November 12, 1997

DRAFT

Constrained Least Squares and Error Propagation

V. Blobel

Abstract

Constrained fits and error propagation are standard tasks in data analysis. The basic equations for both task are given. Subprograms for the two closely related tasks are explained, which require only a minimum of coding in given applications.

II. Institut für Experimentalphysik

Hamburg 1997

1 Error Propagation and Constrained Fits

Error propagation and constrained fits are standard tasks in data analysis. An example for error propagation is the transformation from cartesian to polar coordinates, and an example for constrained fits are the application of energy and momentum conservation to particle reactions. The programs ERRPRP and APLCON described here allow to perform these tasks with a minimum amount of coding.

Measured parameters are denoted by a vector $\boldsymbol{x}$, it has n components $x_1, x_2 \ldots x_n$; the covariance matrix of the parameter vector, which is of special importance for both applications, is denoted by $\boldsymbol{V}_x$:

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ \ldots \\ x_n \end{pmatrix} \qquad \boldsymbol{V}_x = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \ldots & \sigma_{1n} \\ \sigma_{21} & \sigma_{22} & \ldots & \sigma_{2n} \\ \ldots & \ldots & & \ldots \\ \ldots & \ldots & & \ldots \\ \sigma_{n1} & \sigma_{n2} & \ldots & \sigma_{nn} \end{pmatrix}$$

The diagonal element σ_{ii} of the matrix $\boldsymbol{V}_x$ is the *variance* or the square of the *standard deviation* σ_i of the component x_i: $\sigma_{ii} = {\sigma_1}^2$. The element σ_{ij} of the covariance matrix $\boldsymbol{V}_x$ can be written as $\sigma_{ij} = \rho_{ij}\sigma_i\sigma_j$, where ρ_i is the *correlation coefficient* between the components x_i and x_j and can take on values between -1 and $+1$.

The covariance matrix is a symmetric matrix: $\boldsymbol{V}_x^T = \boldsymbol{V}_x$ (i.e. with $\sigma_{ij} = \sigma_{ji}$). The programs make use of the symmetry property of the matrix in order to save space and computation time. Only the elements on and below the diagonal are used,

$$\boldsymbol{V}_x = \begin{pmatrix} \sigma_{11} & & & \\ \sigma_{21} & \sigma_{22} & & \\ \ldots & \ldots & & \\ \ldots & \ldots & & \\ \sigma_{n1} & \sigma_{n2} & \ldots & \sigma_{nn} \end{pmatrix} = \begin{pmatrix} \texttt{VX(1)} & & & \\ \texttt{VX(2)} & \texttt{VX(3)} & & \\ \ldots & \ldots & & \\ \ldots & \ldots & & \\ \texttt{VX(NS-N+1)} & \texttt{VX(NS-N+2)} & \ldots & \texttt{VX(NS)} \end{pmatrix}$$

and only these elements are stored (row-wise) in the array `VX` containing the covariance matrix. For example the element `VX(3)` of the array contains σ_{22} and the element `VX(5)` contains σ_{23}.

Both tasks require the derivative matrix of m quantities with respect to the n components x_i. The programs can calculate all derivatives by numerical methods, with a symmetric formula. The step size used in the calculation is determined automatically from the covariance matrix. Derivative $m \times n$ matrices,

$$\boldsymbol{A} = \begin{pmatrix} A_{11} & A_{12} & A_{13} & \ldots & A_{1n} \\ A_{21} & A_{22} & A_{23} & \ldots & A_{2n} \\ A_{31} & A_{32} & A_{33} & \ldots & A_{3n} \\ \ldots & \ldots & \ldots & & \ldots \\ A_{m1} & A_{m2} & A_{m3} & \ldots & A_{mn} \end{pmatrix}$$

required by the program are stored also row-wise in the array `A`. For example the array-element `A(3)` contains the matrix element A_{13} and for $n = 10$ the array element `A(13)` contains the matrix element A_{23}.

2 Error Propagation

2.1 Formulas

First a linear transformation of the vector of parameters $\boldsymbol{x}$ to a vector of parameters $\boldsymbol{y}$ of m elements is considered:

$$\boldsymbol{y} = \boldsymbol{y}_0 + \boldsymbol{A}\boldsymbol{x}$$

The covariance matrix $\boldsymbol{V}_y$ for the transformed parameters $\boldsymbol{y}$ is

$$\boldsymbol{V}_y = \boldsymbol{A}\boldsymbol{V}_x\boldsymbol{A}^T$$

by the law of error propagation.
If the transformation from $\boldsymbol{x}$ to $\boldsymbol{y}$ is nonlinear, then the covariance matrix $\boldsymbol{V}_y$ of $\boldsymbol{y}$ can only be calculated in an approximation. If $\boldsymbol{A}$ is the derivative matrix of the parameters $\boldsymbol{y}$ with respect to the parameters $\boldsymbol{x}$, with elements (partial derivatives)

$$A_{ij} = \frac{\partial y_i}{\partial x_j} \ ,$$

then the approximate covariance matrix $\boldsymbol{V}_y$ is $\boldsymbol{V}_y = \boldsymbol{A}\boldsymbol{V}_x\boldsymbol{A}^T$ as above.

2.2 Program for Error Propagation

The input to the problem is

$$\begin{aligned} \boldsymbol{x} &= \text{vector of measured parameters} = \text{array } \mathtt{X} \\ \boldsymbol{V}_x &= \text{covariance matrix for } x = \text{array } \mathtt{VX} \end{aligned}$$

and the result is

$$\begin{aligned} \boldsymbol{y} &= \text{vector of measured parameters} = \text{array } \mathtt{Y} \\ \boldsymbol{V}_y &= \text{covariance matrix for } y = \text{array } \mathtt{VY} \ . \end{aligned}$$

Numerical derivatives. The values `NX` and `MY` are the number of measured and of transformed parameters, respectively. The formulas for the transformation to the components y_j of the transformed vector $\boldsymbol{y}$ have to be given in the program, which formally is given by the code:

```
   CALL SIMNIT(NX,MY,IDEBUG,EPSILON)
   X  = parameter values
   VX = covariance matrix
10 Y( 1) = function of X
   ...
   Y(MY) = function of X
   CALL ERRPRP(X,VX,Y,VY,IRET)
   IF(IRET.LT.0) GOTO 10
```

The call to `SIMNIT` initializes the package; the first two parameters are the two dimension parameters of the problem. The integer `IDEBUG` defines the debug level (=0 for minimum printout, 3 = for development). The argument `EPSILON` is presently not used in error propagation.
The statements between statement number 10 and the `IF`-statement are repeatedly executed, to calculate the derivative matrix, which then is used for the error propagation. When the loop between statement 10 and the `IF`-statement is left (for `IRET` ≥ 0), the array `VY` will contain the covariance matrix $\boldsymbol{V}_y$ of the transformed vector $\boldsymbol{y}$.
Example. Transformation from cartesian to polar coordinates in two dimensions
The parameters x_1 and x_2 are cartesian coordinates, which have to be transformed to the parameters

$$y_1 = \sqrt{x_1^2 + x_2^2} \qquad y_2 = \tan\left(\frac{x_2}{x_1}\right) \ .$$

The input covariance matrix is diagonal:

$$\boldsymbol{V}_x = \begin{pmatrix} 0.5 & 0 \\ 0 & 2.0 \end{pmatrix} \ .$$

The numbers given correspond to the values

$$x_1 = 10.0 \pm 0.7 \qquad x_2 = 5.0 \pm 1.4$$

The following code will perform the transformation:

```
      REAL X(2),VX(3),Y(2),VY(3)
      DATA  X / 10.0 , 5.0 /
      DATA VX /  0.5 , 0.0 , 2.0 /
      CALL SIMDIM(2,2,3,0.001)
   10 Y(1) = SQRT(X(1)**2 + X(2)**2)
      Y(2) = ATAN2(X(2),X(1))
      CALL ERRPRP(X,VX,Y,VY,IRET)
      IF(IRET.LT.0) GOTO 10
```

The result is

$$y_1 = 11.18 \pm 0.53 \qquad y_2 = 0.46 \pm 0.08$$

with a correlation coefficient of $\rho = 0.75$ between y_1 and y_2. The full covariance matrix is

$$\boldsymbol{V}_y = \begin{pmatrix} 0.2796 & 0.0321 \\ 0.0321 & 0.0066 \end{pmatrix} .$$

3 Constrained Least Squares

3.1 The Lagrange Multiplier Method

Assume that ideally the parameters $\boldsymbol{x}$ have to satisfy one or more equations

$$f(\boldsymbol{x}) = 0 \qquad \text{(constraint equation)} .$$

Since in general the measured values, denoted by $\boldsymbol{x}_0$, will not satisfy the constraint equations exactly, corrections $\Delta\boldsymbol{x}$ can be calculated, such that the corrected vector $\boldsymbol{x}$, or $\boldsymbol{x}_0 + \Delta\boldsymbol{x}$, satisfies the constraint equations

$$f(\boldsymbol{x}_0 + \Delta\boldsymbol{x}) = 0$$

exactly or at least within round-off errors.
In the method of least squares, the corrections $\Delta\boldsymbol{x}$ have to satisfy the condition

$$\Delta\boldsymbol{x}^T \boldsymbol{V}_x^{-1} \Delta\boldsymbol{x} = \text{minimum} ,$$

where $\boldsymbol{V}_x^{-1}$ is the inverse covariance matrix. If the covariance matrix is a diagonal matrix with elements σ_1^2, σ_2^2 ... σ_n^2 on the diagonal, the sum can be written in this special case as

$$\sum_{i=1}^{n} \left(\frac{\Delta x_i}{\sigma_i} \right)^2 = \text{minimum} .$$

The problem to determine corrections $\Delta\boldsymbol{x}$ under the above conditions is treated using the Lagrange method, where additional parameters, the Lagrangian multipliers λ, are introduced, one for each constraint equation. In the method one has to find a stationary point of the function $Q(\Delta\boldsymbol{x}, \lambda)$,

$$Q(\Delta\boldsymbol{x}, \lambda) = \Delta\boldsymbol{x}^T \boldsymbol{V}_x^{-1} \Delta\boldsymbol{x} + 2\lambda^T f(\boldsymbol{x}_0 + \Delta\boldsymbol{x}) .$$

At a stationary point of this function the derivatives of Q with respect to all $\Delta\boldsymbol{x}_i$ and all λ_j have to vanish (necessary condition).
Usually the constraint equations depend nonlinearly on the components x_i of the parameter vector $\boldsymbol{x}$. Then the problem has to be solved by an iterative method, calculating a sequence of corrections $\Delta\boldsymbol{x}$, which has to converge to the final correction vector. In each iteration the condition equations are approximated by linear equations

$$f_p + \boldsymbol{A}(\Delta\boldsymbol{x} - \Delta\boldsymbol{x}_p) = 0 ,$$

where $\Delta\boldsymbol{x}_p$ is the correction calculated in the previous iteration (initially $\Delta\boldsymbol{x}_p$ =0), f_p are the function values at the point $\boldsymbol{x}_0 + \Delta\boldsymbol{x}_p$ and $\Delta\boldsymbol{x}$ is the new correction, to be calculated. The derivative matrix

$\boldsymbol{A}$ at the point $\boldsymbol{x}_0 + \Delta\boldsymbol{x}$ is given by

$$\boldsymbol{A} = \begin{pmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 & \dots & \partial f_1/\partial x_n \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 & \dots & \partial f_2/\partial x_n \\ \dots & \dots & & \dots \\ \partial f_m/\partial x_1 & \partial f_m/\partial x_2 & \dots & \partial f_m/\partial x_n \end{pmatrix}_p ,$$

where the index p indicates, that the derivatives are calculated at the position of the previous value $\boldsymbol{x}_0 + \Delta\boldsymbol{x}_p$. In the following the inverse $\boldsymbol{V}_x^{-1} = \boldsymbol{W}$ is called the weight matrix. The problem to find a stationary point of the Lagrange equations is equivalent to the simultaneous solution of the following two equations:

$$\begin{array}{lcccl} \boldsymbol{W}\Delta\boldsymbol{x} & + & \boldsymbol{A}^T\lambda & = & 0 \\ \boldsymbol{A}\Delta\boldsymbol{x} & & & = & -\boldsymbol{f}_p + \boldsymbol{A}^T\Delta\boldsymbol{x}_p \end{array} .$$

The two equations above can be considered as one matrix equation

$$\begin{pmatrix} \boldsymbol{W} & \boldsymbol{A}^T \\ \boldsymbol{A} & 0 \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{x} \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ -\boldsymbol{f}_p + \boldsymbol{A}\Delta\boldsymbol{x}_p \end{pmatrix} .$$

If this equation will be solved by matrix inversion, the $n \times n$ submatrix in the inverted matrix at the position of $\boldsymbol{W}$ in the above equation will be the covariance matrix of the fitted parameters values. After the final iteration one can also calculate the quantity

$$c = \Delta\boldsymbol{x}^T \boldsymbol{V}_x^{-1} \Delta\boldsymbol{x} ,$$

which follows a χ^2 distribution with m degrees of freedom, if m is the number of constraint equations.
Unmeasured Parameters. The case, where one or more of the components x_i are unmeasured, can be treated with the same formalism. In this case the user has to determine initial approximations x_{0i} for the unmeasured components x_i, which is usually posssible from selected constraints. To specify the unmeasured components x_i, which by definition have no errors defined, the following convention is used in this program:

the elements in $\boldsymbol{V}_x$ corresponding to unmeasured parameters have to be zero.

Then, internally, the weight matrix $\boldsymbol{W}$ may be obtained by inverting the portion of $\boldsymbol{V}_x$, corresponding to measured parameters, leaving the elements corresponding to unmeasured parameters unchanged (zero value). Interpreted as a weight matrix, the weight matrix elements corresponding to unmeasured parameters are zero. Having defined the weight matrix $\boldsymbol{W}$ in this way, the equations above can be used without modification, to determine the corrections $\Delta\boldsymbol{x}$.

In the program the weight matrix is not formed explicitly; instead a faster, but completely equivalent algorithm is used for the solution of the matrix equation, the calculation of the covariance matrix of the fitted parameters and of the test value c.

Derivatives. In both the error propagation and the constrained fit problem derivative matrices are required, which can have many elements. The programs by default calculate all elements by numerical methods, which makes the application of the program simple. The method requires the use of reasonable step sizes for the numerical derivative, which are determined from the diagonal elements of the covariance matrix. The numerical derivatives are usually sufficiently accurate. If the program is applied to a large number of cases, the use of analytical formulas will increase the speed of the program. The program will automatically check the analytical derivatives by comparison to the numerical values in the first case treated by the program; this feature may help in finding mistakes in the coding of the many analytical derivatives.

The input to the problem is

$$\begin{array}{rcl} \boldsymbol{x}_0 & = & \text{vector of measured parameters} = \text{array } \mathtt{x} \\ \boldsymbol{V}_{x_0} & = & \text{covariance matrix for } \boldsymbol{x}_0 = \text{array } \mathtt{VX} \end{array}$$

and the m constraint equations, and the result is

$$\boldsymbol{x} = \text{vector of fitted parameters} = \text{array } \mathtt{X}$$
$$\boldsymbol{V}_x = \text{covariance matrix for } \boldsymbol{x} = \text{array } \mathtt{VX}$$

The same arrays are used for the the measured and fitted quantities.
In the simplest application with numerical derivatives the code for the constrained least squares fit is:

```
   CALL SIMNIT(NX,MF,IDEBUG,EPSILON)
   X  = parameter values
   VX = covariance matrix
10 F( 1) = function of X
   ...
   F(NF) = function of X
   CALL APLCON(X,VX,F,IRET)
   IF(IRET.LT.0) GOTO 10
```

The code is explained in the next section.

3.2 Program for the Constrained Fit

Given a parameter vector $\boldsymbol{x}$ of dimension $n_x =$ NX and its covariance matrix $\boldsymbol{V}_x$, and NF constraint equations $f_j(\boldsymbol{x}) = 0$, the code has be written according to the following scheme.
Initialization. In the first statement the dimension parameters have to be defined:

```
CALL SIMNIT(NX,NF,IDEBUG,EPSILON)
```

Restriction because of limited dimensions of internal arrays are: NX ≤ 50 and NF ≤ 20. Larger values require to change PARAMETER-statements in the subprograms.
Printout for debugging is controlled by the argument IDEBUG, with the following values:

```
IDEBUG = -1   no printout
          0    only (limited number of) warnings are printed
          1    minimal printout
          2    more printout
          3    full printout of input and result
        > 3    even more printout
```

For initial tests it is recommended to use the value of 3.
The argument EPSILON specifies the required precision of the constraints at convergence. The program will continue with iterations, until the average value of the constraint equations is below EPSILON. A too small value of EPSILON may cause problems due to round-off errors and may require many iterations.
Input parameters. Now the variables X(1) ... X(NX) have to be defined and their covariance matrix elements V(1) .... Variables may be measured ones or unmeasured ones. Unmeasured variables are characterized by zero elements of the covariance matrix (at least the corresponding diagonal element has to be zero). For unmeasured variables the value of the element $\boldsymbol{x}_i$ has to be at some reasonable initial value. In addition it is necessary to define a step size for numerical differentiation for the *unmeasured* variables. The call to define step ST for variable X(I) is

```
CALL SIMSTP(I,ST)
```

For *measured* values the necessary step size is taken from the standard deviation in the covariance matrix; usually this works satisfactory. In special cases a parameter may be ill-defined and this could cause step sizes, which are too large. In this case it is recommended to specify a step size for the measured variable too; this is then taken as the upper limit of the actually used step size.
The user may optionally define limits for physical regions of variable $x_i =$ X(I) by

```
CALL SIMLIM(I,XLOW,XHIG)
```

This is recommended especially, if the constrained equations become meaningless or formally wrong (for example with negative square root) for a certain range of parameter values.

Fit loop. The following coding example represents the loop, which performs the constrained fit. During the loop the variables `X(1) ... X(NX)` are modified, to perform the numerical differentiation and to apply corrections during the fit. The user has to supply the code to calculate the constraint equations `F(1) ... F(NF)`. Finally (at convergence) the covariance matrix is modified to the matrix for the fitted values, which (for sufficient constraints) has nonzero elements for previously unmeasured variables.

```
10 F(1)=function of X(1) ... X(NX)
   ...
   F(NF) = function of X(1) ... X(NX)
   CALL APLCON(X,VX,F,IRET)
   IF(IRET.LT.0) GOTO 10

   IRET = 0    ! convergence reached
   IRET = 1    ! no convergence
```

Non-convergence is assumed for

- chisquare above 4 standard deviations for first 10 iterations
- chisquare above 3 standard deviations for next 10 iterations or more than 20 iterations

The conditions are defined for correctly assigned covariance matrices. The user may scale up his covariance matrix to force more cases with convergence.

Derivatives. By default all the necessary `NX` $\times$ `NF` derivatives are calculated numerically. Usually this works rather well, although for analytically calculated derivatives the precision is higher and execution time is reduced; in certain cases the calculation of numerical derivatives may fail.

The user may calculate the elements in his program. The derivative matrix is in array `A` in

```
COMMON/MATCOM/A(.)
```

with the definition

$$\frac{\partial f_j}{\partial x_i} = \texttt{A(I+NX*(J-1))}$$

and has to be defined by the user before the call of `APLCON`. In the first iteration of the first case the program will automatically compare the elements with values calculated numerically and will printout elements with disagreement.

Example: Reconstruction of π^0 from the decay photons

The package includes the example program APLCTS, which performs the reconstruction of aπ^0 from the decay photons.

Two photons from a π^0 decay are measured. The invariant mass of the two photons is fitted to the mass of the π^0 of 0.135 GeV. For each photon there are three measured parameters:

$$\begin{aligned} E &= \text{energy in GeV} \\ \vartheta &= \text{polar angle} \\ \varphi &= \text{azimutal angle .} \end{aligned}$$

Thus the components of the photon four-vector are

$$P = \begin{pmatrix} p_x \\ p_y \\ p_z \\ E \end{pmatrix} = \begin{pmatrix} E \sin\vartheta \cos\varphi \\ E \sin\vartheta \sin\varphi \\ E \cos\vartheta \\ E \end{pmatrix} .$$

The invariant mass squared of the sum of the two four-vectors $(P_1 + P_2)^2$ is fitted to $m^2_{\pi^0}$.

```
      SUBROUTINE APLCTS
*     example program for APLCON:
*        fit two photons to pi zero mass
*
      REAL A(3),B(3),VA(6),VB(6),X(6),VX(21),F(1)
*            first photon:  E phi theta
      DATA A/ 1.0 , 0.1 , 0.5 /
*            second photon: E phi theta
      DATA B/ 2.0 , 0.2 , 0.4 /
*            covariance matrices
      DATA VA/0.0001 , 0.0 , 0.00002 , 0.0 , 0.0 , 0.00001 /
      DATA VB/0.0002 , 0.0 , 0.00002 , 0.0 , 0.0 , 0.00001 /
*     ...
      WRITE(*,*) 'Test of constrained fit'
*     x = array of measured values
      DO 10 I=1,3
      X(I  ) = A(I)
   10 X(I+3) = B(I)
*     vx = joint covariance matrix
      DO 12 I=1,21
   12 VX(I)=0.0
      CALL SMTOS(VA,1,VX,1,3)
      CALL SMTOS(VB,1,VX,4,3)
*     6 parameters and 1 constraint equations
      CALL SIMNIT(6,1,3,1.E-3)
*     add four vectors
   20 EN = X(1) + X(4)
      PX = X(1)*SIN(X(2))*COS(X(3)) + X(4)*SIN(X(5))*COS(X(6))
      PY = X(1)*SIN(X(2))*SIN(X(3)) + X(4)*SIN(X(5))*SIN(X(6))
      PZ = X(1)*COS(X(2))           + X(4)*COS(X(5))
*     mass squared by square of four vector
      FM2  = EN**2 - PX**2 - PY**2 - PZ**2
*     constraint = calculated mass**2 - pi zero mass**2
      F(1) = FM2 - 0.135**2
*     call fit program
      CALL APLCON(X,VX,F,IREP)
      IF(IREP.LT.0) GOTO 20
*     print fitted mass value
      WRITE(6,101) SQRT(FM2)
  101 FORMAT(' Fitted mass value is',F7.4,' GeV')
      END
```

χ^2 of the fit and pulls. The result of the fit and convergence depends on the correct input of the parameters and the covariance matrix. If everything is correct, then the χ^2 of the fit should follow the theoretical χ^2-distribution with `NDEG` degrees of freedom. These two values are obtained after the fit by the following call:

```
      CALL SIMNCH(NDEG,CHISQ)
*     NDEG  = number of degrees of freedom
*     CHISQ = chi square value
```

The distribution of the corresponding probability P (obtained via `P = PROB(CHISQ,NDEG)`, `PROB` is a CERNLIB function) should be nearly uniform, perhaps with an additional enhancement at `P=0`, from wrong hypotheses. It is recommended to check the individual pull distributions of the parameters. The pull p_i of the parameter x_i is defined by

$$p_i = \frac{\Delta \boldsymbol{x}_i}{\sqrt{\boldsymbol{V}_{ii}^{\text{unf}} - \boldsymbol{V}_{ii}^{\text{fit}}}}$$

where the two matrices $\boldsymbol{A}$ are the covariance matrices before and after the fit. After the convergence of the fit the pull for variable i can be found in the element `A(I)` of the array `A` in common `MATCOM`. The distribution of all pulls (for the correct hypothesis) should ideally follow the standardized normal distribution (mean of zero and standard deviation of one). The interpretation of *deviations* from this ideal behaviour is not so simple: if only one input standard deviation is wrong, then *all* pulls will be influenced, because *all* deviations are distributed on average equally to *all* variables in a least squares fit.

3.3 Utility subprograms

Subprogram `SMPRTV` can be used for a printout of a set of measured parameters $\boldsymbol{x} = $ `X` of dimension $n = $ `NX` and the covariance matrix $\boldsymbol{V}_x = $ `VX`. The call is:

```
CALL SMPRTV(X,VX,NX)
```

The values of parameters x_i and their standard deviations σ_i, calculated from the square root of the diagonal element σ_{ii} of the matrix $\boldsymbol{V}_x$, are printed. In addition the matrix of correlation coeffients ρ_{ij} is printed, which are defined by

$$\rho_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}}$$

and which satisfy the condition $-1 \le \rho_{ij} \le +1$. Values $|\rho_{ij}| = 1$ mean a complete correlation between the parameters x_i and x_j. The correlation coeffients ρ_{ij} represent in a direct way the size of the correlations between the parameters.

Subprogram `SMTOS` can be used to copy all elements of a symmetric matrices or of subsets to another symmetric matrix. The symmetric storage mode is used, with total $n(n+1)/2$ elements for a $n \times n$ symmetrix matrix. The calling sequence is:

```
CALL SMTOS(VA,IA,VX,IX,N)
```

The program copies a $N \times N$ matrix `VA`, starting at index `IA=1`, or a $N \times N$ submatrix of the matrix `VA`, starting at index `IA`, to the matrix `VX`, starting at index `IX`. For complete matrices `IA=1` and `IX=1`. Subroutine `SMTOS` is used in the example above.