REFERENCE COPY
NOT TO BE REMOVED
FROM THE LIBRARY

AERE - R 6799

Copy 2

NOT FOR TNA

United Kingdom Atomic Energy Authority

RESEARCH GROUP

Report

A MODIFIED MARQUARDT SUBROUTINE FOR NON-LINEAR LEAST SQUARES

R. FLETCHER

16 SEP 1976
27 JAN 1978

8 NOV 1989

Theoretical Physics Division,

Atomic Energy Research Establishment,

Harwell, Berkshire.

1971

Price 20p net from H. M. Stationery Office.

2025
15929

C. 13

© - UNITED KINGDOM ATOMIC ENERGY AUTHORITY - 1971
Enquiries about copyright and reproduction should be addressed to the Scientific Administration Office, Atomic Energy Research Establishment, Harwell, Didcot, Berkshire, England.

U.D.C.
681.14:519.281

AERE - R.6799

"A MODIFIED MARQUARDT SUBROUTINE FOR NON-LINEAR LEAST SQUARES"

by

R. Fletcher

SUMMARY

A FORTRAN subroutine is described for minimizing a sum of squares of functions of many variables. Such problems arise in non-linear data fitting, and in the solution of non-linear algebraic equations. The subroutine is based on an algorithm due to Marquardt, but with modifications which improve the performance of the method in certain circumstances, yet which require negligible extra computer time and storage.

Mathematics Branch,
Theoretical Physics Division,
U.K.A.E.A. Research Group,
Atomic Energy Research Establishment,
HARWELL.

May 1971

HL71/2812 (C13)

SBN 7058 0560 0

1. Introduction

The problem under consideration is that of minimizing a sum of squares $S(\underset{\sim}{x})$ of several non-linear functions $r_i(\underset{\sim}{x})$ of many variables $\underset{\sim}{x}$, that is

$$S(\underset{\sim}{x}) = \sum_{i=1}^{m} [r_i(\underset{\sim}{x})]^2$$

where $\underset{\sim}{x} = (x_1, x_2, \ldots, x_n)^T$, and where $m \geqslant n$. Such problems arise typically in non-linear least squares data fitting, when r_i is the residual difference between observed and predicted quantities, and also when solving systems of non-linear equations. Two very similar methods have been published for this problem, by K. Levenberg (Quart. Appl. Maths., 1944, Vol. 2, p.164), and by D. W. Marquardt (Jour. SIAM, 1963, Vol. 11, p.431). If the mxn Jacobian matrix J is defined by $J_{ij} = \partial r_i / \partial x_j$ then each iteration can be written as

$$\underset{\sim}{x}^{(k+1)} = \underset{\sim}{x}^{(k)} + \underset{\sim}{\delta}^{(k)}$$

where $\underset{\sim}{\delta}$ (dropping subscripts where no confusion can occur) is the solution of the set of linear equations

$$(A + \lambda I)\,\underset{\sim}{\delta} = -\underset{\sim}{v} \qquad (1)$$

where $A = J^T J$ and $\underset{\sim}{v} = J^T \underset{\sim}{r}$ are evaluated at $\underset{\sim}{x}^{(k)}$, and where λ is an adjustable parameter which is used to control the iteration.

On any one iteration A and $\underset{\sim}{v}$ are fixed, so that $\underset{\sim}{\delta}$ may be considered as a function $\underset{\sim}{\delta}(\lambda)$ of λ. As $\lambda \rightarrow \infty$, then $\underset{\sim}{\delta}(\lambda) \rightarrow -\underset{\sim}{v}/\lambda$ which is an incremental step along the direction of steepest descent of $S(\underset{\sim}{x})$ at $\underset{\sim}{x}^{(k)}$. As $\lambda \rightarrow 0$, then $\underset{\sim}{\delta}(\lambda) \rightarrow -A^{-1}\underset{\sim}{v}$ which is the correction predicted by the Gauss-Newton or Generalized least squares method. Steepest descent methods are known to be convergent but slow, whereas rapid but less reliable convergence is usually obtained with the Gauss-Newton method. The motivation of the Levenberg-Marquardt methods is that they attempt to choose λ so as to follow the Gauss-Newton method to as large an extent as possible, whilst retaining a bias towards the

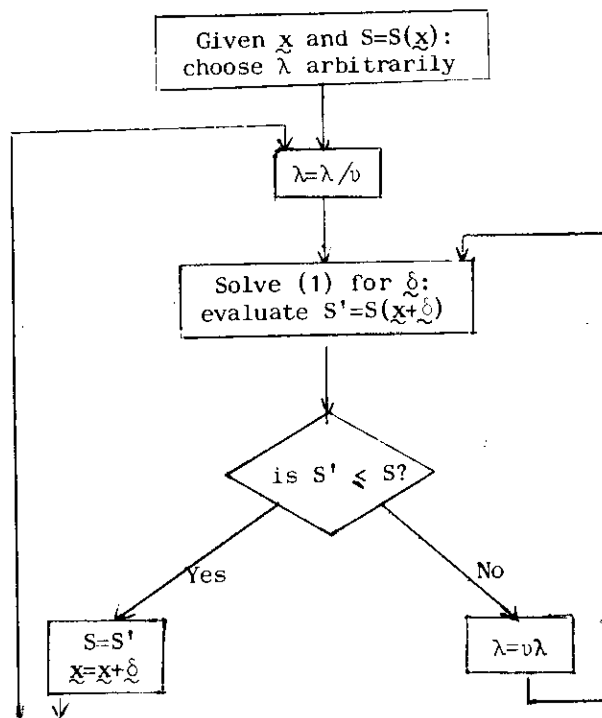
steepest descent direction to prevent divergence.

The methods differ in the way in which λ is selected on each iteration. Levenberg suggested that it would be preferable to estimate the minimum of $S(\underset{\sim}{x}+\underset{\sim}{\delta}(\lambda))$ as a function of λ, on each iteration. This requires the solution of (1) and the calculation of S to be repeated for a number of different values of λ. This suggestion is open to a number of objections but primarily it is less efficient to spend time in looking for a minimum of S with respect to λ, rather than to start a new iteration with more up-to-date information for A and $\underset{\sim}{v}$, once a sufficient reduction in the sum of squares has been obtained. Furthermore the programming of this search process is complicated by a number of difficult ad-hoc decisions, and in this case in particular it is not at all obvious what sort of behaviour $S(\lambda)$ relating S to λ should be assumed, to make the interpolations which would be required. However I do not agree with Marquardt who suggested that Levenberg's scheme would lead to a serious over-emphasis of the bias towards steepest descents. Unfortunately Levenberg never stated his ideas in a sufficiently precise form that a computer program could be written from them, so no further consideration of his choice of λ will be given.

Marquardt suggested a way of varying λ which gave much more hope of being efficient in the number of solutions of (1) and evaluations of S required per iteration. His idea was to choose a fixed parameter υ (υ=10 was recommended), increasing or decreasing λ by multiples of υ or $1/\upsilon$ as necessary, and terminating any iteration once a $\underset{\sim}{\delta}$ had been obtained for which $S(\underset{\sim}{x}+\underset{\sim}{\delta}) < S(\underset{\sim}{x})$. Unfortunately Marquardt's statement of the algorithm makes it appear necessary that $S(\underset{\sim}{x})$ is evaluated at least twice per iteration, and the text confirms this impression. However if his algorithm is written as shown in Figure 1, then it is clear that only one evaluation of $S(\underset{\sim}{x})$ may be required on some iterations.

Marquardt's algorithm is very simple, and some limited tests with a version of it show that it is reasonably efficient. In fact, on many problems, an average of little over one solution of (1) per iteration is required. Unfortunately Marquardt's original FORTRAN subroutine in the IBM SHARE subroutine library seems to be difficult to obtain, and this motivated the writing of a more readily accessible subroutine. Having decided to do this, a number of modifications to eliminate some less favourable aspects of the method were also planned. For instance the arbitrary initial choice of λ, if poor, can cause the wastage of a number of evaluations of S before a realistic value is obtained. This is especially noticeable if υ is chosen to be fairly small, υ=2 say. Another

Figure 1



Flow diagram for Marquardt's method

disadvantage of the method is that the reduction of λ to λ/υ at the start of each iteration may prove to be excessive, especially if υ is chosen to be large ($\upsilon=10$, say). The effect of this is that the average number of evaluations of S per iteration may be about 2, which is unnecessarily inefficient. A further disadvantage of the method is that the test $S' \leqslant S$ (or even $S' < S$) precludes a proof of convergence being made. Finally, when solving problems in which $\underset{\sim}{r}=0$ at the solution, it is possible to achieve a quadratic rate of convergence with the Gauss-Newton method but only a superlinear rate with the Marquardt scheme. The modifications of section 2 of this report represent an attempt to circumvent these difficulties.

2. Modifications

Although Marquardt's idea of replacing $\underset{\sim}{x}^{(k)}$ by $\underset{\sim}{x}^{(k)} + \underset{\sim}{\delta}$ when a better sum of squares is obtained, is followed, the circumstances under which λ is changed are modified. The initial reduction of λ to λ/υ (see Figure 1) is discontinued. After solving (1) and evaluating $S' = S(\underset{\sim}{x}^{(k)} + \underset{\sim}{\delta})$, a new value of λ is calculated by comparing the actual reduction $S-S'$ in the sum of squares with that predicted on a linear model. If $\Phi(\underset{\sim}{x})$ represents the predicted sum of squares, then the predicted reduction is given by

$$\Phi(\underset{\sim}{x}^{(k)}) - \Phi(\underset{\sim}{x}^{(k)} + \underset{\sim}{\delta}) = -2\underset{\sim}{\delta}^T\underset{\sim}{v}^{(k)} - \underset{\sim}{\delta}^T A^{(k)}\underset{\sim}{\delta} .$$

The motivation for the strategy to be described is that if the ratio R of actual reduction/predicted reduction is near 1, then λ ought to be reduced, and if the ratio is near to or less than 0, then λ ought to be increased. However for some intermediate values of λ it is probably best to leave λ unchanged for the next iteration. In fact it has been found satisfactory merely to choose arbitrary constants ρ and σ such that $0 < \rho < \sigma < 1$, and to reduce λ if $R < \rho$, and to increase λ if $R > \sigma$. In fact various experiments were tried with ρ in the range 0.01 to 0.25 and σ in the range .5 to .9; however it was found that the rate of convergence was largely insensitive to different choices of ρ and σ, and in fact the values $\rho = 0.25$ and $\sigma = 0.75$ were finally chosen.

The method chosen for increasing λ is similar to that used by Marquardt in that λ is increased to $\upsilon\lambda$. It was found that on most iterations the value $\upsilon = 2$ would be adequate, but on early iterations, when λ might be much too small, then a larger factor of say $\upsilon = 10$ would be desirable. Thus it was decided to allow the use of a multiple υ between 2 and 10, and an automatic method for choosing a multiple in this range was devised. For large values of λ, increasing λ to $\upsilon\lambda$ corresponds approximately to reducing $\underset{\sim}{\delta}$ to $\underset{\sim}{\delta}/\upsilon$. Now because the sum of squares and its derivative is available at $\underset{\sim}{x}^{(k)}$, and because the sum of squares is available at $\underset{\sim}{x}^{(k)}+\underset{\sim}{\delta}$, it is possible to estimate the optimum correction $\alpha\underset{\sim}{\delta}$ in the direction $\underset{\sim}{\delta}$ from the formula

$$\alpha = 1/(2 - (S(\underset{\sim}{x}^{(k)}+\underset{\sim}{\delta}) - S(\underset{\sim}{x}^{(k)}))/\underset{\sim}{\delta}^T\underset{\sim}{v}).$$

From the assumptions of reciprocity, a multiple $\upsilon = 1/\alpha$ is chosen by which to increase λ. This multiple is replaced by 2 or 10 if it is less than 2 or greater than 10 respectively. The test has worked very well in practice, yet is very simple to apply.

The modification to Marquardt's idea for reducing λ comes from the feeling that the geometric progression choice of λ works least well for very small λ. As $\lambda \rightarrow 0$, $\| \underset{\sim}{\delta}(\lambda) \| / \| \underset{\sim}{\delta}(\lambda/\upsilon) \| \rightarrow 1$ and the changes in $\underset{\sim}{\delta}$ on replacing λ by λ/υ are much smaller than might be desired. For instance if good progress is made with the Marquardt method for a number of iterations, then a λ of around machine accuracy might be obtained. If an

iteration then occurs on which the sum of squares is not improved, quite a number of increases of λ and hence evaluations of S might be necessary before a significant reduction in δ is obtained. Another disadvantage of the geometric progression method is that it precludes the quadratic rate of convergence being achieved when $r = 0$ at the solution. One way of getting around these difficulties is to reduce λ to λ/υ as with Marquardt's method ($\upsilon = 2$ has been used), but to define a "cut-off" value λ_c such that any values of $\lambda < \lambda_c$ are replaced by $\lambda = 0$. However there are some difficulties with the cut-off strategy which have to be overcome before it is acceptable.

The most important difficulty lies in the actual choice of λ_c. Too small a value, although not catastrophic, has the effect that very little modification is being made to the method at all. Too large a value however can be catastrophic, in that the iteration can oscillate by failing to make progress with a value of $\lambda = 0$, and then making an incremental step along the steepest descent vector with $\lambda = \lambda_c$. Clearly it is necessary to make a good automatic choice of λ_c and not an arbitrary one. To do this, it is argued that λ_c would be suitable if it caused $\| \delta(\lambda_c) \| / \| \delta(0) \| = \frac{1}{2}$. Then to go from $\lambda = 0$ to $\lambda = \lambda_c$ would cause $\| \delta \|$, to be halved which is what happens with large λ on going from λ to 2λ. An estimate of such a λ_c, which is usually realistic, yet which is on the small side and therefore fail-safe, is given by

$$\lambda_c = 1/\|A^{-1}\|.$$

To show this a simple lemma will be proved.

<u>Lemma</u> If the spectral decomposition of A is given by $A = \sum_i \mu_i \xi_i \xi_i^T$ where $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_n$, and if $0 \leq \lambda \leq \mu_n$, then $\| \delta(\lambda) \|_2 \geq \frac{1}{2} \| \delta(0) \|_2$.

<u>Proof</u> By (1) and the decomposition of A, $\delta(\lambda)$ can be written

$$\delta(\lambda) = \sum_i \xi_i \, \xi_i^T v \, / \, (\mu_i + \lambda)$$

whence

$$\delta(\lambda)^T \delta(\lambda) = \sum_i (\xi_i^T v / (\mu_i + \lambda))^2 \qquad (2)$$

Therefore the inequality

$$\tfrac{1}{4}\underset{\sim}{\delta}(0)^T\underset{\sim}{\delta}(0) = \sum_i (\underset{\sim}{\xi}_i^T\underset{\sim}{v}/(2\mu_i))^2$$

$$\leq \sum_i (\underset{\sim}{\xi}_i^T\underset{\sim}{v}/(\mu_i+\lambda))^2 = \underset{\sim}{\delta}(\lambda)^T\underset{\sim}{\delta}(\lambda) \qquad (3)$$

can be obtained, whence the Lemma follows. QED.

Clearly the choice $\lambda_c = 1/\|A^{-1}\|$ satisfies the conditions of the Lemma for any definition of $\|.\|$. In practice both the L_∞ norm of A^{-1} and the trace of A^{-1} have been calculated, both being overestimates of μ_n^{-1}. These estimates of μ_n^{-1} have usually been no worse than about $2\mu_n^{-1}$ in the examples which have been considered. Furthermore the term in $\underset{\sim}{\xi}_i^T\underset{\sim}{v}/(\mu_n+\lambda)$ usually dominates (2) for small λ, so the inequality (3) is usually fairly tight. Thus a quite effective and fail-safe choice of a cut-off value can be determined.

The best way of using this result in an algorithm must now be determined. First of all, to solve equations (1) requires $\sim n^3/6$ operations, whilst the additional calculation to obtain A^{-1} is $\sim n^3/3$ operations. Thus it is somewhat expensive in computer time to recalculate λ_c on each iteration (although not in storage because A^{-1} can overwrite the Choleski factor L). On the other hand, to set up λ_c once only might lead to the oscillatory behaviour described above, if the eigensolution of A changed significantly with $\underset{\sim}{x}$. The compromise which has been adopted is to recalculate λ_c every time λ is increased from zero to some positive number. This avoids any possibility of oscillation, yet in practice has required the evaluation of A^{-1} at most about twice per problem.

Another difficulty with a cut-off strategy is that a method in which λ is increased to some multiple $\nu\lambda$ no longer works when $\lambda = 0$. However a simple way of avoiding the difficulty is to adopt the convention that a change from $\lambda = 0$ to $\lambda = \lambda_c$ is equivalent to doubling λ, by virtue of $\|\underset{\sim}{\delta}(\lambda_c)\| \approx \tfrac{1}{2}\|\underset{\sim}{\delta}(0)\|$. Given this convention, then the strategy for increasing λ which was described earlier in this section can be applied without change.

When a cut-off strategy is in use, an arbitrary initial choice of λ is no longer necessary, because the choice $\lambda = 0$ suggests itself in a natural way and has therefore been used in the subroutine. In some data fitting problems it has been found that this value of λ can be used on every iteration. However it might be possible to argue a case

for choosing $\lambda = \lambda_c$ initially, and this would certainly be preferable to the arbitrary initial choice $\lambda = .01$ in the unmodified Marquardt method.

A flow diagram for the modified algorithm is given in Figure 2.

3. A FORTRAN subroutine

A FORTRAN subroutine has been written to implement Marquardt's method, together with the modifications described in the last section. An additional feature which is also included, is the ability to scale the variables. Marquardt shows that solving the system

$$(A + \lambda D)\underset{\sim}{\delta} = -\underset{\sim}{v}$$

at each iteration, where D is a constant diagonal matrix with $D_{ii} > 0$ for all i, is equivalent to using scaled variables $\underset{\sim}{x}^*$ such that changes δ_i^* in x_i^* are related to changes δ_i in x_i by $\delta_i^* = \sqrt{D_{ii}}\ \delta_i$. It is important that the variables be scaled in a realistic way because the method has the important property that the solution $\underset{\sim}{\delta}$ of equation (1) is the correction which minimizes the prediction $\Phi(\underset{\sim}{x}-\underset{\sim}{x}^{(k)})$ subject to $\| \underset{\sim}{x}-\underset{\sim}{x}^{(k)} \|_2 \leq \| \underset{\sim}{\delta} \|_2$. This implies that use of the L_2 norm, and hence the scaling of the variables, should be appropriate. A good choice of D is that described by Marquardt (q.v.) in which D is chosen as the diagonal of A, evaluated using the initial $\underset{\sim}{x}$. This choice is given as standard in the subroutine, although the choices D = I or any other D supplied by the user, are allowed as options.

The FORTRAN subroutine (identifier VA07A) is listed in Appendix 1, and the specification sheet giving details of its use appears as Appendix 2. Specification sheets of two other subroutines called by VA07A are given in Appendix 3. It should be mentioned here that VA07A uses a feature of one of these subroutines, MA10A, which is not described in its specification sheet. When solving equations, the Choleski factor L (for which $A = LL^T$) is stored as L^T in the working space A, thus overwriting only the diagonal elements of the lower triangle of A. It is important that any replacement for MA10A should possess the same property.

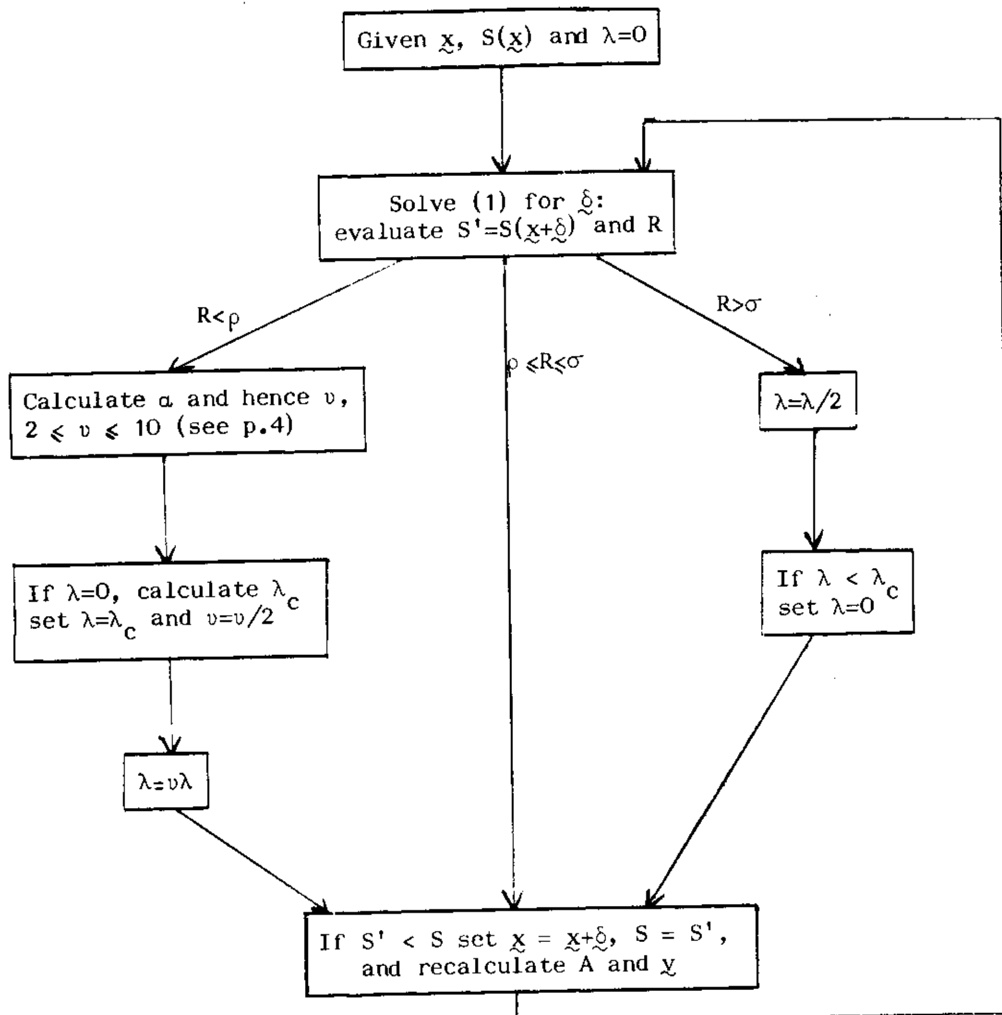
Finally the results on a few test problems are given as they might be useful for comparative purposes. The accuracy obtained in each variable is .00005 corresponding to about 4 decimal places or to a discrepancy of about 10^{-9} in the minimum S.

Table 1

Rosenbrock's sum of squares ($r_1=1-x_1$, $r_2=10(x_2-x_1^2)$)		
M=N=2	13 iterations	17 calls of RESID
Chebyquad sum of squares (see R. Fletcher, Computer J., 1965, Vol. 8, p.33)		
M=N=2	3 iterations	4 calls of RESID
M=N=4	4 iterations	6 calls of RESID
M=N=6	6 iterations	8 calls of RESID
M=N=8	15 iterations	22 calls of RESID

The Chebyquad N=8 case is interesting in that the equations have no exact solution. This implies that the matrix A evaluated at the value of $\underset{\sim}{x}$ corresponding to the minimum sum of squares will be singular, and hence that a Gauss-Newton method or Gauss-Newton method with linear searches would work badly. The figures show a quite acceptable amount of computation for the modified Marquardt method. The subroutine has also been tested on some data fitting problems for which M > N, with satisfactory results.

Figure 2



Flow diagram for the modified Marquardt method

Appendix 1

The listing of the FORTRAN subroutine VA07A

```
0001          SUBROUTINE VA07A(RESID,LSQ,M,N,X,R,SS,A,D,EPS,IPRINT,MAXFN,MODE)
0002          DIMENSION X(1),R(1),A(N,1),D(1),EPS(1)
0003          COMMON/VA07B/S(25)
0004          COMMON/VA07C/T(25)
0005          COMMON/VA07D/U(25)
0006          COMMON/VA07E/V(25)
0007          COMMON/VA07F/W(200)
0008          IF(IPRINT.NE.0)PRINT 1000
0009     1000 FORMAT('1ENTRY TO VA07A')
0010          RHO=.25
0011          SIG=.75
0012          Q=0.
0013          QC=1.
0014          IFL=0
0015          CALL RESID(M,N,X,R,IFL)
0016          IF(IFL.EQ.0)GOTO5
0017          PRINT 999
0018      999 FORMAT('0INITIAL DATA CAUSES A FAIL IN SUBROUTINE RESID')
0019          RETURN
0020        5 CONTINUE
0021          IR=1
0022          CALL MC03AS(R(1),R(2),R(1),R(2),0.,SS,M,4)
0023          CALL LSQ(M,N,X,R,A,V)
0024          IT=0
0025          IF(MODE.EQ.3)GOTO19
0026          IF(MODE.EQ.2)GOTO11
0027          DO 10 I=1,N
0028          D(I)=A(I,I)
0029          IF(D(I).LE.0.)D(I)=1.
0030       10 CONTINUE
0031       19 CONTINUE
0032          DO 18 I=1,N
0033       18 T(I)=SQRT(D(I))
0034          GOTO20
0035       11 CONTINUE
0036          DO 12 I=1,N
0037          T(I)=1.
0038       12 D(I)=1.
0039       20 CONTINUE
0040          IF(IPRINT.EQ.0)GOTO21
0041          IF(MOD(IT,IABS(IPRINT)).NE.0)GOTO21
```

```
0042               PRINT 1001,IT,IR
0043         1001  FORMAT('0',30I4)
0044               PRINT 1002,SS
0045         1002  FORMAT((8E15.7))
0046               PRINT 1002,(X(I),I=1,N)
0047               PRINT 1002,(V(I),I=1,N)
0048               IF(IPRINT.LT.0)GOTO21
0049               PRINT 1002,(R(I),I=1,M)
0050           21  CONTINUE
0051               IT=IT+1
0052               DO 31 I=1,N
0053               U(I)=A(I,I)
0054               IF(U(I).GE.0.)GOTO31
0055               PRINT 1003,I,I
0056         1003  FORMAT('0ERROR IN A(',I3,',',I3,')')
0057               RETURN
0058           31  CONTINUE
0059           35  CONTINUE
0060               DO 36 I=1,N
0061               S(I)=V(I)
0062           36  A(I,I)=U(I)+Q*D(I)
0063               CALL MA10A(A,S,N,1,NR,0,N,N)
0064               IF(NR.EQ.0)GOTO40
0065               Q=2.*Q
0066               IF(Q.EQ.0.)C=1.
0067               GOTO35
0068           40  CONTINUE
0069               DO 37 I=1,N
0070           37  W(I)=S(I)
0071               CALL MC03AS(V(1),V(2),W(1),W(2),0.,VW,N,4)
0072               IF(VW.LE.0.)GOTO83
0073               DO 42 I=1,N
0074               Z=U(I)*W(I)
0075               IF(I.GT.1) CALL MC03AS(A(I,1),A(I,2),W(1),W(2),Z,Z,I-1,4)
0076               IF(I.LT.N)
                  1  CALL MC03AS(A(I+1,I),A(I+2,I),W(I+1),W(I+2),Z,Z,N-I,4)
0077           42  S(I)=2.*V(I)-Z
0078               CALL MC03AS(S(1),S(2),W(1),W(2),0.,DQ,N,4)
0079               J=0
0080               DO 50 I=1,N
0081               IF(ABS(W(I)).GT.EPS(I))J=1
```

```
0082        50 S(I)=X(I)-W(I)
0083           IFL=0
0084           CALL RESID(M,N,S,W,IFL)
0085           IF(IFL.EQ.0)GOTO51
0086           Y=.1
0087           DS=0.
0088           GOTO52
0089        51 CONTINUE
0090           IR=IR+1
0091           CALL MC03AS(W(1),W(2),W(1),W(2),0.,SSP,M,4)
0092           DS=SS-SSP
0093           IF(J.EQ.0.OR.DQ.LE.0..OR.IR.EQ.MAXFN)GOTO80
0094           IF(DS.GE.RHC*DQ)GOTO60
0095           Y=.5
0096           Z=2.*VW-DS
0097           IF(Z.GT.0.)Y=VW/Z
0098           IF(Y.GT..5)Y=.5
0099           IF(Y.LT..1)Y=.1
0100        52 CONTINUE
0101           IF(Q.NE.0.)GOTO58
0102           Y=2.*Y
0103           DO 521 I=1,N
0104       521 A(I,I)=1./A(I,I)
0105           DO 522 I=2,N
0106           II=I-1
0107           DO 522 J=1,II
0108           CALL MC03AS(A(J,J),A(J,J+1),A(J,I),A(J+1,I),0.,Z,I-J,7)
0109       522 A(J,I)=Z*A(I,I)
0110           DO 53 I=1,N
0111           DO 53 J=I,N
0112           CALL MC03AS(A(I,J),A(I,J+1),A(J,J),A(J,J+1),0.,Z,N-J+1,4)
0113        53 A(I,J)=ABS(Z)
0114           Q=0.
0115           TR=0.
0116           DO 54 I=1,N
0117           TR=TR+A(I,I)*D(I)
0118           Z=0.
0119           DO 55 J=1,I
0120        55 Z=Z+A(J,I)*T(J)
0121           IF(I.EQ.N)GOTO56
0122           II=I+1
```

```
0123            DO 57 J=I1,N
0124         57 Z=Z+A(I,J)*T(J)
0125         56 CONTINUE
0126            Z=Z*T(I)
0127            IF(Z.GT.Q)Q=Z
0128         54 CONTINUE
0129            IF(TR.LT.Q)Q=TR
0130            Q=1./Q
0131            QC=Q
0132         58 CONTINUE
0133            Q=Q/Y
0134            IF(DS)35,35,70
0135         60 CONTINUE
0136            IF(DS.LE.SIG*DQ)GOTO70
0137            Q=Q*.5
0138            IF(Q.LT.QC)Q=0.
0139         70 CONTINUE
0140            SS=SSP
0141            DO 71 I=1,N
0142         71 X(I)=S(I)
0143            DO 72 I=1,M
0144         72 R(I)=W(I)
0145            CALL LSQ(M,N,X,R,A,V)
0146            GOTO20
0147         80 CONTINUE
0148            IF(DS.LE.0.)GOTO83
0149            SS=SSP
0150            DO 81 I=1,N
0151         81 X(I)=S(I)
0152            DO 82 I=1,M
0153         82 R(I)=W(I)
0154         83 CONTINUE
0155            IF(IPRINT.EQ.0)RETURN
0156            PRINT 1001,IT,IR
0157            PRINT 1002,SS
0158            PRINT 1002,(X(I),I=1,N)
0159            IF(IPRINT.LT.0)RETURN
0160            PRINT 1002,(R(I),I=1,M)
0161            RETURN
0162            END
```

Appendix 2

The specification sheet for the FORTRAN subroutine VA07A

1. Purpose

To find a local minimum of a sum of squares of m non-linear functions of n variables, that is to

$$\text{minimize} \sum_{i=1}^{m} [r_i(x_1,x_2,\ldots,x_n)]^2$$

Typically the r_i might be the residuals of a non-linear least squares data fitting problem, as in the example of section 8. The user must be able to calculate the functions r_i and the partial derivatives $\partial r_i/\partial x_j$ for all i,j : this information is presented to VA07A by two user subroutines as described in section 3. The method is described by R. Fletcher (1971), "A modified Marquardt subroutine for non-linear least squares", Harwell report, AERE R.6799; it is iterative so that an initial approximation to $x_1,x_2,\ldots,x_n$ must be supplied. The method allows the imposition of constraints in a limited way as described in section 6. It may also be possible to improve the performance of the method by scaling the variables in a realistic way (see section 5). An automatic choice can be made by VA07A or this can be overriden by the user if desired.

2. Argument List

SUBROUTINE VA07A (RESID,LSQ,M,N,X,R,SS,A,D,EPS,IPRINT,MAXFN,MODE)

RESID)
LSQ) identifiers of the user subroutines - see section 3.

M an INTEGER set to the number of functions m. M must be $\geq$ N.

N an INTEGER set to the number of variables n. N must be $\geq$ 2.

X a REAL array of N elements, set so that X(I) is the initial approximation to x_i. The best approximation to the minimum which is found will overwrite X on exit from VA07A.

R a REAL array of M elements, which is such that on exit from VA07A R(I) contains the value of the residual $r_i(x_1,x_2,\ldots,x_n)$ corresponding to the X above. R need not be set by the user on entry to VA07A.

SS a REAL variable which on exit from VA07A contains the sum of squares of the R(I) above. SS need not be set by the user on entry to VA07A.

A a REAL array of at least N^2 elements, used by VA07A as working space.

D a REAL array of N elements, only to be set if MODE = 3, and which controls scaling (see section 5).

EPS — a REAL array of N elements, set so that EPS(I) is the absolute accuracy to which x_i should approximate the solution.

IPRINT — an INTEGER which controls the frequency and amount of printing - see section 4.

MAXFN — an INTEGER giving an upper limit to the number of times RESID is called - see section 3.

MODE — an INTEGER which governs the method of scaling the variables - see section 5.

3. User subroutines

The user must provide two subroutines, one to calculate the residuals, and the other to calculate derivatives. The user may choose any identifier for these subroutines, and these must be supplied in the calling sequence. An EXTERNAL statement must also appear in the user's MAIN program. (see the example in section 8). These subroutines should be written as follows.

(a)
```
SUBROUTINE  RESID (M,N,X,R,IFL)
DIMENSION  X(1),R(1)
-----
```
statements to evaluate $r_i(x_1,x_2,\ldots x_n)$ for i=1,2,...,m and store them in R(I), I=1,2,...,M. $x_1,x_2,\ldots,x_n$ are given in X(1), X(2),..., X(N)
```
----
RETURN
END
```

If for any reason, one or more of the r_i cannot be evaluated with the given $x_1,x_2,\ldots,x_n$ (for example if overflow or negative square root would occur), then the INTEGER variable IFL should be set to 1 and a RETURN given. This feature can also be used to impose constraints on the variables in a limited way - see section 6.

(b)
```
SUBROUTINE  LSQ(M,N,X,R,A,V)
DIMENSION  X(1),R(1),A(N,1),V(1)
------
```
statements to set up the coefficients of the least squares normal equations, that is to evaluate

$$A_{ij} = \sum_{k=1}^{m} \frac{\partial r_k}{\partial x_i} \frac{\partial r_k}{\partial x_j} \qquad \text{for } i=1,2,\ldots,n \text{ and for } j=1,2,\ldots,i,$$

and

$$v_i = \sum_{k=1}^{m} r_k \frac{\partial r_k}{\partial x_i} \qquad \text{for } i=1,2,\ldots,n,$$

where r_k and $\partial r_k/\partial x_i$ are evaluated for $x_1,x_2,\ldots,x_n$ as given in X(1),X(2),...,X(N). In fact the values r_k, k=1,2,...,m will already have been evaluated for these $x_1,x_2,\ldots,x_n$ in an immediately previous call of RESID, and these values are available in R(1),R(2),....,R(M)
```
------
RETURN
END
```

One way of programming LSQ is to declare an array of size MxN (DR(M,N) say) and to set DR(K,I) equal to $\partial r_k/\partial x_i$ for all K=1,2,...,M and all I=1,2,...,N. Then A and V can be evaluated using the statements

```
      DO 1 I=1,N
      CALL MC03AS (D(1,I), D(2,I), R(1), R(2), 0., V(I), M, 4)
      DO 1 J=1,I
    1 CALL MC03AS (D(1,I), D(2,I), D(1,J), D(2,J), 0., A(I,J), M,4)
```

see the example of section 8. It may be possible to do this more efficiently without using MN storage locations for DR - however note that A is overwritten by VA07A so that any constant elements in A must be reset. Because A is a symmetric matrix, <u>only the lower triangle need be set</u>.

In most problems, the calculation of derivatives in LSQ will involve terms (e.g. cosines, exponentials, etc.) which have already been calculated in RESID. It is usually very inefficient to recalculate such expressions and they should be passed from RESID to LSQ via a COMMON block - see the example in section 8. Alternatively LSQ can be written as if it were a secondary entry point to RESID.

For the sake of efficiency LSQ is only called if the sum of squares of residuals evaluated by RESID is an improvement on the best previously obtained. However in practice most calls of RESID are followed by a call of LSQ. This should be taken into account when setting the parameter MAXFN.

4. Printing

Printing starts on a new page with the text ENTRY TO VA07A. At the beginning of the first iteration and on every subsequent |IPRINT| iterations the numbers

```
IT   IR
SS
X(1),X(2),...,X(N)               (8 to a line)
V(1),V(2),...,V(N)               (     "      )
R(1),R(2),...........,R(M)       (     "      )
```

are printed as shown. IT is the previous no. of iterations, IR is the no. of calls of RESID, X is current best approximation, R and SS are the corresponding residuals and sum of squares, and V is the corresponding quantity defined in section 3(b), where in fact 2V is the gradient vector of the sum of squares. The same information is printed out on exit from VA07A, excepting that V is not given as it is not usually available.

Exceptions to the above occur if IPRINT=0, when none of the above printing takes place, and if IPRINT < 0 when printing of R(1),...,R(M) is suppressed. Furthermore diagnostics may be produced in certain error situations.

5. Scaling the variables

At each iteration the equations

$$(A + \lambda D)\underset{\sim}{\delta} = -\underset{\sim}{v}$$

are solved to obtain a correction $\underset{\sim}{\delta}$ to the current approximation $\underset{\sim}{x}$. D is a constant diagonal matrix with $D_{ii} > 0$, and different choices of D correspond to different prescalings of the variables. D_{ii} is represented by the Ith element of the parameter D in the calling sequence of VA07A and may be specified in different ways by setting the parameter MODE. The following are permitted.

MODE = 1 (the normal setting): D(I) for I=1,2,...,N is set automatically by VA07A to A(I,I), where A is the matrix calculated by LSQ from the given initial approximation, or to 1 if exceptionally A(I,I)=0. This choice is described further in R.6799

MODE = 2 D(I) is set automatically by VA07A to 1 (corresponding to no change of scale).

MODE = 3 D(I) is set by the user through the parameter D in the parameter list.

If MODE = 1 or 2, therefore, no user action is required as regards scaling.

6. Constraints

When a RETURN is given in RESID with IFL=1, as described in section 3(a), then the iteration is repeated with a larger value of λ, causing a smaller correction to be made to the variables. This feature can be used to impose constraints in a limited and simple minded way. It is merely necessary in RESID to check whether the values $x_1,x_2,\ldots,x_n$ violate any of the constraints on the variables, in which case the INTEGER variable IFL is set to 1 and a RETURN is given. This device is illustrated in the next section, and is worth trying when an unconstrained minimum is expected to exist, although success is by no means guaranteed.

7. General

Use of COMMON - none
Private workspace - see under restrictions in use below
Other routines - calls MC03AS (double length scalar product) and MA10A (Choleski method for linear equations)
System dependence - none
Date of routine - April 1971
Restrictions - VA07A is restricted to N=25 and M=200 directly. However these restrictions can be circumvented, when single length is being used, by adding the following named COMMON statements to the users MAIN program.

(i) to increase the N limit (to $\bar{N}$ say), include

COMMON/VA07B/S($\bar{N}$)
COMMON/VA07C/T($\bar{N}$)
COMMON/VA07D/U($\bar{N}$)
COMMON/VA07E/V($\bar{N}$)

(ii) to increase the M limit (to $\bar{M}$ say), include

COMMON/VA07F/W($\bar{M}$)

To increase both limits, include both sets of named COMMON statements.

The changes in the double length version are as above but with the addition of D to the name, that is
COMMON/VA07BD/S($\bar{N}$)
etc.

8. An Example

Consider the problem of fitting the data $y(t_1)$, $y(t_2)$, ..., $y(t_{25})$ by a function of the form

$$f(t) = a + bt + c \exp(-\tfrac{1}{2}(t-d)^2/e^2),$$

where a,b,c,d and e are parameters to be determined. The problem can be posed as that of choosing a,b,...,e so as to

$$\text{minimize} \sum_{i=1}^{25} [r_i(a,b,\ldots,e)]^2$$

where

$$r_i(a,b,\ldots,e) = a + bt_i + c \exp(-\tfrac{1}{2}(t_i-d)^2/e^2) - y(t_i).$$

Thus the problem has 25 residuals, 5 variables, and $t_1,t_2,\ldots,t_{25}$ and $y(t_1)$, $y(t_2)$,...$y(t_{25})$ are given data. The required partial derivatives are easily obtainable, namely

$$\frac{\partial r_i}{\partial a} = 1 \qquad \frac{\partial r_i}{\partial b} = t_i \qquad \frac{\partial r_i}{\partial c} = \exp(\ldots\ldots)$$

$$\frac{\partial r_i}{\partial d} = \frac{c(t_i-d)}{e^2} \exp(\ldots.) \qquad \frac{\partial r_i}{\partial e} = \frac{c(t_i-d)^2}{e^3} \exp(\ldots.).$$

Note how the exponentials which occur in calculating r_i also occur in the calculation of the derivatives. Finally r_i will overflow if $e = 0$ and in fact a minimum subject to the constraint $e > 0$ is required.

The MAIN program for this problem is as follows

```
REAL A(25), D(5), X(5), EPS(5), R(25)
COMMON Y(25), T(25), EX(25), DR(25,5)
EXTERNAL  GAUSSR, GAUSSD
     ----
     statements to read y(ti) --> Y(I), ti --> T(I), to set DR(I,1)=1 and
     DR(I,2) = T(I), for I=1,2,...,25; also to set initial approximations to
     a,b,...,e into X(1), X(2), ..., X(5) and the respective tolerances into
     EPS(1), EPS(2), ..., EPS(5).
     ----
CALL VA07A (GAUSSR, GAUSSD, 25, 5, X, R, SS, A, D, EPS, 1, 100, 1)
STOP
END
```

and the user subroutines are

```
      SUBROUTINE GAUSSR (M, N, X, R, IFL)
      DIMENSION V(1), R(1)
      COMMON Y(25), T(25), EX(25), DR(25,5)
      IF(X(5).GT.O.) GOTO 3
      IFL=1
      RETURN
    3 CONTINUE
      DO 1 I=1,M
      EX(I) = EXP(-.5*((T(I)-X(4))/X(5))**2)
    1 R(I) = X(1) + X(2)*T(I) + X(3)*EX(I)-Y(I)
      RETURN
      END

      SUBROUTINE GAUSSD(M, N, X, R, A, V)
      DIMENSION X(1), R(1), A(N,1), V(1)
      COMMON Y(25), T(25), EX(25), DR(25,5)
      DO 1 I=1,M
      DR(I,3) = EX(I)
      DR(I,4) = X(3)*(T(I)-X(4))*EX(I)/X(5)**2
    1 DR(I,5) = X(3)*(T(I)-X(4))**2*EX(I)/X(5)**3
      DO 2 I=1,N
      CALL MC03AS (DR(1,I), DR(2,I), R(1), R(2), O., V(I), M, 4)
      DO 2 J=1,I
    2 CALL MC03AS (DR(1,I), DR(2,I), DR(1,J), DR(2,J), O., A(I,J), M, 4)
      RETURN
      END
```

Appendix 3

Specification sheets for the FORTRAN subroutines MC03AS and MA10A which are called by VA07A

1. PURPOSE

To evaluate the sum of an inner product and a constant using double-length accumulation to minimize rounding errors.

i.e. to evaluate

$$\pm x \pm \sum_{k=1}^{N} a_k b_k$$

The vectors a, b can be stored in any regular fashion. This routine is written in IBM 360/ ASSEMBLER LANGUAGE.

2. ARGUMENT LIST

SUBROUTINE MC03AS (A(I),A(J),B(K),B(L),X,SUM,N,IFLAG)

All arguments except SUM must be set by the calling program.

A an array containing the elements of the vector a. A(I) is that member of A containing the value of a_1. A(J) is the member of A containing a_2. Subsequent members of the vector a are stored in A at equal intervals i.e. a_M is contained in A(I+(M-1)*(J-I)).

B an array containing the vector b. B(K), B(L) are the elements of this array containing the first and second elements of the vector b as for A, a above, i.e. b_M is stored in B(I+(M-1)*(K-L)).

X is the constant to be added to the inner product.

N the number of elements in each vector a, b

SUM is the required sum, and is set by MC03AS.

IFLAG an integer parameter which specifies the combination of signs required, and also specifies whether the result in SUM is to be rounded (r) or unrounded (u).

IFLAG	SIGN OF X	SIGN OF Σ	ROUNDING
0	+	+	u
1	+	−	u
2	−	+	u
3	−	−	u
4	+	+	r
5	+	−	r
6	−	+	r
7	−	−	r

3. METHOD

The sum is accumulated in double-length floating point register. No results are stored until the accumulation is complete.

1. Purpose

This subroutine inverts a symmetric positive definite matrix A or solves the equations Ax = b with one or more right hand sides, or does both of these operations. Only the elements of the lower triangle of A need be defined, and if the matrix is not strictly positive definite the routine will invariably do an error return.

The equations are of the form

$$\sum_{J=1}^{M} A(I,J).\ X(J,K) = B(I,K) \quad I = 1, 2, \ldots, M \quad K = 1, 2, \ldots, N$$

Therefore A is an M x M matrix and there are N right hand sides.

2. Argument List

SUBROUTINE $MA_{10}A$ (A, B, M, N, NR, M1, IA, IB)

A is a two dimensional array containg the elements of the matrix. Only the elements A(I,K), I $\geqslant$ J need be set on entry to the routine. If the inverse has been asked for it will be found in A on exit, unless the matrix is not positive definite, in which case A will contain rubbish.

B is a two dimensional array containing the right hand sides of the equations. If equation solving has been asked for, the solutions will over write B; X(I,K) will be found in B(I,K).

M is the number of equations.

N is the number of right hand sides.

NR is a parameter which will be set to zero on exit if the inversion has been completed. If the matrix is not positive definite NR will be set to one.

M1 is a parameter that determines the operations that will be carried out by the routine. See next paragraph for details.

IA and IB define the first dimension of the arrays A and B, so that if the dimension statement of the calling routine is

DIMENSION A(α,), B(β,) then set IA = α and IB = β.

3. Controlling the Routine

(a) To carry out inversion only, enter with N $\leqslant$ 0 and M1 > 0.

(b) To solve equations only, enter with N > 0 and M1 = 0.

(c) To invert and solve, enter with N > 0 and M1 > 0

(d) After doing (b) it is possible to re-enter the routine for the purpose of finishing the inversion of A (which will have been partly done by the equation-solving process). To do this, set M1 < 0 for the re-entry.

4. Output

If the inversion cannot be completed a message will be printed.

5. General

The routine does not use common or auxiliary storage.

6. Other routines

$MC_{03}AS$ is called by this routine and therefore must be loaded with it.

7. Method

Symmetric Choleski decomposition is used to find the lower triangular matrix L for which $LL^T = A$.