Least Squares in Newstar

Contributed by W.N. Brouw, 15 April, 1995 — Version 3

Contents

1 Introduction

Tryimg to find a more stable non-linear least squares method (*LSQ*) than the one currently used in Newstar, I found another few areas that could be added or improved in the existing routines. In the end I made the following changes:

- improved error estimates for solved variables
- included support for user defined constraints
- support for more stable non-linear solutions
- changed the user interface to routines (they are now called WNML... rather than WNMI... , WNMY... , WNMX... or WNMZ...)

In addition the present document describes the usage and background of the different routines.

Least squares solutions in Newstar have the following general background:

1. all are based on creating normal equations from an, initially unknown, number of condition equations

2. the actual *LSQ*-object is a symmetric (or Hermitian) matrix
3. matrix inversions are done using in-place Cholesky methods where possible: it is for standard use the fastest, least memory consuming and a stable method. In cases where the inverse matrix is needed, the method uses resources of the same order as full LU-decomposition.
4. input/output to the *LSQ*-object is done in single precision, internally all calculations are done in double precision
5. all routines using the *LSQ* package should include the LSQ_O_DEF file
6. complex numbers are assumed to be contiguous pairs of real numbers with the real part being the first

The following terminology is used throughout:

- n number of unknowns to be solved
- m number of simultaneous knowns
- N number of condition equations
- χ^2 merit function
- z^* complex conjugate of z
- $z_{\Re}$ real part of z
- $z_{\Im}$ imaginary part of z
- x column vector of unknowns $x_0, \ldots, x_{n-1}$
- a_i vector of factors $a_{0,i}, \ldots, a_{n-1,i}$ in condition equation i ($i = 0, \ldots, N-1$)
- C the $N \times n$ array of condition equations
- A $n \times n$ array: normal equations matrix
- L n column vector: right-hand side normal equations
- y_i model value for condition equation i ($i = 0, \ldots, N-1$)
- l_i measured value for condition equation i ($i = 0, \ldots, N-1$)
- σ_i standard deviation for condition equation i ($i = 0, \ldots, N-1$)
- w_i weight for condition equation i ($w_i = 1/\sigma_i^2$)
- $[ab]$ shorthand for $\sum_{i=0}^{N-1} w_i a_i b_i$

The condition equations can, with the above definitions, be written in one of the following ways:

$$a_i \cdot x = l_i \qquad i = 0, \ldots, N-1 \tag{1}$$

or as:

$$\sum_{k=0}^{n-1} a_{ik} x_k = l_i \qquad i = 0, \ldots, N-1 \tag{2}$$

or as:

$$C \cdot x = l \tag{3}$$

The normal matrix is defined as:

$$A = C^T \cdot Q^{-1} \cdot C \tag{4}$$

resulting in the normal equations:

$$A \cdot x = L \tag{5}$$

where $L = C^T \cdot l$ and Q is the covariance matrix of the observations. In Newstar only covariance matrices with the same value in each column i (the 'weight' w_i) are considered.

2 Linear equations

2.1 Real

The merit function we want to minimise is:

$$\chi^2 = \sum_{i=0}^{N-1} \left[\frac{l_i - y_i}{\sigma_i}\right]^2 \tag{6}$$

For the minimum of χ^2 holds:

$$\frac{\partial \chi^2}{\partial a_k} = 0 \qquad\qquad k = 0, \ldots, n-1 \tag{7}$$

which leads to the following set of normal equations:

$$\sum_{i=0}^{N-1} \frac{[l_i - y_i]}{\sigma_i^2} \frac{\partial y_i}{\partial a_k} = 0 \qquad\qquad k = 0, \ldots, n-1 \tag{8}$$

or in matrix form:

$$\begin{pmatrix} [a_0 a_0] & [a_0 a_1] & \cdots & [a_0 a_{n-1}] \\ [a_1 a_0] & [a_1 a_1] & \cdots & [a_1 a_{n-1}] \\ \vdots & \vdots & \ddots & \vdots \\ [a_{n-1} a_0] & [a_{n-1} a_1] & \cdots & [a_{n-1} a_{n-1}] \end{pmatrix} \cdot x = \begin{pmatrix} [a_0 l] \\ [a_1 l] \\ \vdots \\ [a_{n-1} l] \end{pmatrix} \tag{9}$$

or:

$$A \cdot x = L \tag{10}$$

This symmetric set of equations can be solved, if the matrix is positive definite, i.e. if there are no dependencies between the columns. The solution is given by:

$$x = A^{-1} \cdot L \tag{11}$$

2.1.1 Errors

After solution for the unknowns x, χ^2 as defined in equation 6 can be directly calculated if we rewrite it as:

$$\chi^2 = [ll] - 2\sum_{k=0}^{n-1}[a_k l]x_k + \sum_{i=0}^{N-1} w_i y_i^2 \tag{12}$$

Noting that $y_i = \sum_{k=0}^{n-1} a_k x_k$ and using equation equation 9 to note that $[a_i l] = \sum_{k=0}^{n-1}[a_i a_k]x_k$, we can rewrite equation 12 as:

$$\chi^2 = [ll] - \sum_{k=0}^{n-1} x_k [a_k l] \tag{13}$$

The χ^2 could be used to assess the goodness of fit if the actual σ_i's were known, and the errors are normal distributed. In general the actual values of σ_i are not known, and often the distribution is not normal (.e.g. if we solve for logarithmic values). An estimate of the standard deviation can be made by:

$$\sigma^2 = \frac{\left[(l_i - y_i)^2\right]}{N-n} \tag{14}$$

which can be estimated by:

$$\sigma_o^2 = \frac{\chi^2}{N-n} \tag{15}$$

to give 'an error per observation'. The 'error per unit weight', or the standard deviation, can be expressed as:

$$\sigma_w^2 = \frac{\chi^2}{[1]} \frac{N}{N-n} \tag{16}$$

The uncertainty in the solution x_i can be expressed as:

$$\sigma^2(x_i) = \sum_{k=1}^{N-1} \sigma_k^2 \left(\frac{\partial x_i}{\partial y_k}\right)^2 \tag{17}$$

Since

$$x_i = \sum_{k=0}^{n-1} \left(A^{-1}\right)_{ik} [a_k l] \tag{18}$$

we have:

$$\frac{\partial x_i}{\partial y_k} = \sum_{j=0}^{n-1} \left(A^{-1}\right)_{ij} \frac{a_{kj}}{\sigma_k^2} \tag{19}$$

leading to:

$$\sigma^2(x_i) = \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} \left(A^{-1}\right)_{ik} \left(A^{-1}\right)_{il} \left[\sum_{j=0}^{N-1} \frac{a_{kj} a_{lj}}{\sigma_j^2}\right] \tag{20}$$

Doing the sums, this equation reduces to:

$$\sigma^2(x_i) = \left(A^{-1}\right)_{ii} \tag{21}$$

If the σ_i was not known originally, the estimate of the standard uncertainties in the unknowns x are:

$$\sigma(x_i) = \sigma_o \sqrt{\left(A^{-1}\right)_{ii}} \tag{22}$$

If the uncertainties in the unknowns are wanted, Newstar calculates the inverse matrix A^{-1} by backsubstitution of the identity matrix, and the uncertainties by equation 22 .

2.1.2 Newstar calls

Real *LSQ* in Newstar is implemented by the following calls:

1. generate an *LSQ*-object with:
   LOGICAL WNMLGA(*LSQ*, *type*, *n* [, *m*]) with:

   LSQ INTEGER, return pointer to created *LSQ*-object (if *.true.*)
   type coded INTEGER; coded as: 0 or LSQ_T_REAL [+ LSQ_T_MULTIPLE]. The second part indicates that multiple ($m > 1$) simultaneous equations are requested.
   n INTEGER, number of unknowns
   m INTEGER, number of simultaneous equations if LSQ_T_MULTIPLE specified (default one)
2. generate normal equations from the condition equations with successive calls to:
   CALL WNMLMN(*LSQ*, *type*, *condition*, *weight*, *measured*) with:

   LSQ *LSQ*-object pointer
   type 0 or LSQ_C_REAL to indicate that the condition equations are real
   condition factors in condition equation as REAL$(0 : n-1)$
   weight REAL, weight of condition equation ($w = 1/\sigma^2$). Note that for the best arithmetic performance, the w_i are best normalised to a maximum value of one.
   measured measured value(s) as REAL$(0 : m-1)$
3. triangularise the normal equations matrix A with:
   LOGICAL WNMLTN(*LSQ*) with:

LSQ *LSQ*-object pointer

The value of WNMLTN is *.false.* if the matrix A is not invertable. Note that if an unknown does not appear in any condition equation with a non-zero coefficient (i.e. if the corresponding $A_{ii} \equiv 0$) WNMLTN assumes a zero solution for the corresponding unknown. The check on dependency between equations is done with a default precision of 10^{-6}. This precision can be changed (sec. 2.1.3)).

4. obtain the solution by back substitution with:
   CALL WNMLSN(*LSQ*,*solution*,*error*,*sd*) with:

LSQ *LSQ*-object pointer
solution solution x as REAL$(0:n-1, 0:m-1)$
error adjustment error per unit weight (σ_w)as REAL$(0:m-1)$
sd standard deviation (σ_o) as REAL$(0:m-1)$

5. obtain the covariance matrix or something similar with:
   CALL WNMLME(*LSQ*,*uncertainties*) or:
   CALL WNMLCV(*LSQ*,*covariance*) or:
   CALL WNMLIN(*LSQ*) with:

LSQ *LSQ*-object pointer
uncertainties the uncertainties in the unknowns x as REAL$(0:n-1, 0:m-1)$
covariance the covariance (inverse) matrix of the solution, as REAL$(0:n-1, 0:n-1)$

WNMLIN will calculate the covariance matrix internally only. In all three cases the covariance matrix will overwrite the normal array, and it will be used, rather than back substitution, in WNMLSN if that routine is called after one of the inversion ones.

6. free *LSQ*-object with:
   CALL WNMLFA(*LSQ*)

2.1.3 Additional Newstar calls

A number of other calls are available to aid, especially, re-use of the *LSQ*-object:

- re-initialise the *LSQ*-object for re-use with:
  CALL WNMLIA(*LSQ*, *type* [, *collinearity*]) with:

LSQ *LSQ*-object pointer
type coded INTEGER, coded as a combination of:

  - 0: default, equal to LSQ_I_ALL
  - LSQ_I_NORM: re-initialise the 'unknown' part of the object (i.e. the normal array A)
  - LSQ_I_KNOWN: re-initialise the 'known' part of the object (i.e. the vector L)
  - LSQ_I_SOL: equal to NORM + KNOWN
  - LSQ_I_NONLIN: re-initialise the non-linear part of the object
  - LSQ_I_ALL: equal to SOL + NONLIN
  - LSQ_I_PREC: set the internal check precision to *collinearity*

collinearity precision factor to be used (REAL). Only used if LSQ_I_PREC set. Note that the default internal value is 10^{-6}.

- re-create (parts of) the normal equations with successive calls to:
  CALL WNMLMN(*LSQ*, *type*, *condition*, *weight*, *measured*) with:

LSQ *LSQ*-object pointer
type the standard LSQ_C_REAL etc can be used, with in addition one of the following:

  - LSQ_C_NONORM: do not update the normal array (A)
  - LSQ_C_NOKNOWN: do not update the 'known' part of the normal equations (L)

condition factors in condition equation as REAL$(0:n-1)$
weight REAL, weight of condition equation ($w = 1/\sigma^2$) Note that for the best arithmetic performance, the w_i are best normalised to a maximum value of one.
measured measured value(s) as REAL$(0:m-1)$

2.2 Complex

The merit function we want to minimise in this case is:

$$\chi^2 = \sum_{i=0}^{N-1} \left[\frac{l_i - y_i}{\sigma_i}\right]\left[\frac{l_i - y_i}{\sigma_i}\right]^* \tag{23}$$

Differentiating χ^2 leads to the following set of normal equations:

$$\begin{pmatrix} [a_0^* a_0] & [a_0^* a_1] & \cdots & [a_0^* a_{n-1}] \\ [a_1^* a_0] & [a_1^* a_1] & \cdots & [a_1^* a_{n-1}] \\ \vdots & \vdots & \ddots & \vdots \\ [a_{n-1}^* a_0] & [a_{n-1}^* a_1] & \cdots & [a_{n-1}^* a_{n-1}] \end{pmatrix} \cdot x = \begin{pmatrix} [a_0^* l] \\ [a_1^* l] \\ \vdots \\ [a_{n-1}^* l] \end{pmatrix} \tag{24}$$

The normal matrix is Hermitian. It can be solved by Choleski methods. However, internally the matrix is converted to a real form. Although this has an, in general, small memory penalty, it has no influence on CPU time, and makes it possible to use the same routines for complex and real solutions. The conversion to real is done by splitting each element A_{ij} of the normal matrix into $A_{\Re ij} + \imath A_{\Im ij}$ and replacing it by:

$$A_{ij} = \begin{pmatrix} A_{ij\Re} & -A_{ij\Im} \\ A_{ij\Im} & A_{ij\Re} \end{pmatrix} \tag{25}$$

and simular replacements for the vector elements x_i and L_i as:

$$x_i = \begin{pmatrix} x_{i\Re} \\ x_{i\Im} \end{pmatrix} \tag{26}$$

and:

$$L_i = \begin{pmatrix} L_{i\Re} \\ L_{i\Im} \end{pmatrix} \tag{27}$$

Another reason for solving real rather than complex equations is given in the next section.

2.2.1 Separable complex

In cases where both the unknowns x and their complex conjugates x^* appear in the condition equations, differentiating the merit function equation 23 will not produce a symmetric or Hermitian normal matrix, since there exists no linear relation between x_i and x_i^*. We could, of course, solve for $2n$ complex equations with added constraints that the sum of even and odd unknowns must be real, and their difference imaginary, but this will lead to $4n$ complex equations.

If, however, we consider each complex unknown as two real unknowns (i.e. $x_{i\Re}$ and $x_{i\Im}$) then differentiating equation 23 produces the following symmetric set of $2n$ real equations:

$$\begin{pmatrix} [a_0{}^* a_0]_\Re & [a_1{}^* a_0]_\Im & [a_2{}^* a_0]_\Re & \cdots & [a_{2n-1}{}^* a_0]_\Im \\ -[a_0{}^* a_1]_\Im & [a_1{}^* a_1]_\Re & -[a_2{}^* a_1]_\Im & \cdots & [a_{2n-1}{}^* a_1]_\Re \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -[a_0{}^* a_{2n-1}]_\Im & [a_1{}^* a_{2n-1}]_\Re & -[a_2{}^* a_{2n-1}]_\Im & \cdots & [a_{2n-1}{}^* a_{2n-1}]_\Re \end{pmatrix} \cdot$$

$$\cdot \begin{pmatrix} x_{0\Re} \\ x_{0\Im} \\ x_{1\Re} \\ \vdots \\ x_{n-1\Im} \end{pmatrix} = \begin{pmatrix} [a_0{}^* l]_\Re \\ [a_1{}^* l]_\Im \\ [a_2{}^* l]_\Re \\ \vdots \\ [a_{2n-1}{}^* l]_\Im \end{pmatrix} \tag{28}$$

A number of special cases can be distinguished.

In the 'normal' complex case (previous section), $a_{2i} \equiv a_{2i+1}$, and equation 28 reduces to equation 25 . If all a_{2i} and a_{2i+1} are real, but specified, e.g., as a complex number, equation 28 deteriorates into:

$$\begin{pmatrix} [a_{0\Re}a_{0\Re}] & 0 & [a_{0\Re}a_{1\Re}] & \cdots & 0 \\ 0 & [a_{0\Im}a_{0\Im}] & 0 & \cdots & [a_{0\Im}a_{n-1\Im}] \\ [a_{1\Re}a_{0\Re}] & 0 & [a_{1\Re}a_{1\Re}] & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & [a_{n-1\Im}a_{0\Im}] & 0 & \cdots & [a_{n-1\Im}a_{n-1\Im}] \end{pmatrix} . \\ \cdot \begin{pmatrix} x_{0\Re} \\ x_{0\Im} \\ x_{1\Re} \\ \vdots \\ x_{n-1\Im} \end{pmatrix} = \begin{pmatrix} [a_{0\Re}l_{\Re}] \\ [a_{0\Im}l_{\Im}] \\ [a_{1\Re}l_{\Re}] \\ \vdots \\ [a_{n-1\Im}l_{\Im}] \end{pmatrix} \tag{29}$$

Note that in this case there is a true separation of the real and imaginary parts of the different unknowns, and two separate real solutions with each n unknowns will produce exactly the same result.

The full and special cases are all catered for in the Newstar routines.

2.2.2 Errors

Using arguments comparable to those in (sec. 2.1.1), we can get equation 13 for the complex case as:

$$\chi^2 = [ll^*] - \sum_{k=0}^{n-1} \left(x_{k\Re}[a_{2k}^* l]_{\Re} + x_{k\Im}\left[a_{2k+1}^* l\right]_{\Im} \right) \tag{30}$$

with the a's and x's as defined in equation 29 .

2.2.3 Newstar calls

Complex *LSQ* in Newstar is implemented as a real *LSQ* with $2n$ unknowns. All calls are identical to those with those in (sec. 2.1.2), but with different *type* and some COMPLEX rather than REAL arguments:

1. generate an *LSQ*-object with:
   LOGICAL WNMLGA(*LSQ*, *type*, *n* [, *m*]) with:

   LSQ INTEGER, return pointer to created *LSQ*-object (if *.true.*)
   type coded INTEGER; coded as: LSQ_T_COMPLEX [+ LSQ_T_MULTIPLE]. The second part indicates that multiple ($m > 1$) simultaneous equations are requested.
   n INTEGER, number of (complex) unknowns
   m INTEGER, number of simultaneous equations if LSQ_T_MULTIPLE specified (default one)
2. generate normal equations from the condition equations with successive calls to:
   CALL WNMLMN(*LSQ*, *type*, *condition*, *weight*, *measured*) with:

   LSQ *LSQ*-object pointer
   type coded INTEGER, coded as:

   - 0: default, equal to LSQ_C_COMPLEX
   - LSQ_C_COMPLEX: coefficients in *condition* are n complex numbers for n complex unknowns
   - LSQ_C_REAL: coefficients in *condition* are $2n$ real numbers for n complex unknowns. Each pair are the coefficients for the real and imaginary parts of each unknown
   - LSQ_C_DCOMPLEX: coefficients in *condition* are $2n$ complex numbers for n complex unknowns. Each pair are the coefficients for the real and imaginary parts of each unknown
   - LSQ_C_CCOMPLEX: coefficients in *condition* are $2n$ complex numbers for n complex unknowns. Each pair are the coefficients for each unknown and its conjugate

   condition factors in condition equation as
   REAL$(0 : 2n-1)$ or as COMPLEX$(0 : n-1)$ or as COMPLEX$(0 : 2n-1)$, depending on *type*

weight REAL, weight of condition equation ($w = 1/\sigma^2$). Note that for the best arithmetic performance, the w_i are best normalised to a maximum value of one.

measured measured value(s) as COMPLEX$(0 : m-1)$

3. triangularise the normal equations matrix A with:
   LOGICAL WNMLTN(*LSQ*) with:

4. obtain the solution by back substitution with:
   CALL WNMLSN(*LSQ*,*solution*,*error*,*sd*) with:

LSQ *LSQ*-object pointer

solution solution x as COMPLEX$(0 : n-1, 0 : m-1)$

error adjustment error per unit weight (σ_w)as REAL$(0 : m-1)$

sd standard deviation (σ_o) as REAL$(0 : m-1)$

5. obtain the covariance matrix or something similar with:
   CALL WNMLME(*LSQ*,*uncertainties*) or:
   CALL WNMLCV(*LSQ*,*covariance*) or:
   CALL WNMLIN(*LSQ*) with:

LSQ *LSQ*-object pointer

uncertainties the uncertainties in the unknowns x as COMPLEX$(0 : n-1, 0 : m-1)$. Note that in actual fact the errors are for the real and imaginary parts only. True errors for the complex unknowns will be $\sqrt{\frac{x_{\Im}^2\sigma_{\Im}^2 + x_{\Re}^2\sigma_{\Re}^2}{xx^*}}$

covariance the covariance (inverse) matrix of the solution, as REAL$(0 : 2n-1, 0 : 2n-1)$

WNMLIN will calculate the covariance matrix internally only. In all three cases the covariance matrix will overwrite the normal array, and it will be used, rather than back substitution, in WNMLSN if that routine is called after one of the inversion ones.

6. free *LSQ*-object with:
   CALL WNMLFA(*LSQ*)

The same additional calls to re-use an (*LSQ*-object (sec. 2.1.3)), with the same arguments, are available as for the real case. The type and number of *condition* are, of course, dependent on the *type* of solution (real or complex) and *type* of condition equations.

3 Dependant linear equations

If there are not enough independent condition equations, the normal matrix A cannot be inverted, and a call to WNMLTN will fail with a *.false.* return value.

The equations could still be solved if some additional 'constraint' equations would be introduced. In the more complex cases the precise, let alone the best, form for these additional equations is difficult to determine (e.g. the redundancy situation in Westerbork).

A method known as 'Single value decomposition' (*SVD*) can be used to obtain the minimal set of orthogonal equations that have to be added to solve the *LSQ* problem. Several implementations exist in the literature.

In general we can distinguish three types of constrained equations:

- the minimum number of sufficient constraint equations are known to be able to solve a system of equations
- constraints are used to add additional information (e.g. the sum of angles of a triangle)
- no actual constraint equations are known

All three cases are handled in Newstar, the first two in the same way.

The general constraint situation arises from the use of Lagrange multiplicators. Assume that in addition to the condition equations, with measured values, we have a set of p rigorous equations:

$$\phi_i(x) = 0 \qquad i = 0, \ldots, p-1 \tag{31}$$

We must therefore make χ^2 minimal, subject to the set of equation 31 , or:

$$\sum_{i=0}^{n-1} \frac{\partial \chi^2}{\partial x_i} dx_i = 0 \tag{32}$$

subject to the conditions:

$$\sum_{i=0}^{n-1} \frac{\partial \phi_k}{\partial x_i} dx_i = 0 \qquad k = 0, \ldots, p-1 \tag{33}$$

which leads to a set of $n+p$ equations:

$$\frac{\partial \chi^2}{\partial x_i} + \sum_{k=0}^{p-1} \lambda_k \frac{\partial \phi_k}{\partial x_i} = 0 \qquad i = 0, \ldots, n-1 \tag{34}$$

together with the equation 31 .

Note that in Newstar I have chosen for having constraint equations linear in the unknowns, with a zero value. In cases where this is not adequate (e.g. $x+y+z = 360$) a simple linear transformation will suffice to make it e.g. $x' + y' + z' = 0$.

Defining the second term in equation 34 as B_{ik}, we can write our expanded set of normal equations as:

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} L \\ 0 \end{pmatrix} \tag{35}$$

3.1 Unknown constraints

The Newstar routines to handle this situation use a method based on "Pseudo-inverse berekening en Cholesky factorisatie", Hans van der Marel, 27 maart 1990. I will briefly describe Hans' paper, together with the additions I have made.

In the case we are considering, the normal equation A has not full column rank. Therefore, there exists, if the rank defect is p, an $n \times p$ matrix G, a basis for the null-space of A, with $AG = 0$. If we assume that B has just sufficient constraint equations to solve the rank defect, the inverse of the matrix in equation 35 is:

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix}^{-1} = \begin{pmatrix} A^{\#} & G(B^T G)^{-1} \\ (G^T B)^{-1} G^T & 0 \end{pmatrix} \tag{36}$$

with $A^{\#}$ the pseudo-inverse of A. A number of relations hold for $A^{\#}$, the most important for us that $B^T A^{\#} = 0$ and $A^{\#} B = 0$, or B is a base for the null-space of $A^{\#}$. For a mininorm solution we can choose $B = G$. In that case $A^{\#}$ is the Moore-Penrose inverse, and $B^T G$ is regular and symmetric.

We can now proceed in the following way:

1. Do a Cholesky factorisation of the normal equations A until the remaining columns of A are dependent. Pivoting along the diagonal of A occurs to make sure that A can be partitioned in an independent and a dependent part with rank defect n_2. Dependency is determined by looking at the angle between a column and the space formed by the already determined independent space (the so-called 'collinearity number').

   If $n_1 = n - n_2$, we can say that after n_1 Choleski steps, we have:

   $$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} U_{11}^T & 0 \\ U_{12}^T & I \end{pmatrix} \cdot \begin{pmatrix} U_{11} & U_{12} \\ 0 & \overline{A}_{22} \end{pmatrix} \tag{37}$$

   with U_{11} the Cholesky factor of A_{11}, $U_{12} = {U_{11}^T}^{-1} A_{12}$ and $\overline{A}_{22} = A_{22} - U_{12}^T U_{12}$.

   The collinearity angle δ can be determined for the current column i by: $\sin^2 \delta = u_{ii}^2 / a_{ii}$.
2. Determine a G_1 replacing the (rectangular) U_{12} from $U_{11} G_1 = -U_{12}$.
3. Determine a (symmetric) G_2 replacing A_{22} from the rank basis $I + G_1^T G_1 = (G_2^T - 1)(G_2 - 1)$
4. determine constraint equations $G = (G_1 G_2, G_2)^T$.

5. Solve for the n_1 independent unknowns x_1 using A_{11} and L_1 by back substitution
6. Solve constraint equations for the remaining n_2 unknowns, using G_2 by back substitution: $x_2 = -G_2^{-1} G_1^T x_1$
7. Make Baarda's S-transform of independent n_1 unknowns: $x_1 = x_1 + G_1 x_2$.

Setting $F = A_1^{-1} L_1$, $D = -G_2^{-1} G_1^T$ and $E = \begin{pmatrix} I_{n_1} + G_1 D \\ D \end{pmatrix}$, we can write the above steps as:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} I_{n_1} + G_1 D \\ D \end{pmatrix} \left(A_1^{-1} L_1 \right) = E \cdot F \tag{38}$$

3.1.1 Errors

Errors are determined in the same way as described in (sec. 2.1.1), where it should be noted that equation 13 can either be used summing over n_1 variables and using the first guess of x_1, or over n and using the final x. Internally the first option is used. The uncertainties in x are determined by calculating the covariance matrix. Using similar arithmetic as in (sec. 2.1.1), it can be shown that:

$$A^{-1} = E \cdot a_1^{-1} \cdot E^T \tag{39}$$

A^{-1} is calculated by first solving $H = E\left(A_1^{-1} \cdot I_{n_1}\right)$ and then $A^{-1} = E \cdot H^T$.

3.1.2 Newstar calls

The implementation in Newstar of dependent equations adds only one routine WNMLTR, which replaces in this case the call to WNMLTN in (sec. 2.1.2) and in (sec. 2.2.3). The call is:

- Do Cholesky factorisation of underdetermined system:
  LOGICAL WNMLTR(*LSQ*,*rank*) with:

  LSQ *LSQ*-object pointer
  rank return INTEGER, the rank of A. Note that in the complex cases the maximum rank is $2n$.

All other calls remain identical, i.e. WNMLSN knows if a rank defect was detected by WNMLTR. The actual constraint equations used in the solution can be obtained by:

- Get constraint equations:
  CALL WNMLGC(*LSQ*,*n-constraint*,*constraints*) with:

  LSQ *LSQ*-object pointer
  n-constraint the INTEGER number of constraint equations returned, i.e. the rank deficiency
  constraints the REAL (or COMPLEX) constraint equations, returned as an array$(0 : n-1, 0 :$ *n-constraint*$-1, 0 : m-1)$

  Note that in the complex case *n-constraint* can have a maximum value of $2n$, and the returned equations are in LSQ_C_REAL type.

3.2 Known constraints

In the case of known constraints, hence when B is known in equation 35 , this equation can be solved. Cholesky decomposition does not work in this case, and a, transparent, Crout LU decomposition is done to determine the (symmetric) covariance (i.e. inverse) matrix of the left hand side of equation 35 . .

3.2.1 Errors

Errors are determined as explained in (sec. 2.1.1). Since all constraint equations are $\equiv 0$, the sum in equation 13 is taken over n, not over $n + p$ if p are the number of constraint equations.

3.2.2 Newstar calls

The basic differences with the normal procedures are:

1. the *LSQ*-object should know the number of constraint equations
2. the constraint equations (B^T) should be specified.

The calls that are necessary:

1. generate an *LSQ*-object with:
   LOGICAL WNMLGA(*LSQ*, *type*, *n* , *m*, *p*) with:

   LSQ INTEGER, return pointer to created *LSQ*-object (if *.true.*)
   type coded INTEGER; coded as:
   { LSQ_T_REAL | LSQ_T_COMPLEX} + LSQ_T_CONSTRAINTS
   [+ LSQ_T_MULTIPLE].
   The second part indicates that multiple ($m > 1$) simultaneous equations are requested.
   n INTEGER, number of unknowns
   m INTEGER, number of simultaneous equations if LSQ_T_MULTIPLE specified or 0 (default one)
   p INTEGER, number of constraint equations
2. specify the constraint equations with a call to:
   CALL WNMLMC(*LSQ*, *type*, *constraints*) with:

   LSQ *LSQ*-object pointer
   type coded INTEGER with same values as WNMLMN. In the complex case and COMPLEX type, the number of equations given should be $p/2$. that the constraint equations are real
   constraints constraint equations as REAL$(0 : n-1, 0 : p-1)$
   or as COMPLEX$(0 : n-1, 0 : p/2-1)$ or as REAL$(0 : 2n-1, 0 : p-1)$
   or as COMPLEX$(0 : 2n-1, 0 : p/2-1)$, depending on *type*

4 Non-linear equations

In the case of non-linear condition equations, no simple solution to minimise the merit function (e.g. equation 6) exists. A simple solution is to take the condition equation, make a first order Taylor expansion around an estimated $\hat{x}$; solve for dx, and iterate till solution found. This approach was used in Newstar originally.

However, better and more stable methods exist (e.g. steepest-descend) for some circumstances. A method that seems to be quite stable and using both steepest-descent and Taylor expansion, is the method by Levenberg-Marquardt (see e.g. "Numerical recipes", W.H. Press *et al.*, Cambridge University Press).

If we have an estimate for x, we can find a better one by:

$$\hat{x}_{next} = \hat{x} + H^{-1} \cdot \left[-\nabla\chi^2(\hat{x})\right] \tag{40}$$

where H is the Hessian matrix of χ^2. If our approximation is not good enough, we could use the steepest-descent by calculating:

$$\hat{x}_{next} = \hat{x} - constant \cdot \left[-\nabla\chi^2(\hat{x})\right] \tag{41}$$

The Hessian matrix H has as elements:

$$H_{ij} = \frac{\partial^2\chi^2}{\partial x_i \partial x_j} = 2\sum_{k=0}^{N-1} \frac{1}{\sigma_i^2}\left[\frac{\partial y_k}{\partial x_i}\frac{\partial y_k}{\partial x_j} - (l_i - y_i)\,\frac{\partial^2 y_i}{\partial x_i \partial x_j}\right] \tag{42}$$

By dropping the second term in equation 42 , and multiplying each H_{ii} term with $(1+\lambda)$, and defining the first derivative of χ^2 as b, we can combine the equations equation 40 , equation 41 into:

$$H \cdot dx = b \tag{43}$$

which can be solved as standard normal equations.

Choosing λ is the crux of the matter, and where to stop iterating. A start value of $\lambda = 0.001$ is used in the following routines. The method looks at the χ^2 for $\hat{x} + dx$. If it has increased over the value of χ^2 for $\hat{x}$, $\hat{x}$ is unchanged, and $\lambda = 10\lambda$. If there is a decrease; a new value for $\hat{x}$ is used, and $\lambda = \lambda/10$. Iteration can be stopped if the fractional decrease in χ^2 is small (say < 0.001); never if there is an increase in χ^2.

4.1 Errors

Errors are calculated as described in (sec. 2.1.1). The errors returned by WNMLSN are, of course, only approximations, since the original equations are non-linear, but give a good impression of the residuals. The covariance matrix should be calculated by doing a final linear run on the residuals, and solve the equations.

4.2 Newstar calls

The basic difference in calls is the use of WNMLNN(or in special cases maybe WNMLNR) rather than WNMLTN to triangularise the normal matrix. The condition equations should contain the derivative to the x_k at the position of $\hat{x}$; and the measured values the value $(l_i - y_i(\hat{x}))$.

The actual steps in the process are than:

1. create *LSQ*-object with WNMLGA (note that LSQ_T_MULTIPLE is not allowed)
2. fill it with condition equations, as described in (sec. 2.1.2) or in (sec. 2.2.3), with WNMLMN, using the residuals with an estimated solution as the l-terms, and the derivative with respect to the unknowns of the condition equations as the condition equations
3. triangularise and solve with:
   LOGICAL WNMLNN(*LSQ*,0,*solution*,*error*,*goodness*) or:
   LOGICAL WNMLNR(*LSQ*,*rank*,*solution*,*error*,*goodness*) with:

   LSQ *LSQ*-object pointer
   rank return INTEGER, the rank of A. Note that in the complex case the maximum rank is $2n$.
   solution the estimated solution should be given, and will be updated to a new solution of x as REAL$(0 : n-1)$ or COMPLEX$(0 : n-1)$
   error adjustment error per unit weight (σ_w)as REAL
   goodness goodness of current estimate as REAL. If *goodness* > 0 no stable solution reached yet; if *goodness* ≤ 0 and $|goodness| < 0.001$ the process can be stopped. *goodness* is the fractional improvement in χ^2.
4. check the *goodness* returned parameter. If it is non-positive and has an absolute value less than what you think is a good change (e.g. 0.001) stop the process
5. else re-start at 2
6. if satisfied, and the covariance matrix is wanted, re-initialise the *LSQ*-object with LSQ_I_ALL, reload condition equations, use WNMLTR (or WNMLTN if you are sure of an independent set of equations), and ask for the covariance matrix or vector (WNMLCV or WNMLME).
7. free *LSQ*-object with WNMLFA

5 Summary

An *LSQ*-object takes a total space of:

$$\begin{array}{rcl} 9 & + & \text{(administration)} \\ (n+p)(n+p+1)/2 & + & \text{(normal array)} \\ 4m & + & \text{(error calculations)} \\ m(m+p) & + & \text{(known sides)} \\ (n+p+1)/2 & + & \text{(pivot area)} \\ (n+p) & & \text{(solution) 8 byte words} \end{array} \tag{44}$$

where n is the number of unknowns ($2n$ if complex); m the number of simultaneously to be solved equations; p the number of external constraint equations. In the non-linear case two *LSQ*-objects are used. In cases where the inverted normal array is calculated (known constraints, or explicit calls to WNMLME, WNMLCV or WNMLIN) a temporary storage of n^2 8-byte words is used.

The following calls are available:

1. generate an *LSQ*-object with:
   LOGICAL WNMLGA(*LSQ*, *type*, *n* [, *m*[, *p*[, *collinearity*]]]) with:

   LSQ INTEGER, return pointer to created *LSQ*-object (if *.true.*)

   type coded INTEGER; coded as: 0 or { LSQ_T_REAL | LSQ_T_COMPLEX} (REAL is default if not specified)
   + one or more of the following optional ones:
   LSQ_T_MULTIPLE if m present (else 1 assumed)
   LSQ_T_CONSTRAINT if p present (else 0 assumed)
   LSQ_T_PREC if *collinearity* present (else default 10^{-6} assumed)
   Optional trailing arguments may be omitted; embedded ones should have a value, preferably "0".

   n INTEGER, number of unknowns

   m INTEGER, number of simultaneous equations if LSQ_T_MULTIPLE specified (default 1)

   p INTEGER, number of user specified constraints if LSQ_T_CONSTRAINT (default 0)

   collinearity REAL, the value to be used in dependency checks (default 10^{-6})

2. delete an *LSQ*-object with:
   CALL WNMLFA(*LSQ*) with:

   LSQ INTEGER, pointer to *LSQ*-object

3. re-initialise an existing *LSQ*-object for re-use with:
   CALL WNMLIA(*LSQ*, *type* [, *collinearity*]) with:

   LSQ *LSQ*-object pointer
   type coded INTEGER, coded as a combination of:

   - 0: default, equal to LSQ_I_ALL
   - LSQ_I_NORM: re-initialise the 'unknown' part of the object (i.e. the normal array A)
   - LSQ_I_KNOWN: re-initialise the 'known' part of the object (i.e. the vector L)
   - LSQ_I_SOL: equal to NORM + KNOWN
   - LSQ_I_NONLIN: re-initialise the non-linear part of the object
   - LSQ_I_ALL: equal to SOL + NONLIN
   - LSQ_I_PREC: set the internal check precision to *collinearity*

   collinearity precision factor to be used (REAL). Only used if LSQ_I_PREC set. Note that the default internal value is 10^{-6}.

4. generate normal equations from the condition equations with successive calls to:
   CALL WNMLMN(*LSQ*, *type*, *condition*, *weight*, *measured*) with:

   LSQ *LSQ*-object pointer
   type coded INTEGER, coded in the real case as one of :

   - 0: default, equal to LSQ_C_REAL
   - LSQ_C_REAL condition equations have REAL$(0 : n - 1)$ values

   or in the complex case as one of:

   - 0: default, equal to LSQ_C_COMPLEX
   - LSQ_C_REAL condition equations have REAL$(0 : 2n - 1)$ values
   - LSQ_C_COMPLEX condition equations have COMPLEX$(0 : n - 1)$ values
   - LSQ_C_CCOMPLEX condition equations have COMPLEX$(0 : 2n - 1)$ values for conjugate variables
   - LSQ_C_DCOMPLEX condition equations have COMPLEX$(0 : 2n - 1)$ values for separate real and imaginary variables

   in both cases it can be combined with one or more of:

   - LSQ_C_NONORM to bypass updating the normal array (A)
   - LSQ_C_NOKNOWN to bypass updating the known vector(s) (L)

   condition factors in condition equation as REAL$(0 : n - 1)$, REAL$(0 : 2n - 1)$, COMPLEX$(0 : n - 1)$ or COMPLEX$(0 : 2n - 1)$, depending on the *type*
   weight REAL, weight of condition equation ($w = 1/\sigma^2$). Note that for the best arithmetic performance, the w_i are best normalised to a maximum value of one.
   measured measured value(s) as REAL$(0 : m - 1)$ or COMPLEX$(0 : m - 1)$

5. specify known constraint equations with a call to:
   CALL WNMLMC(*LSQ*, *type*, *constraints*) with:

   - *LSQ* — *LSQ*-object pointer
   - *type* — the same *type* as used in WNMLMN to indicate the type of equations
   - *constraints* — constraint equations as REAL$(0:n-1,0:p-1)$ or as COMPLEX$(0:n-1,0:p/2-1)$, COMPLEX$(0:2n-1,0:p/2-1)$ or as REAL$(0:2n-1,0:p-1)$, depending on *type*

6. prepare normal equations for solution with one of:
   LOGICAL WNMLTN(*LSQ*)
   LOGICAL WNMLTR(*LSQ*,*rank*)
   LOGICAL WNMLNN(*LSQ*,0,*solution*,*error*,*goodness*)
   LOGICAL WNMLNR(*LSQ*,*rank*,*solution*,*error*,*goodness*) with:

   - *LSQ* — *LSQ*-object pointer
   - *rank* — return INTEGER, the rank of A. Note that in the complex case the maximum rank is $2n$.
   - *solution* — the estimated solution should be given, and will be updated to a new solution of x as REAL$(0:n-1)$ or COMPLEX$(0:n-1)$
   - *error* — adjustment error per unit weight (σ_w)as REAL
   - *goodness* — goodness of current estimate as REAL. If *goodness* > 0 no stable solution reached yet; if *goodness* ≤ 0 and $|$*goodness*$| < 0.001$ the process can be stopped. *goodness* is the fractional improvement in χ^2.

   WNMLTN assumes that the normal equations are fully determined; WNMLTR will determine the minimum orthogonal set of constraint equations necessary to solve the normal equations; WNMLNN will solve an update to a solution *solution*for non-linear condition equations; WNMLNR will solve as WNMLNN, but will also calculate the set of constraints if necessary.

7. obtain the solution by back substitution for the non-linear case with:
   CALL WNMLSN(*LSQ*,*solution*,*error*,*sd*) with:

   - *LSQ* — *LSQ*-object pointer
   - *solution* — solution x as REAL$(0:n-1,0:m-1)$ or COMPLEX$(0:n-1,0:m-1)$
   - *error* — adjustment error per unit weight (σ_w)as REAL$(0:m-1)$
   - *sd* — standard deviation (σ_o) as REAL$(0:m-1)$

8. get calculated constraint equations:
   CALL WNMLGC(*LSQ*,*n-constraint*,*constraints*) with:

   - *LSQ* — *LSQ*-object pointer
   - *n-constraint* — the INTEGER number of constraint equations returned, i.e. the (real) rank deficiency
   - *constraints* — the REAL (or COMPLEX) constraint equations, returned as an array$(0:n-1,0:$*n-constraint*$-1,0:m-1)$

   Note that in the complex case *n-constraint* can have a maximum value of $2n$.

9. obtain the covariance matrix or something similar with:
   CALL WNMLME(*LSQ*,*uncertainties*) or:
   CALL WNMLCV(*LSQ*,*covariance*) or:
   CALL WNMLIN(*LSQ*) with:

   - *LSQ* — *LSQ*-object pointer
   - *uncertainties* — the uncertainties in the unknowns x as REAL$(0:n-1,0:m-1)$ or as COMPLEX$(0:n-1,0:m-1)$
   - *covariance* — the covariance (inverse) matrix of the solution, as REAL$(0:n-1,0:n-1)$ or as REAL$(0:2n-1,0:2n-1)$ for the complex case

   WNMLIN will calculate the covariance matrix internally only. In all three cases the covariance matrix will overwrite the normal array, and it will be used, rather than back substitution, in WNMLSN if that routine is called after one of the inversion ones. After an inversion call the constraint equations can no longer be obtained