULT IS SINGULAR ** **)
END
SUBROUTINE PCOLRD(S,MaxS,IRS,JCS,NPSTRT,NP,FN,W,ZV,SGSTAR) 

TO ADD IN PROCESS NOISE EFFECTS INTO THE SQUARE ROOT 
INFORMATION FILTER, AND TO GENERATE WEIGHTING COEFFICIENTS 
FOR SMOOTHING. IT IS ASSUMED THAT VARIABLES X(NPSTRT), 
X(NPSTRT+1),...,X(NPSTRT+NP-1) ARE COLORED NOISE AND THAT 
EACH COMPONENT SATISES A MODEL EQUATION OF THE FORM 
X(NPSTRT+J+1)=EM*X(NPSTRT+J)+W(NPSTRT+J), FOR DETAILS, SEE 
'FACTORIZATION METHODS FOR DISCRETE SEQUENTIAL ESTIMATION'. 
G.J. BIERMAN, ACADEMIC PRESS (1977) 

FOR SMOOTHING, REMOVE THE COMMENT STATEMENTS ON THE 3 LINES 
OF "SMOOTHING ONLY" CODE. THE SIGNIFICANCE OF THE SMOOTHING 
MATRIX IS EXPLAINED IN THE FUNCTIONAL DESCRIPTION. 

S(IRS,JCS) INPUT SQUARE ROOT INFORMATION ARRAY. OUTPUT COLORED 
NOISE ARRAY HOUSED HERE TOO. IF THERE IS SMOOTHING, 
NO ADDITIONAL ROWS ARE INCLUDED IN S. 
'MAXS' ROW DIMENSION OF S. IF THERE ARE SMOOTHING COMPUTA-
ITIONS IT IS NECESSARY THAT MAXS.GE.IRS+NP BECAUSE 
THE BOTTOM NP ROWS OF S HOUSE THE SMOOTHING 
INFORMATION 
IRS NUMBER OF ROWS OF S (IGE. NUMBER OF FILTER VARIABLES) 
(JCS.GE.2) 
JCS NUMBER OF COLUMNS OF S (EQUAL NUMBER OF FILTER 
VARIABLES + POSSIBLY A RIGHT SIDE) WHICH CONTAINS 
The DATA EQUATION NORMALIZED ESTIMATE (JCS.GE.1) 

NPSTRT LOCATION OF THE FIRST COLORED NOISE VARIABLE 
(1.LE.NPSTRT.LE.JCS) 
NP NUMBER OF CONTIGUOUS COLORED NOISE VARIABLES (NP.GE.1) 
EM(NP) COLORED NOISE MAPPING COEFFICIENTS 
(OF EXPONENTIAL FORM: EM=EXP(-DT/TAU)) 
R(NP) RECIPROCAL PROCESS NOISE STANDARD DEVIATIONS 
(MUST BE POSITIVE) 
Z(NP) Z(NP)=W-ESTIMATE (PROCESS NOISE ESTIMATES ARE 
GENERALY ZERO MEAN). WHEN ZW=0 ONE CAN OMIT THE 
RIGHT HAND SIDE COLUMN. 
V(IRS) WORK VECTOR 
SGSTAR(NP) VECTOR OF SMOOTHING COEFFICIENTS. WHEN THE SMOOTHING 
CODE IS COMMENTED OUT SGSTAR IS NOT USED. 

COGNIZANT PERSONS: G.J. BIERMAN/M.W. NEAD (JPL, FEB.1978) 

IMPLICIT DOUBLE PRECISION (A-H,O-Z) 
DIMENSION S(MAXS,JCS),EM(NP),W(NP),V(IRS),SGSTAR(1) 
DOUBLE PRECISION ALPHA,SIGMA,BETA,GAMMA 

ZERO=0.00 
ONE=1.00 

DO 70 JCOLRD=1,NP 
ALPHA=RW(JCOLRD)*FM(JCOLP) 
SIGMA=ALPHA**2 
DO 10 K=1,IRS 
V(K)=S(K,NPCOL) 

104
C TRANSFORMATION VECTOR

10 SIGMA=SIGMA+V(K)**2
SIGMA=DSORT(SIGMA)
ALPHA=ALPHA-SIGMA  Q LAST ELEMENT OF HOUSEHOLDER
TRANSFORMATION VECTOR

C

C ** ** **

C SSTAR(JCOLRD)=SIGMA  Q USFD FOR SMOOTHING ONLY

C

C ** ** **

C BETA=ONE/(SIGMA*ALPHA)  Q HOUSEHOLDER=I+RFTA*V*V**T

C HOUSEHOLDER TRANSFORMATION DEFINED, NOW APPLY TO S, I.E. 60 LONPCOLR650

DO 60 KOL=1,JCS
IF (KOLNE.NPCOL) GO TO 30
GAMMA= RW(JCOLRD)*ALPHA*BETA
C

C ** ** **

C S(IRS+JCOLRD,NPCOL)=RW(JCOLRD)+GAMMA*ALPHA  Q SMOOTHING ONLY

C

C ** ** **

C DO 20 K=1,IRS
20 S(K,NPCOL)=GAMMA*V(K)

C

C ** ** **

C DO 20 K=1,IRS
20 S(K,NPCOL)=GAMMA*V(K)

C

C ** ** **

C IF ZW ALWAYS ZERO, COMMENT OUT THE ABOVE IF TEST
C

C

C IF ZW ALWAYS ZERO, COMMENT OUT THE ABOVE IF TEST
C

C

C

C

C

C

C

C RETURN
END
SUBROUTINE RINCON (RIN,N,ROUT,NCH)
C
TO COMPUTE THE INVERSE OF THE UPPER TRIANGULAR VECTOR STORED
INPUT MATRIX RIN AND STORE THE RESULT IN ROUT. (RIN=ROUT IS
PERMITTED) AND TO COMPUTE A CONDITION NUMBER ESTIMATE.
CNB=FR0R.NORM(RIN)*FR0R.NORM(RIN)**1).
THE FROBENIUS NORM IS THE SQUARE ROOT OF THE SUM OF SQUARES
OF THE ELEMENTS. THIS CONDITION NUMBER BOUND IS USED AS
AN UPPER BOUND AND IT ACTS AS A LOWER BOUND ON THE ACTUAL
CONDITION NUMBER OF THE PROBLEM. (SEE THE BOOK "SOLVING LEAST
SQUARES" BY LAWSON AND HANSON)
C
IF RIN IS SINGULAR, RINCON COMPUTES THE INVERSE TO THE LEFT OF
THE FIRST ZERO DIAGONAL, A MESSAGE IS PRINTED AND THE CONDITION
NUMBER BOUND COMPUTATION IS ABORTED.
C
RIN(N*(N+1)/2) INPUT VECTOR STORED UPPER TRIANGULAR MATRIX
N  DIMENSION OF R MATRICES, N.GE.2
ROUT(N*(N+1)/2) OUTPUT VECTOR STORED UPPER TRIANGULAR MATRIX
C
CNB  CONDITION NUMBER BOUND. IF CNB IS THE CONDITION
      NUMBER OF RIN, THEN CNB/N.LE.10.
C
COGNIZANT PERSONS: G.J.BIFMAH/M.W.HAN (JPL/FFR.197A)
C
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DOUBLE PRECISION RNM,DINV,SUM,RNMOUT
DIMENSION RIN(1), ROUT(1)
C
Z=0.00
ONE=1.00
NTOT=N*(N+1)/2
C
RNM=Z
DO 10 J=1,NTOT
  10 RNM=RNM+RIN(J)**2
C
REPLACE CALL UTINV (RIN,N,ROUT) BY UTINV CODE
C
IF (RIN(1).NE.2) GO TO 20
J=1
WRITE (6,100) J,J
RETURN
C
20 ROUT(1)=ONE/RIN(1)
C
JJ=1
DO 50 J=2,N
  50 JJ0LD=JJ
  JJ=JJ+J
  IF (RIN(JJ).NE.2) GO TO 30
  WRITE (6,100) J,J
  RETURN
30  DINV=ONE/RIN(JJ)
   ROUT(JJ)=DINV
   II=0
   IK=1
   JM1=J-1
   DO 50  I=1,JM1
      II=II+1
      IK=II
      SUM=Z
      DO 40  K=I+1,JM1
         SUM=SUM+ROUT(IK)*RIN(JJOLD+K)
      40     IK=IK+K
   50    ROUT(JJOLD+I)=SUM/DINV
C    C
C
C
RNMOUT=Z
   DO 60  J=1,NTOT
      RNMOUT=RNMOUT+ROUT(J)*2
   60    RNMSQRT(RNM*RNMOUT)
      CNB=RNM
C
C
WRITE (6,110) RNMSQRT
RETURN
C
100 FORMAT (1H0,10X,** MATRIX INVERSE COMPUTED ONLY UP TO BUT NOT RINCRAD
  INCLUDING COLUMN**14,** MATRIX DIAGONAL **14,** IS ZERO **RTNCRAD30
  2)
110 FORMAT(1H0,5X,** CONDITION NUMBER ROUNDED**0.018.10**,CN/STC.R0.0
  **CN/NDT**)
END
SUBROUTINE RI2COV (RINV,N,SIG,COVOUT,KROW,KCOL)

C ** compute the covariance matrix and/or the standard deviations.
C 0: a vector stored upper triangular square root covariance matrix. the output covariance matrix is also vector stored.
C
C RINV(N*(N+1)/2) input vector stored upper triangular covariance square root. (RINV=INVERSE OF THE RINV MATRIX)
C N dimension of the RINV matrix, N.GT.2
C SIG(N) output vector of standard deviations
C COVOUT(N*(N+1)/2) output vector stored covariance matrix (COVOUT = RINV IS ALLOWED)
C KROW .GT.0 computes the covariance and sigmas corresponding to the first KROW variables of the RINV matrix.
C KROW .LT.0 computes only the sigmas of the first KROW variables of the RINV matrix.
C KROW .EQ.0 no covariance, but all sigmas (e.g. use RINV to compute them)
C N rows of RINV.
C KCOL no. of columns of COVOUT that are computed
C IF KCOL.LE.0 THEN KCOL=KROW. IF KROW.LE.0 THEN THIS INPUT IS IGNORED.
C
C COGNIZANT PERSONS: G.J.RIEQMAN/M.W.NEAD (JPL, MARCH 1978)

C IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C DOUBLE PRECISION SUM
C DIMENSION RINV(1), SIG(1), COVOUT(1)
C
C ZERO=3.0D0
C LIM=NN
C KCOL=KCOL
C IF (KCOL.LE.0) KCOL=KROW
C IF (KROW.LE.0) LIM=INTARS(KROW)
C      ** compute sigmas
C
C IKS=0
C DO 2 J=1,LIM
C      IKS=IKS+J
C      SUM=SUM+RINV(IK)**2
C 1      IK=IK+1
C 2      SIG(J)=DSQRT(SUM)
C
C IF (KROW.LE.0) RETURN
C      ** compute covariance
C
C     ..0
C NMI=LIM
C IF (KROW.EQ.N) NMI=N-1
C DO 10 J=1,NMI:
C      JJ=J+J
C      COVOUT(JJ)=SIG(J)**2
C 10   

108
IJS=JJ+J
JP1=J+1
DO 10 I=JP1*KKOL
   IK=IJS
   IMJ=I-J
   SUM=ZERO
   DO 5 K=I+1
      IJK=IK+IMJ
      SUM=SUM+PHI(K)*PHI(JK)
   5 CONTINUE
   IK=IK+K
   COVOUT(IJS)=SUM
10   IJS=IJS+1
C
   IF (KROW.EQ.N) COVOUT(JJ+N)=SIG(N)**2
RETURN
END
SUBROUTINE R2A(P,LR,NAMR,A,IA,LA,NAMA)

TO PLACE THE TRIANGULAR VECTOR STORED MATRIX R INTO THE MATRIX A AND TO ARRANGE THE COLUMNS TO MATCH THE DESIRED NAMA PARAMETER LIST. NAMES IN THE NAMA LIST THAT DO NOT CORRESPOND TO ANY NAME IN NAMR HAVE ZERO ENTRIES IN THE CORRESPONDING COLUMN.

R(LR*(LR+1)/2) INPUT UPPER TRIANGULAR VECTOR STORED ARRAY
   LR DIMENSION OF R
   NAMR(L) PARAMETER NAMES ASSOCIATED WITH R
   A(LR*LA) MATRIX TO HOUSE THE REARRANGED R MATRIX
   IA ROW DIMENSION OF A, IA>=LR
   LA NO. OF PARAMETER NAMES ASSOCIATED WITH THE NAMA(LA) PARAMETER NAMES FOR THE OUTPUT A MATRIX

COGNIZANT PERSONS: G.J.RIEPMAN/M.W.NEAD (JPL SEPT. 1976)

IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION R(1),NAMR(1),A(IA,1),NAMA(1)

ZERO=0.
DO 5 J=1,LA
   DO 5 K=1,LR  @ ZERO A(LP,LA)
   5 A(K*J)=ZERO
   DO 40 J=1,LA
      DO 10 I=1,LR
         IF (NAMR(I).EQ.NAMA(J)) GO TO 20
      10 CONTINUE
      GO TO 40
   20 JJ=I*(I-1)/2
      DO 30 K=1,I
         A(K*J)=R(JJ+K)
      30 CONTINUE
   40 CONTINUE

RETURN
END
SUBROUTINE R2RA (R,NR,NAM,RA,NPA,NAMA)  

TO COPY THE UPPER LEFT (LOWER RIGHT) PORTION OF A VECTOR 
STORED UPPER TRIANGULAR MATRIX R INTO THE LOWER RIGHT 
(UPPER LEFT) PORTION OF A VECTOR STORED TRIANGULAR 
MATRIX RA.  

R(NR*(NR+1)/2) INPUT VECTOR STORED UPPER TRIANGULAR MATRIX 
NR DIMENSION OF R  
NAM(NR) NAMES ASSOCIATED WITH R  
THIS INPUT NAMELIST IS DESTROYED  
RA(NRA*(NRA+1)/2) OUTPUT VECTOR STORED UPPER TRIANGULAR MATRIX 
NRA IF NRA=0 ON INPUT, THEN NAMA(1) SHOULD HAVE 
IN THIS CASE THE NUMBER OF NAMES IN NAMA AND 
NR WILL BE COMPUTED. THE LOWER RIGHT BLOCK OF R 
IF NRA=LAST NAME OF THE UPPER LEFT BLOCK 
OF R WILL BE THE UPPPR LEFT BLOCK OF PA, 
THEN THE UPPER BLOCK IS TO BE MOVED TO THE LOWER 
CORNER OF RA. WHEN USED IN THIS M0D NAME=NR 
ON OUTPUT.  
NAMA(NRA) NAMES ASSOCIATED WITH PA  
IF NRA=0 ON INPUT, THEN NAMA(1) SHOULD HAVE THE FIRST NAME OF THE 
OUTPUT NAMELIST AND THE NUMBER OF NAMES IN NAMA IS COMPUTED. 
THE LOWER RIGHT BLOCK OF R WILL BE THE UPPER LEFT BLOCK OF RA. 
IF NRA=LAST NAME OF THE UPPER LEFT BLOCK THAT IS TO BE MOVFED, 
THEN THE UPPER BLOCK IS TO BE MOVED TO THE LOWER RIGHT 
POSITION. WHEN USED IN THIS M0D NR=NR 
ON OUTPUT.  
THE NAMES OF THE RELOCATED BLOCK ARE ALSO MOVFD. THE RESULT 
CAN COINCIDE WITH R AND NAMA WITH NAME.  

COGNIZANT PERSONS: G.J.RIERMAN/M.W.NEAD (JPL, SEPT. 1976)  

IMPLICIT DOUBLE PRECISION (A-H,O-Z)  
DIMENSION R(1),RA(1),NAM(1),NAMA(1)  
LOGICAL IS 

IS=.FALSE.  
LOCN=NAMA(1)  
IS=FALSE CORRESPONDS TO MOVING UPPER LEFT CORNER OF R TO 
LOWER RT. CORNER OF RA  
IF (NRA.EQ.0) GO TO 1  
LOCN=NRA  
IS=.TRUE.  
IS=TRUE CORRESPONDS TO MOVING LOWER LEFT CORNER OF R TO 
UPPER RT. CORNER OF RA  
1 DO 3 I=1,NR  
IF (NAM(I).EQ.LOCN) GO TO 4  
3 CONTINUE  
WRITE (6,100)  
100 FORMAT (1H0,20X,'NAMA(1) NOT IN NAMELIST OF R MATRIX')
RETURN  

C 4 K=I  
   KM=K-1  
   IF (K) GO TO 15  
C  
   IJS=K*(K+1)/2-1  
   NRA=NR*K+1  
   IJA=0  
   KOLA=0  
   DO 10 KOL=K+NR  
      KOLA=KOLA+1  
      NAMA(KOLA-KM)=NAM(KOLA)  
      DO 5 IR=1*KOLA  
         IJA=IJA+1  
      5      RA(IJA)=R(IJS+IR)  
   10 IJS=IJS*KOL  
RETURN  
C  
   IJ=K*(K+1)/2  
   IJA=NR*(NR+1)/2  
   L=NR*KM1  
   KOLA=K  
   DO 25 KOLA=NR*L+1  
      IJS=IJA  
      NAMA(KOLA)=NAM(KOLA)  
      DO 20 IR=KOLA+L+1  
         RA(IJS)=R(IJ)  
         IJS=IJS-1  
      20      IJ=IJ-1  
   25 IJA=IJA-KOLA  
   NRA=NR  
C  
RETURN  
END
SUBROUTINE RUOR(RIN,N,ROUT,IS)

C FOR N.GT.0 THIS SUBROUTINE TRANSFORMS AN UPPER TRIANGULAR VECTOR
C STORED SRIF MATRIX TO U-D FORM, AND WHEN N.LT.0 THE U-D VECTOR
C STORED ARRAY IS TRANSFORMED TO A VECTOR STORED SRIF ARRAY
C
C RIN((N+1)*(N+2)/2) INPUT VECTOR STORED SRIF OR U-D ARRAY
C ROUT((N+1)*(N+2)/2) OUTPUT IS THE CORRESPONDING U-D OR SRIF
C N INPUT (RIN=ROUT IS PERMITTED)
C N.GT.0 THE (INPUT) SRIF ARRAY IS (OUTPUT)
C N.LT.0 THE (INPUT) U-D ARRAY IS (OUTPUT)
C IS = 0 THERE IS NO RT. SINF OR ESTIMATE STORED IN
C COLUMN N+1, AND DIN NEED HAVE ONLY
C N COLUMNS, I.E. RIN(N(N+1)/2)
C IS = 1 THERE IS A RT. SINF INPUT TO THE SRIF AND
C AN ESTIMATE FOR THE U-D ARRAY, THESE RESIDE
C IN COLUMN N+1.
C
THIS SUBROUTINE USES SUBROUTINE RINCON
C
COGNIZANT PERSONS G.J.BIERMAN/M.W.NEAD (JPL, FFPE197A)

C IMPLICIT DOUBLE PRECISION (A-H,O-7)
C DIMENSION RIN(1), ROUT(1)
C
ONE= 1.0D0
NP1= IS + IABS(N)
JJ=1
IDIMR= NP1*(NP1 +1)/2
IF (IS.EQ.0) GO TO 5
RNN=RIN(IDIMR)
RIN(IDIMR)= ONE

5 IF (N.LT.0) GO TO 30
CALL RINCON(RIN(NP1),ROUT,CNA)
ROUT(1)= ROUT(1)**2
DO 20 J=2,N
S=ONE/ROUT(J+J)
ROUT(J+J)= ROUT(J+J)**2
JM1=J-1
DO 10 I=1,JM1
10 ROUT(J+I)= ROUT(J+I)*S
20 JJ=JJ+ J
GO TO 70

C 30 NNE=-N
ROUT(1)= SQRT(RIN(1))
C
*** SOME MACHINES REQUIRE DSORT FOR DOUBLE PRECISION
C
DO 50 J=2,N
ROUT(J+J)= SQRT(RIN(J+J))
50 ROUT(JJ+I)= RIN(JJ+1)*S
60 CALL RINCON(ROUT*NP1,ROUT*CNP)

C
70 IF (IS.EQ.1) RIN(IDIMR)=RHN
RETURN
END
SUBROUTINE SFU(FEL, IROW, JCOL, NF, IFU, MAXFU, IFJU, JDIAG)

TO COMPUTE FU(IFU,N)=-FIU WHERE F IS SPARSE AND ONLY THE NON-ZERO ELEMENTS ARE DEFINED AND U IS VECTOR STORED.

UPPER TRIANGULAR WITH IMPLICITLY DEFINED UNIT DIAGONAL ELEMENTS

FEL(NF)   VALUES OF THE NON-ZERO ELEMENTS OF THE F MATRIX
IROW(NF)   ROW INDICES OF THE F ELEMENTS
JCOL(NF)   COLUMN INDICES OF THE F ELEMENTS
FI(ROW(K))=FEL(K)
NF       NUMBER OF NON-ZERO ELEMENTS OF THE F MATRIX
UP(N*(N+1)/2)   UPPER TRIANGULAR VECTOR STORED MATRIX WITH IMPLICITLY DEFINED UNIT DIAGONAL ELEMENTS
(I,J) ARE NOT IN FACT, (UNITY)
N       DIMENSION OF U MATRIX
IFU       OUTPUT RESULT
MAXFU     ROW DIMENSION OF FII MATRIX
I.E. FII MUST HAVE AT LEAST AS MANY ROWS AS FII.

ADDITIONAL ROWS OF FII COULD CORRESPOND TO ZERO ROWS OF F.

JDIAG(N)   DIAGONAL ELEMENT INDICES OF A VECTOR STORED UPPER TRIANGULAR MATRIX,
            I.E. JDIAG(K)=K*(K+1)/2=JDIAG(K-1)+K

COGNIZANT PERSONS: G. J. R. H.-S. M. W. HEAD (JPL 70, FER 70)

IMPLICIT DOUBLE PRECISION (A-H, O-Z; DIMENSION FEL(NF), IROW(NF), JCOL(NF), JDIAG(N)

DO 10 J=1,N
      DO 10 I=1,IFU
            IF(MAXFU=IFU) IT IS MORE EFFICIENT TO REPLACE THIS LOOP BY
C                             DO 10 IFUN=IFU:N
                           10    FU(I,J)=ZERO
            DO 10 IJ=1,IFU
         10   IF(IF(J,J)=ZERO
            DO 30 NEL=1,NF
                 NEL REPRESENTS THE ELEMENT NUMBER OF THE F MATRIX
                 I=IROW(NEL)
                 J=JCOL(NEL)
                 FIU=FEL(NEL)
                 FU(I,J)=FU(I,J)+FIU
            THIS ACCOUNTS FOR THE IMPLICIT UNIT DIAGONAL U MATRIX
            WHEN NON-UNIT DIAGONALS ARE USED, DELETE THE ABOVE LINE AND USE J INSTEAD OF JP1 RFLOW
            IF (J,EQ.N) GO TO 70
            WHEN IT IS KNOWN THAT THE LAST COLUMN OF F IS ZERO
            THIS 'IF' TEST MAY BE OMITTED
C         JP1=J+1

SF1000010
SF1000020
SF1000030
SF1000040
SF1000050
SF1000060
SF1000070
SF1000080
SF1000090
SF1000100
SF1000110
SF1000120
SF1000130
SF1000140
SF1000150
SF1000160
SF1000170
SF1000180
SF1000190
SF1000200
SF1000210
SF1000220
SF1000230
SF1000240
SF1000250
SF1000260
SF1000270
SF1000280
SF1000290
SF1000300
SF1000310
SF1000320
SF1000330
SF1000340
SF1000350
SF1000360
SF1000370
SF1000380
SF1000390
SF1000400
SF1000410
SF1000420
SF1000430
SF1000440
SF1000450
SF1000460
SF1000470
SF1000480
SF1000490
SF1000500
SF1000510
SF1000520
SF1000530
SF1000540
SF1000550
IK = JDIAG(J) + J
DO 20 K = JP1 + N
   FU(I*K) = FU(I*K) + FIJ*U(IK)
20    IK = IK + K
30    CONTINUE
C
RETURN
END
C

SUM=ONE/(SUM*V(J))

THE HOUSEHOLDER TRANSFORMATION IS I=SUM*V*V**T

JPI=J+1

IF (JPI.GT.JCS) 60 TO 40

DO 30 K=JPI JCS

DELTA=ZERO

DO 20 I=J+1 PS

20 DELTA=DELTA+S(I*K)*V(I)

DELTA=DELTA*SUM

DO 30 I=J+1 PS

30 S(I*K)=S(I*K)+DELTA*V(I)

40 CONTINUE

C

RETURN

END
SUBROUTINE THH(P,N,IA,M,SOS,NSRT)

C
C THIS SUBROUTINE PERFORMS A TRIANGULARIZATION OF A PECTANGULAR
C MATRIX INTO A SINGLE-SUBSCRIPTED ARRAY BY APPLICATION OF
C HOUSEHOLDER ORTHONORMAL TRANSFORMATIONS.
C
R(N*(N+3)/2) VECTOR STORED SQUARE ROOT INFORMATION MATRIX
(LAST N LOCATIONS MAY CONTAIN A RIGHT HAND SIDE)
N
C DIMENSION OF P MATRIX
A(N+1)
C MEASURMENT MATRIX
IA
C ROW DIMENSION OF A
M
C NUMBER OF ROWS OF A THAT ARE TO BE COMBINED WITH R
SOS
C ACCUMULATED ROOT SUM OF SQUARES OF THE RESIDUALS
SORT(Z-A*(EST)**2), INCLUDES A PRIORI
SOS MUST BE INITIALIZED AS A VARIABLE; NOT AS A
NUMERICAL VALUE. IF INPUT SOS.LT.0. NO SOS
COMPUTATION OCCURS.
NSRT
C FIRST COL OF THE INPUT A MATRIX THAT HAS A NONZERO
ENTRY. IF NSRT.LE.1, IT IS SET TO 1. THIS OPTION
IS CONVENIENT WHEN PACKING A PRIORI BY BATCHES AND
THE A MATRIX HAS LeADING COLUMNS OF ZFROS.
C
C ON ENTRY R CONTAINS A PRIORI SQUARE ROOT INFORMATION FILTER (SRIF)
ARRAY, AND ON EXIT IT CONTAINS THE A POSTERIORI (PACKED) ARRAY.
C ON ENTRY A CONTAINS OBSERVATIONS WHICH ARE DESTROYED BY THE
C INTERNAL COMPUTATIONS.
C ON ENTRY IF SOS IS .LT. ZERO PROGRAM WILL ASSUME THERE IS NO
C RIGHT HAND SIDE DATA AND WILL NOT ALTER SOS OR USE LAST N
C LOCATIONS OF VECTOR R.
C
C
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION A(IA+1),R(1)

DOUBLE PRECISION SUM, ONE, ETA, DELTA
C
FPS=.1D-200 Q MACHINE DEPENDENT ACCURACY TERM
ZERO=0.D0
ONE=1.D0
NSRT=NSRT
C
IF (NSTART.LE.0) NSTART=1
NP1=N+1
IF(SOS.LT.ZERO) NP1=N
KK=NSRT*(NSTART-1)/2
DO 100 J=NSTART,N
KM=KK+J
SUM=ZERO
DO 20 I=1,M
SUM=SUM+A(I,J)**2
20 SUM=SUM+R(KK)**2
IF(SUM.LE.ZERO) GO TO 100
Q J-TH STEP OF HOUSEHOLDER REDUCTION
K=KK+J
SUM=SUM+R(K)**2
SUM=DSORT(SUM)
100 CONTINUE
C
C
IF(R(K#) GT ZERO) SUM=-SUM
DELTA=R(K#)-SUM
R(K#)=SUM

JP1=JP1+1
IF (JP1 GT NP1) GO TO 105

BETA=SUM+DELTA
IF (BETA GT EPS) GO TO 100
BETA=ONE/BETA

J=KK
L=L

** READY TO APPLY J-TH HOUSEHOLDER TRANS. 
DO 40 K=JP1,NP1
JJ=J+L
SUM=DELTA*R(JJ)
DO 30 I=1,M
30 SUM=SUM+A(I+J)*A(I,K)
IF(SUM.EQ.ZERO) GO TO 40
SUM=SUM+BETA

C BETA DIVIDE USED HERE TO AVOID OVERFLOW IN
C PROBLEMS WITH NEAR COLUMN COLLINERITY. IN THAT CASE
C COMMENT OUT LINE 630 AND CHANGE * TO / IN LINE 740
R(JJ)=R(JJ)+SUM*DELTA
DO 35 I=1,M
35 A(I,K)=A(I+K)+SUM*A(I,J)
40 CONTINUE
100 CONTINUE

105 IF(SOS.LT.ZERO) RETURN

C CALCULATE SOS
C SUM=ZERO
GO TO 110 I=1,M
110 SUM=SUM+A(I+NP1)**2
SOS=DSQRT(SOS**2+SUM)
C RETURN
END
SUBROUTINE TTHH(R, PA, N)

THIS SUBROUTINE COMBINES TWO SINGLE SUBSCRIPTED SPRIF ARRAYS
USING HOUSEHOLDER ORTHOGONAL TRANSFORMATIONS

R(N*(N+1)/2) INPUT VECTOR STORED UPPER TRIANGULAR MATRIX.
RESULT IS IN P

RA(N*(N+1)/2) THE SECOND INPUT VECTOR STORED UPPER TRIANGULAR
MATRIX. THIS MATRIX IS DESTROYED BY THE
COMPUTATION

N DIMENSION OF THE ESTIMATED PARAMETER VECTOR.
A NEGATIVE VALUE FOR N IS USED TO NOTE THAT
R AND RA HAVE RT, HAND SIDES INCLUDED AND
HAVE DIM=ARS(N)*(ARS(N)+3)/2.

ON EXIT RA IS CHANGED AND R CONTAINS THE RESULTING SPRIF ARRAY

C Cognizant Persons G.J. Riehman/M.W. Head (JPL, Jan. 1976)
C
C IMPLICIT DOUBLE PRECISION(A-H, 0-Z)
C DIMENSION PA(1)* R(1)
C DOUBLE PRECISION SUM & FOR USE IN SINGLE PRECISION VERSION
C
C ZERO=0.
ONE=1.
NP1=N
IF (N.GT.0) GO TO 10
N=-N
NP1=N+1

10 IJS=1

10 DO 10 J=1, N

20 DO 30 I=IJS+KK

90 K=0

10 KK=KK+J

20 SUM=SUM+RA(I)**2

10 SUM=SUM+RA(I)**2

30 IF (SUM.LE.ZERO) GO TO 100

30 SUM=SQR(SUM)

100 IF (R(KK).GT.ZERO) SUM=-SUM

100 DELTA=R(KK)-SUM

100 R(KK)=SUM

100 L/J

100 JJ=J

100 JP1=J+1

100 IKS=KK+1

40 LOOP APPLIES TRANSFORM TO COLS. J+1 TO NPI

40 DO 40 K=JP1, NPI

40 JJ=JJ+L

40 L=L+1

40 IKS=IKS

40 SUM=DELTA*RA(JJ)

40 DO 50 I=IJS, KK

40 SUM=SUM+RA(IK)*PA(I)

50 CONTINUE
30  IK=IK+1
    IF (SUM.EQ.ZERO) GO TO 40
    SUM=SUM*BETA
    R(JJ)=R(JJ)+SUM*DELTA
    IK=IKS
    DO 35 I=IJS,KK
    RA(IK)=RA(IK)+SUM*RA(I)
    35  IK=IK+1
30  IKS=IKS+1
20  IJS=IJS+1
C
RETURN
END
SUBROUTINE TWOMAT (A+*LEN+*CAR+*TFY+*CHAR+*NAMES)

C
C TO DISPLAY A VECTOR STORED UPPER TRIANGULAR MATRIX IN A
C TWO-DIMENSIONAL TRIANGULAR FORMAT
C
C A(N*(N+1)/2) VECTOR CONTAINING UPPER TRIANGULAR MATRIX (DP)
C N DIMENSION OF MATRIX
C LEN NUMFR OF COLUMNS TO RE PRINTFD, 7 OR 12
C CAR(N) PARAMETER NAMES
C TEXT( ) AN ARRAY OF FIELD DATA CHARACTERS TO RE PRINTED AS
C A TITLE PRECEDING THE MATRIX
C NCHAR NUMBER OF CHARACTERS, INCLUDING SPACES, THAT
C ARE TO RE PRINTED IN TEXT( )
C ABS(NCHAR), LE=114. NCHAR NEGATIVE IS USED
C TO AVOID SKIPPING TO A NEW PAGE TO START
C PRINTING
C NAMES TRUF TO PRINT PARAMETER NAMES
C
C COGNIZANT PFRSON: M.W. NKSAD (JPL, OCT. 1977)
C
PARAMETER J12=12* J7=7
DOUBLE PRECISION A(N)
INTEGER CAR(N), TEXT(1), L(J12), LIST(J12)
LOGICAL NAMES
INTEGER V(4),VFMT(J12),VFMT(J7),VFMT(J12)
DATA V/'(2X,1X,E10.5)'/(VFMT(I),I=1,12)
DATA KON7/D17.A1+/ KON12/F10.5+/)
C
M1=M2 ROW LIMITS FOR EACH PRINT SEQUENCE
N1=N2 COL LIMITS FOR EACH LINE OF PRINT
L(I) LOC OF EACH COLUMN IN A ROW
KT ROW COUNTER
C
** ** ** INITIALIZE COUNTERS
C
IF (LEN.EQ.70) GO TO 5
IF (LEN.EQ.7) GO TO 1
IF (LEN.EQ.12) GO TO 2
WRITE (6,230) LFN
LEN=12
GO TO 2
1 V(4)=KON7; J0=7; J0M1=J0-1; J0P1=J0+1
1 REPEAT I=1; J0; VFMT(I)=VFMT(I)
GO TO 5
2 V(4)=KON12; J0=12; J0M1=J0-1; J0P1=J0+1
1 REPEAT I=1; J0; VFMT(I)=VFMT(I)
5 M1=1
M2=J0
N1=1
KT=0
V(2)=AD1X+1
IF (.NOT.*NAMES) V(2)=15,2X

123
C
NC=IAR5(NCHAR)/6
IF (MOD(NCHAR+6),NE.0) NC=NC+1
IF (NCHAR.GE.0) WRITE (6,200) (TEXT(I),I=1,NC)
IF (NCHAR.LT.0) WRITE (6,205) (TEXT(I),I=1,NC)
10 IF (M2,GT,N) M2=N
IF (.NOT.+NAMES) GO TO 20
IF (LEN.EQ.7) WRITE (6,210) (CAR(I),I=N1,M2)
IF (LEN.EQ.12) WRITE (6,211) (CAR(I),I=N1,M2)
GO TO 40
20 M=M1
L2=M2-N1+1
DO 30 I=1,L2
LIST(I)=M
30 N=N+1
IF (LEN.EQ.7) WRITE (6,220) (LIST(I),I=1,L2)
IF (LEN.EQ.12) WRITE (6,221) (LIST(I),I=1,L2)
40 CONTINUE
C
*** *** ***
DO 190 IC=M1,M2
K=1
IF (IC.LE.(KT*JO)) GO TO 60
JJ=0
DO 50 J=1,IC
JJ=JJ+J
L(K)=JJ
I1=IC-KT*JO
IF (I1.EQ.JO) GO TO 90
GO TO 70
60 CONTINUE
C
I1=1
L(K)=L(K)+1
70 CONTINUE
DO 80 I=I1,JO+1
K=K+1
I1=I1+KT*JO
80 L(K)=L(K)+I
90 CONTINUE
C
I2=M2NO((M2+1-KT*JO))-1
V(3)=VFMT(I1)
IF (.NOT.+NAMES) GO TO 180
WRITE (6,V) CAR(IC),(A(L(I)),I=1,12)
GO TO 190
180 WRITE (6,V) IC,(A(L(I)),I=1,12)
190 CONTINUE
IF (M2,GT,N) RETURN
N1=M2+1
M2=M2+JO
KT=KT+1
IF (NCHAR.GE.0) WRITE (6,201) (TEXT(I),I=1,NC)
IF (NCHAR.LT.0) WRITE (6,206) (TEXT(I),I=1,NC)
GO TO 10
C
200 FORMAT (1H1,2X,21A6) Q TITLE
205 FORMAT (1H0,2X,21A6) Q TITLE

124
201 FORMAT (1H1, 2X, 'CONTINUE') *19A6) TITLE
206 FORMAT (1H0, 2X, 'CONTINUE') *19A6) TITLE
210 FORMAT (1H0, 5X, 7(11X, A6)) HORIZONTAL NAMES
220 FORMAT (1H0, 3X, 7(11X, I6)) HORIZONTAL NAMES
211 FORMAT (1H0, 5X, 12(4X, A6)) HORIZONTAL NAMES
221 FORMAT (1H0, 3X, 12(4X, I6)) HORIZONTAL NAMES
230 FORMAT (1H0, 20X, 'TWO Mat Called with LENGTH = *13)

C
END
SUBROUTINE TZERO (R,N,IS,IF)
C
TO ZERO OUT ROWS IS (ISTART) TO IF (IFINAL) OF A VECTOR
STORED UPPER TRIANGULAR MATRIX
C
R(N*(N+1)/2) INPUT VECTOR STORED UPPER TRIANGULAR MATRIX
N DIMENSION OF R
IS FIRST ROW OF R THAT IS TO BE SET TO ZERO
IR LAST ROW OF R THAT IS TO BE SET TO ZERO
C
COGNIZANT PERSONS: G.J.RIEMAN/C.F.PETERS (JPL, NOV. 1975)
C
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION R(1)
C
ZERO=U.DO
IJS=IS*(IS-1)/2
DO 10 I=IS+1,IF
IJS=IJS+I
IJ=IJS
DO 10 J=I+N
R(IJ)=ZERO
IJ=IJ+J
10 CONTINUE
C
RETURN
END
SUBROUTINE UDCOL(U,N,KS,NCOLOR,V,FM,Q)

COLORED NOISE UPDATING OF THE U-D COVARIANCE FACTORS, I.E.
U*U'-PHI*U*DIAG(0,1,...,0,0)+
PHI=DIAG(0(KS-1),FM(1),...,EM(NCOLOR),0(KS-1+NCOLOR)),
Q=DIAG(0(KS-1),Q(1),...,Q(NCOLOR),0(KS-1+NCOLOR)),
Q(K) IS A VECTOR OF ZEROS

THE ALGORITHM USED IS THE RIFRMAN-THORNTON ONE COMPONENT
AT-A-TIME UPDATE CF, RIFRMAN & FACTORIZATION METHOD
FOR DISCRETE SEQUENTIAL ESTIMATIONS, ACADEMIC PRESS (1977)
PP. 147-148

U(N*(N+1)/2) INPUT U-D VECTOR STORED COVARIANCE FACTORS,
THF COLORED NOISE UPDATE RESULT RESIDES IN U ON OUTPUT
N FILTER DIMENSION, IF THE LAST COLUMN OF U HOUSES THE FILTER ESTIMATES, THEN
N=NUMBER FILTER VARIABLES + 1
KS THE LOCATION OF THE FIRST COLORED NOISE TERM
(KS,GE,1,AND,KS,LF,N)
NCOLOR THE NUMBER OF COLORFD NOISE TERMS (NCOLOR,GE,1)
V(KS-1+NCOLOR) WORK VECTOR
EM(NCOLOR) INPUT VECTOR OF COLORED NOISE MAPPING TERMS
Q(NCOLOR) INPUT VECTOR OF PROCESS NOISE VARIANCES

SUBROUTINE REQUIRED: RANK1

COGNIZANT PERSON: G.J.RIFRMAN (JPL, JAN. 1977)
DOUBLE PRECISION TMP,S
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION U(1),V(1),EM(1),Q(1)

C
C *** *** *** INITIALIZATION
N1=N-1
KSM=KS-1
JJOLOD=KS*KSM1/2
KOL=KSM
C *** *** ***
DO 50 K=1,NCOLOR
KOL1=KOL
KOL=KOL+1
JJ=JJOLOD*KOL
TMP=U(JJ)*EM(K)
C=Q(K)*U(JJ)
S=TMP*EM(K)+Q(K)
U(JJ)=S
QD(J) UPDATE
C
IF(KOL,GE,N) GO TO 20
IJ JJ
DO 10 J=KOL,N1
IJ=IJ+J
10 CONTINUE

127
10 U(IJ)=U(IJ)*EM(K)  * UPDATING ROW KOL ENTRIES

20 IF (JJ.EQ.1) GO TO 50       * WHEN KS=1, N=1
   IF (S.LE.0.00) GO TO 30
   TMP=TMP/S         * TMP=EM(K)*D(KOL)-OLD/D(KOL)-NEW
   C=C/S             * C=Q(K)*D(KOL)-OLD/D(KOL)-NEW

30 DO 40 I=1,KOLM1
   V(I)=U(JJOLD+I)
   U(JJOLD+I)=TMP*V(I)
   IF (KOLM1.GT.1) GO TO 45
   U(I)=U(I)+C*V(I)**2
   GO TO 50
45 CALL RANK1(U+U,KOLM1,C,V)
50 JJOLD=JJ

RETURN
END
SUBROUTINE UDMEAS (U,N,R,A,F,G,ALPHA)

COMPUTES ESTIMATE AND MEASUREMENT UPDATED COVARIANCE, P=UD*DiagT

*** INPUTS ***
U UPPER TRIANGULAR MATRIX, WITH N ELEMENTS STORED AS THE DIAGONAL. U IS VECTOR STORED AND CORRESPONDS TO THE A PRIORI COVARIANCE. IF STATE ESTIMATES ARE COMPUTED, THE LAST COLUMN OF U CONTAINS X.
N DIMENSION OF THE STATE ESTIMATE, N+G+1
R MEASUREMENT VARIANCE
A VECTOR OF MEASUREMENT COEFFICIENTS, IF DATA THEN A(N+1)=Z
ALPHA IF ALPHA LESS THAN ZERO NO ESTIMATES ARE COMPUTED (AND X AND Z NEED NOT BE INCLUDED)

*** OUTPUTS ***
U UPDATED, VECTOR STORED FACTORS AND ESTIMATE AND
U((N+1)(N+2)/2) CONTAINS (Z-A**T*X)
ALPHA INNOVATIONS VARIANCE OF THE MEASUREMENT RESIDUAL
G VECTOR OF UNWEIGHTED KALMAN GAINS, THE KALMAN GAIN K IS EQUAL TO G/ALPHA
F CONTAINS (A**T*A)/(2*ALPHA) ONE CAN HAVE F OVERWRITE A TO SAVE STORAGE


IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION U(1), A(1), F(1), G(1)
DOUBLE PRECISION SUM,BETA,GAMMA
LOGICAL IEST

ZERO=0.00
IEST=.FALSE.
ONE=1.00
NP1=N+1
NP2=N+2
NTOT=N*NP1/2
IF (ALPHA.LE.ZERO) GO TO 3
SUM=A(NP1)
DO 1 J=1,N
1 SUM=SUM-A(J)*U(NTOT+J)

IF (ALPHA.LE.ZERO) GO TO 3
SUM=A(NP1)
U(NTOT+NP1)=SUM
IEST=.TRUE.

JNN=NTOT
DO 10 L=2,N
J=NP2-L
JJ=JNN-J
SUM=A(J)
JH1=J-1
DO 5 K=1,JH1

10 CONTINUE

129
SUM=SUM+U(JJ+K)*F(J)
F(J)=SUM
G(J)=SUM*U(JJN)

J=J+1
G(J)=U(JJ+1)*F(J)

F=U*T*A AND G=U*(U*T*A)

SUM=R*G(J)*F(J)
GAMMA=0
IF (SUM.GT.ZERO) GAMMA=ONE/SUM
IF (F(J).NE.ZERO) U(JJ)=U(JJ)*R*GAMMA

DO 15 K=1,JM
      TEMP=G(K)
      SUM=SUM+TEMP*F(J)
P=F(J)*GAMMA
      JM=JM+1
      DO 15 K=1,JM
      U(K)=SUM+TEMP*F(J)
P=-F(J)*(SUM(J-1))
      JM=JM+1
      DO 15 K=1,JM
      U(K)=SUM+TEMP*F(J)
P=-F(J)*(SUM(J-1))
      K=K+1
      IF (TEMP.EQ.ZERO) GO TO 20
      GAMMA=I/SUM
      U(KJ)=U(KJ)*GAMMA
      IF (.NOT.IEST) RETURN
      F(NP)=U(NTOT+NP)*GAMMA
      DO 30 J=1,N
      U(NTOT+J)=U(NTOT+J)+G(J)*F(NP)
      RETURN
      END
SUBROUTINE U2COV (UIN, POUT, N)

TO OBTAIN A COVARIANCE FROM ITS U-D FACTORIZATION, BOTH MATRICES ARE VECTOR STORED AND THE OUTPUT COVARIANCE CAN OVERWRITE THE INPUT U-D ARRAY. UIN=U-D IS RELATED TO POUT VIA POUT=UIN(*T)

N*(N+1)/2 INPUT U-D FACTORS, VECTOR STORED WITH THE N ENTRIES STORED ON THE DIAGONAL OF UIN

POUT(N*(N+1)/2) OUTPUT COVARIANCE, VECTOR STORED

C DIMENSION OF THE MATRICES INVOLVED, N*GT.1

COGNIZANT PERSONS: G.J. BIERMAN/W.W. HEAD (JPL, FEB. 1977)

IMPLICIT DOUBLE PRECISION (A-H, O-7)

DIMENSION UIN(1), POUT(1)

OUT(1)=UIN(1)

DO 20 J=2, N

JJ=J

JJL=JJ

POUT(JJ)=UIN(JJ)

S=POUT(JJ)

II=0

JM1=J-1

DO 20 I=1, JM1

II=II+1

ALPHA=S*UIN(JJL+II)  Q JJL+II=(I,J)

IK=II

DO 10 K=I, JM1

POUT(IK)=POUT(IK)+ALPHA*UIN(JJL+K)  Q JJL+K=(K,J)

10 CONTINUE

10 POUT(JJL+I)=ALPHA

RETURN

END
SUBROUTINE UD2SIG(U,N,SIG,TEXT,NCT)

COMPUTE STANDARD DEVIATIONS (SIGMAS) FROM U-D COVARIANCE FACTORS

U(N*(N+1)/2) INPUT VECTOR STORED ARRAY CONTAINING THE U-D
FACTORS. THE D (DIAGONAL) ELEMENTS ARE STORED
ON THE DIAGONAL

N U MATRIX DIMENSION, N.GT.1
SIG(N) VECTOR OF OUTPUT STANDARD DEVIATIONS
TEXT( ) ARRAY OF FIELD-NUMBER CHARACTERS TO BE PRINTED
NCT NUMBER OF CHARACTERS IN TEXT, 0.LE.NCT.LE.126

C


IMPLICIT DOUBLE PRECISION (A-H-O-Z)
INTEGER TEXT(1)
DIMENSION U(1), SIG(1)

J=1
SIG(1)=U(1)
DO 10 J=2,N
JJ=JJ+J
S=U(JJ)
SIG(J)=S
JM=J-1
DO 10 I=1,JM1
SIG(I)=SIG(I)+S*U(JJ+I)**2
10 CONTINUE

C WE NOW HAVE VARIANCES

DO 20 J=1,N
SIG(J)=SORT(SIG(J))
IF (NCT.EQ.0) GO TO 30
NC=NCT/6
IF (MOD(NC+6).NE.0) NC=NC+1
WRITE (6+40) (TEXT(I),I=1+NC)
WRITE (6+50) (SIG(I),I=1,N)
30 RETURN

40 FORMAT (1H0,2X,21A6)
50 FORMAT (1H0,6D18,10)
END
SUBROUTINE UTINV(RIN,N,ROUT)
C C TO INVERT AN UPPFR TRIANGULAR VECTOR STORED MATRIX AND STORE THE RESULT IN VECTOR FORM. THE ALGORITHM IS SO ARRANGED THAT THE RESULT CAN OVERWRITE THE INPUT. C C IN ADDITION TO SOLVE RX=Z SET RIN(N*(N+1)/2+1)=Z(1) , ETC., AND SET RIN((N+1)*(N+2)/2)=-1. CALL THE SUBROUTINE USING N+1 INSTEAD OF N. ON RETURN THE FIRST N ENTRIES OF COLUMN N+1 WILL CONTAIN X.
C RIN(N*(N+1)/2) INPUT VECTOR STORED UPPFR TRIANGULAR MATRIX N C ROUT(N*(N+1)/2) OUTPUT VECTOR STORED UPPFR TRIANGULAR MATRIX INVERSE C C COGNIZANT PERSONS: G.J.RIERMAN/M.W.HEAD (JPL, JAN.1978)
C C DOUBLE PRECISION RIN(1), ROUT(1), ZERO, DINV, ONE, SUM
C ZERO=0.0D0
ONE=1.0D0
C IF (RIN(1) .NE. ZERO) GO TO 5
J=1
WRITE (6,100) J,J
RETURN
C 5 ROUT(1)=ONE/RIN(1)

J=1
DO 20 J=2,N
JJOLD=JJ
JJ=JJ+J
IF (RIN(JJ) .NE. ZERO) GO TO 10
WRITE (6,100) J,J
RETURN

20 DINV=ONE/RIN(JJ)
ROUT(JJ)=DINV
II=0
IK=1
JJM1=J-1
DO 20 I=1,JJM1
II=II+I
IK=II
SUM=ZERO
DO 15 K=I,JJM1
SUM=SUM+ROUT(IK)*RIN(JJOLD+K)
15 IK=IK+K
DO 20 ROUT(JJOLD+I)=-SUM*DINV
RETURN
C
C 10 FORMAT (1HC,10X,'** MATRIX INVERSE COMPUTED ONLY UP TO BUT NOT UNTIL**
1 INCLUDING COLUMN'II',** MATRIX DIAGONAL'II', IS ZERO **')

133
SUBROUTINE UTPROW (RIN,N,POUT,NRY)

TO COMPUTE THE INVERSE OF AN UPPER TRIANGULAR (VECTOR STORED)
MATRIX WHEN THE LOWER PORTION OF THE INVERSE IS GIVEN

ON INPUT:

RIN  RXY  ROUT  *  *  WHERE  D  
   RX  RXY  *  *  *  RY  RXY

ON OUTPUT: RIN IS UNCHANGED AND ROUT=R**-1
THE RESULT CAN OVERWRITE THE INPUT (I.E., RIN=ROUT)

RIN(N*(N+1)/2)  INPUT VECTOR STORED TRIANGULAR MATRIX
N  MATRIX DIMENSION
ROUT(N*(N+1)/2)  OUTPUT VECTOR STORED MATRIX, ON INPUT THE
BOTTOM NRY ROWS CONTAIN THE LOWER PORTION
OF R**-1, ON OUTPUT ROUT=R**-1
NRY  DIMENSION OF LOWER (ALREADY INVERTED)
    TRIANGULAR R, IF NRY=0, ORDINARY MATRIX
    INVERSION RESULTS.

COGNIZANT PERSONS: G.J.RIFMAN/M.W.NEAL  (JPL MARCH 1977)

DOUBLE PRECISION  RIN(1), ROUT(1), SUM, ZERO, ONE, DINV
DATA ONE/1.000000, ZRO/0.000000/

INITIALIZATION

N=N*(N+1)/2  Q NO. ELEMENTS IN R
IRST=N-NRY  Q FIRST ROW TO BE INVERTED
IRLY=IRST+1  Q IRLY=PREVIOUS IROW INDEX
II=IRST*IRLY/2  Q II=DIAGONAL

DO 40 IROW=IRST+1,-1,0
   IF (RIN(II)+DINV*ZERO) GO TO 10
      WRITE (6,50) IROW
   RETURN

10 DINV=ONE/RIN(II)
   ROUT(II)=DINV
   KJS=NRY+IROW  Q KJ(START)
   IKS=II+IROW  Q IK(START)

C

IF (IRLY.GT.N) GO TO 35
   DO 30 J=N+IRLY-1
      KJS=KJS+1
      SUM=SUM+RIN(IK)*ROUT(KJ)
   C

30  CONTINUE
C

DO 20 K=IRLY+1
   C

20  CONTINUE
IK=IK+K

C ROUT(KJS)=SUM*INV

IRLST=IROW

II=II-IROW

RETURN

FORMAT (1HO+10X,'RIN DIAGONAL',I8,' IS ZFRO')

END
SUBROUTINE WGS (W, IMAXW, IW, JW, DW, IV, V)
MODIFIED GRAMM-SCHMIDT ALGORITHM FOR REDUCING WDW(**T)** TO U**T**W
FORM WHERE U IS A VECTOR STORED TRIANGULAR MATRIX WITH THE
RESULTING D ELEMENTS STORED ON THE DIAGONAL

W(IW,JW) INPUT MATRIX TO BE REDUCED TO TRIANGULAR FORM.
THIS MATRIX IS DESTROYED BY THE CALCULATION
IW.LE.IMAXW.AND.IW.GT.1
IMAXW ROW DIMENSION OF W MATRIX
IW NO. ROWS OF W MATRIX, DIMENSION OF U
JW NO. COLS OF W MATRIX
DW(IW) VECTOR OF NON-NEGATIVE WEIGHTS FOR THE
ORTHOGONALIZATION PROCESS. THE DI'S ARE UNCHANGED
BY THE CALCULATION.
U(IW*(IW+1)/2) OUTPUT UPPER TRIANGULAR VECTOR STORED OUTPUT
V(JW) WORK VECTOR

(SEE BOOK:
'FACTORIZATION METHODS FOR DISCRETE SEQUENTIAL ESTIMATION',
BY G.J.RIFRMAN)

ESTIMATION

COGNIZANT PERSONS: G.J.BIERMAN/M.W.MHEAD (JPL, FEB.1979)

IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DOUBLE PRECISION SUM,Z, DINV
DIMENSION W(IMAXW+1), DW(1), U(1), V(1)

Z=0. DO
ONE=1 .DO
IWP2=IW+2
DO 100 L=2,IW
J=IWP2-L
SUM=Z
DO 40 K=1,JW
V(K)=W(J,K)
U(K)=DW(K)*U(K)
SUM=V(K)*U(K)+SUM
40 W(J+K)=SUM
DINV=SUM
JM=J+1
IF (SUM.GT.Z) GO TO 45
C W(J,K)=0. WHEN DINV=0 (DINV=NORM(W(J,K)**2))
DO 44 K=1,JM1
44 W(J+K)=Z
GO TO 100
45 DO 50 K=1,JM1
SUM=Z
50 DO 50 I=1,JW
SUM=W(K+I)*U(I)+SUM
SUM=SUM/DINV
C DIVIDE HERE (IN PLACE OF RECIPROCAL MULTIPLY) TO AVOID
C POSSIBLE OVERFLOW
C
DO 60 I=1,JW
   W(K+I)=W(K+I)-SUM*V(I)
60    W(J+K)=SUM
100   CONTINUE
C END EQ.(4.10) OF BOOK
C U(K,J) STORED IN W(J,K)
C THE LOWER PART OF W IS U TRANSPOSE
C
SUM=2
DO 105 K=1,JW
105   SUM=SUM+W(K)*W(1+K)**2+SUM
U(1)=SUM
IJ=1
DO 110 J=2,JW
   DO 110 I=1,J
      IJ=IJ+1
110   U(IJ)=W(J+I)
C RETURN
END