A Free Nonlinear Estimation and Equation Solving Subprogram for VBA

Edward M. Rosen

EMR Technology Group
Chesterfield, Mo. 63017

Introduction

Clough [1] observed that his students preferred Excel/VBA over Mathcad[2] and Matlab [3] . This observation led the author to examine what software was freely available for VBA.

A very comprehensive linear system package is available from Volpi [4] as discussed in Rosen[5] . The package (MATRIX 2.3) is available as an Excel add-in. Though designed for spreadsheet use, the functions and subprograms written in VBA can be used in a VBA program, some directly and others with slight modifications. Routines are available for simultaneous linear equations, eigenvalues and eigenvectors of symmetrical matrices, singular value decomposition as well as many other computations of linear algebra.

Another free program is the SOLVER program in EXCEL. It can be executed both on the spreadsheet or in VBA[6]. The author has found that SOLVER appears to do well on some problems but gives mixed results on others .

The technology of XSOLVE (the subject of this paper) , though dated[7] , has been extensively tested and is easily configured to solve a wide range of problems. There is a large literature and programs on VBA[8] but there appears to be a dearth of free, easily accessible VBA programs that can solve both sets of nonlinear equations as well as regression problems.

The Algorithm of XSOLVE

We seek to find a solution to the n equations in the k unknowns X

$$
\begin{aligned}
&z_i\,(x_1, x_2, \ldots\ x_k) = y_1\\
&z_2(x_1, x_2, \ldots\ x_k) = y_2\\
&\quad\vdots\\
&z_n(x_1, x_2 \ldots.\ x_k) = y_n
\end{aligned}
\qquad \text{(A.1)}
$$

By a solution we mean an X such that

$$\phi = \sum_{i=1}^{i=n} (z_i - y_i)^2 \qquad \text{(A.2)}$$

is a minimum. The functional form of z_i is assumed to be known and the y_i are constants. Linearizing Equation (A.2) about a base point X_o there results

$$P\,\Delta X = -F \qquad \text{(A.3)}$$

where

P = an n x k matrix whose elements are

$$\frac{\partial z_i}{\partial x_j} \qquad I = 1, 2, \ldots,n \qquad j = 1, 2,\ldots, k$$

F = an n x 1 vector defined by

$$f_i = z_i - y_i \qquad I = 1, 2, \ldots, n$$

ΔX = n x 1 vector defined by

$$X = X_o + \Delta X \qquad (A.4)$$

For equation solving (n = k) Equation (A.3) is the Newton-Raphson method and P is the Jacobian. In this case if P is not singular the ΔX could be calculated directly from Equation (A.3).

Derivatives for the P matrix may be calculated analytically. If they are calculated numerically then 0.001 times the larger of the starting point + 0.01 or the current point is used as a perturbation in a forward difference. (At the upper bound of a variable a backward difference is used.)

Multiplying Equation (A.3) by P^T there results

$$P^T P \Delta X = - P^T F \qquad (A.5)$$

Letting $A = P^T P$ and noting that

$$-P^T F = -1/2 \frac{\partial \phi}{\partial X} = G \qquad (A.6)$$

Equation (A.5) becomes

$$A \Delta X = G \qquad (A.7)$$

which is generally known as the "Gauss-Newton" method. The G vector is ½ the negative gradient vector of ϕ .

After A is calculated (which must be positive semi-definite) if a diagonal element is such that 1 + diagonal element = 1. that element and all the other elements in that row as well as the right hand side of Equation (A.7) are set to 0. (A small diagonal element implies there is no effect of the parameter on any of the equations.) Equation (A.7) is then scaled according (for $a_{ii} \neq 0$.)

$$A^* = (aij^*) = \frac{a_{ij}}{\sqrt{a_{ii}}\sqrt{a_{jj}}} \qquad (A.8)$$

and

$$G^* = (g_i^*) = \frac{g_i}{\sqrt{a_{ii}}}$$

from which Equation (A.7) becomes

$$A^* \Delta X^* = G^* \qquad (A.9)$$

Marquardt [9] modified Equation (A.9) to be

$$(A^* + \lambda I)\, \Delta X^* = G^* \qquad (A.10)$$

which is the necessary condition for the minimum of ψ where

$$\psi = \Delta X^T D\, \Delta X + \frac{1}{\lambda} [\, (A\Delta X - G)^T (A\Delta X - G)] \qquad (A.11)$$

and

$$D = \text{diag}\,(A^T A) \qquad (A.12)$$

Thus,

$$A^* = D^{-1/2}\, A\, D^{-1/2} \text{ and } G^* = D^{-1/2}\, G \qquad (A.13)$$

From Equation (A.10)

$$\Delta X^* = (A^* + \lambda I)^{-1}\, G^* \qquad (A.14)$$

where

$$\Delta x_i = \frac{\Delta x_i^*}{\sqrt{a_{ii}}} \qquad (A.15)$$

Equation (A.14) tells us how to search for a solution by going from base point to base point via Equation (A.4). When λ is small Δ X becomes the Gauss- Newton search vector. When λ is large Δ X becomes the steepest descent search vector. As λ is varied between these two ranges a curved path is traced out between the Gauss-Newton and the steepest descent directions. Since both directions point in a direction to always decrease ϕ locally the angle between the Gauss-Newton and steepest decent direction must always be less than 90 degrees.

Equation (A.14) is solved by elimination . If it is found to be singular Δ X* is computed by finding the eigenvalues (E) of A* (which is symmetric and positive semi-definite) from subprogram JACOBI:

$$A^* = SES^T \qquad (A.16)$$

Then

$$\Delta \bar{X}^* = (E + \lambda I)^{-1} \bar{G}^* \qquad \text{(A.17)}$$

where

$$\Delta \bar{X}^* = S^T \Delta X^* \qquad \text{(A.18)}$$

$$\bar{G}^* = S^T G^* \qquad \text{(A.19)}$$

In solving Equation (A.17) if $e_i \leq 10^{-8}$

then $\Delta \bar{x}_i^*$ is set to zero. Otherwise $\Delta \bar{x}_i^* = \dfrac{\bar{g}_i^*}{e_i + \lambda}$ (A.20)

This avoids the possibility of taking large steps when there exists a local singularity of A* especially as $\lambda \rightarrow 0$. If a predicted step falls outside the bounds it will be projected back onto the boundary.

For the case when equations are being solved (n=k) the equations are scaled at each new base point by dividing each equation by the corresponding square root of the diagonal element of $P P^T$. The criterion for movement is whether or not a trial point has a scaled sum of squares (calculated internally) smaller than the base scaled sum of squares.

The XSOLVE Subprogram

The XSOLVE VBA program has been translated from a listing of an older FORTRAN program. There are a number of required parameters (which gives it flexibility) but once these have been set they can be easily used as a template for other problems.

Figure 1 describes the routines used by XSOLVE. MCP is the macro which calls XSOLVE. XSOLVE in turn calls eigenvalue and eigenvector routines. SUB's for these calculations are taken from the MATRIX routines of Reference 4. Zval is an optional subprogram which evaluates the computed functions.

Figure 1 XSOLVE and Associated Routines

Routine	Source
MCP	Macro calling XSOLVE and optionally Zval
Zval	Optional sub to evaluate functions
XSOLVE	Modified BSOLVE Routine from Henley and Rosen
Jacobi	Written to integrate Matrix eigenvalue and eigenvector routines
XmatEigenvalue_Jacobi	Sub modification of Matrix function Mateigenvalue_Jacobi
XmatEigenfunction_Jacobi	Sub modification of Matrix function Mateigenfunction_Jacobi
Mat_Jacobi_Find_Max	Sub from Matrix
MAX	Private Function from Matrix
Matcopy	Sub from Matrix

To use XSOLVE the following setup and calling sequence is required:

1. Set K, N the number of unknowns and the number of equations.
2. Set Dim NDATA(26), DATA(16), OUTPUT(6)
   Redim X(2*K), XV(K), XMAX(K), XMIN(K), Y(N), Z(2*N),
   PJ(N),P(K*(N+2) +N), A(K,K+2), AC(K,K+2)
3. Set NDATA(1) = 1
4. Set DATA (I) = 1 to 4 (Generally all 0)
5. Set XV(I) = 1 to K (0, constant , 1 Numerical Derivatives, -1 Analytical Derivatives)
6. Set XMAX (I) = 1 To K, (max values of unknowns)
7. Set XMIN(I) = 1 To K (min values of unknowns)
8. Set Y(I) = 1 to N (values of desired values)
9. Set Initial values of X(I) = 1 to K (unknowns)
10. Evaluate computed values (Z(I) = 1 to N) at initial values of X
11. Call XSOLVE (K, N, NDATA, DATA, X, XV, XMAX, XMIN, Y, Z, PJ, OUTPUT, P, A, AC)
12. Test NDATA(2) to see if the function must be evaluated, the derivative is to be evaluated or a new base point has been found.
13. If a new base point has been found test NDATA(3) to see if the search has been terminated. If it has not recall XSOLVE.

A full description of the values in the calling sequence is given in the Appendix.

Example Problems

Example 1 **"**This problem[10] was found to be difficult for some very good algorithms". It (**MGH09**) is classified as higher level of difficulty.

The problem is to find parameters b1, b2, b3 and b4 to best fit:

y = b1*(x**2+x*b2)/(x**2+x*b3 +b4)

y	x
1.957000E-01	4.000000E+00
1.947000E-01	2.000000E+00
1.735000E-01	1.000000E+00
1.600000E-01	5.000000E-01
8.440000E-02	2.500000E-01
6.270000E-02	1.670000E-01
4.560000E-02	1.250000E-01
3.420000E-02	1.000000E-01
3.230000E-02	8.330000E-02
2.350000E-02	7.140000E-02
2.460000E-02	6.250000E-02

NIST Certified Values and XSOLVE Values.

Parameter	NIST	XSOLVE
b1	1.928069E-01	1.928069E-01
b2	1.912823E-01	1.912825E-01
b3	1.230565E-01	1.230565E-01
b4	1.360623E-01	1.360624E-01
Sum squares	3.0750560E-04	3.0750560E-04

Starting Values: (25 39 41.5 39)

Table 1 is the VBA input file for this problem.

Example 2 This is POLYMATH[11] problem Twoeq1- Pipe diameter calculation for a specified pressure drop. It is two equations in the two unknowns D and fF:

```
f(D) = -dp/rho+2*fF*v*v*L/D
f(fF) = if (Re<2100) then (fF-16/Re) else (fF-1/(4*log(Re*(fF)^(1/2))-0.4)^2)
dp=103000
L=100
T=25+273.15
Q=0.0025
pi=3.1416
rho=46.048+T*(9.418+T*(-0.0329+T*(4.882e-5-T*2.895e-8)))
vis=exp(-10.547+541.69/(T-144.53))
v=Q/(pi*D^2/4)
kvis=vis/rho
Re=v*D/kvis

D(0)=0.04
fF(0)=.001
```

XSOLVE Solution (starting with (0.1, 0.1) :

D = 0.0389653
fF = 0.00459053 Residual sum of squares: 2.70229E-15

This corresponds to the POLYMATH solution.

The log is a base 10 log. In VBA only the natural logarithm is available so the natural log must be multiplied by 0.4342944. In the spreadsheet log is base 10.

Table 2 is the input file for this problem.

Conclusions

The complete VBA files for the two example problems may be obtained by contacting the author at edwardmemrose@gmail.com. Output for the example problems is generated onto the spreadsheet.

References:

1. Clough, D. E., "ChE's Teaching Introductory Computing to CHE Students – A Modern Computing Course with Emphasis on Problem Solving and Programming", Proceedings of the 2002 American Society for Engineering Education Annual Conference & Exposition

2 . MathCad: http://www.ptc.com/products/mathcad/

3. MatLab: http://www.mathworks.com/

4. http://digilander.libero.it/foxes/SoftwareDownload.htm

5. Rosen, E. M. "Topics in Excel's VBA 2003: Functions and Add-Ins" CACHE News, Winter 2005.

6. "How to Create Visual Basic Macros Using Excel Solver in Excel 97" http://peltiertech.com/Excel/SolverVBA.html

7. Press et al., *Numerical Recipes, 3rd Edition*, Cambridge University Press, 2007

8. Numerical Recipes in VB for Excel & Access rnfc.org/ivey/2009/08/10/numerical -recipes

9. Marquardt, D. W, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters" Journal of the Society for Induatrial and Applied Mathematics, Vol II, No 2 June , 1963.

10. NIST Data set MGH09: http://www.itl.nist.gov/div898/strd/nls/data/LINKS/DATA/MGH09.dat

11. POLYMATH: http://www.polymath-software.com/library/nle/Twoeq1.htm

Table 1 Example 1 Input File

```
 Sub MCP      ********************************************************

' Set For XSOLVE CALL
' Input File for MCH09

  Dim KK, NN, I, I1, J As Integer
  KK = 4
  NN = 11

  Dim DATA(16), OUTPUT(6) As Double
  Dim NDATA(26) As Integer

  ReDim X(NN) As Double

  ReDim A(2 * KK), AV(KK), AMAX(KK), AMIN(KK), Y(NN), Z(2 * NN), _
        PJ(NN), P(KK * (NN + 2) + NN), AB(KK, KK + 2), AC(KK, KK + 2) As Double

   NDATA(1) = 1
  For I1 = 1 To 5
    DATA(I1) = 0#
  Next I1

  DATA(4) = 0.00000001

  For I1 = 1 To KK
    AV(I1) = -1#
    AMAX(I1) = 50#
    AMIN(I1) = 0#
  Next I1

' Initial Values of Unknowns
      A(1) = 25
      A(2) = 39
      A(3) = 41.5
      A(4) = 39

     X(1) = 4
     X(2) = 2
     X(3) = 1
     X(4) = 0.5
     X(5) = 0.25
     X(6) = 0.167
     X(7) = 0.125
     X(8) = 0.1
     X(9) = 0.0833
```

```
    X(10) = 0.0714
    X(11) = 0.0625

    Y(1) = 0.1957
    Y(2) = 0.1947
    Y(3) = 0.1735
    Y(4) = 0.16
    Y(5) = 0.0844
    Y(6) = 0.0627
    Y(7) = 0.0456
    Y(8) = 0.0342
    Y(9) = 0.0323
    Y(10) = 0.0235
    Y(11) = 0.0246
Q20:
 '  Evaluate Z Vector

  For I = 1 To NN
        Z(I) = (A(1) * (X(I) ^ 2 + X(I) * A(2))) / (X(I) ^ 2 + X(I) * A(3) + A(4))
  Next I
  GoTo Q40
Q30:
 ' Evaluate Derivatives
    J = NDATA(4)
    For I = 1 To NN
     If J = 1 Then PJ(I) = (X(I) ^ 2 + X(I) * A(2)) / (X(I) ^ 2 + X(I) * A(3) + A(4))
     If J = 2 Then PJ(I) = A(1) * X(I) / (X(I) ^ 2 + X(I) * A(3) + A(4))
     If J = 3 Then PJ(I) = -A(1) * X(I) * (X(I) ^ 2 + X(I) * A(2)) / (X(I) ^ 2 + X(I) * A(3) + A(4)) ^ 2
     If J = 4 Then PJ(I) = -A(1) * (X(I) ^ 2 + X(I) * A(2)) / (X(I) ^ 2 + X(I) * A(3) + A(4)) ^ 2
     Next I
Q40:
   Call XSOLVE(KK, NN, NDATA, DATA, A, AV, AMAX, AMIN, Y, _
        Z, PJ, OUTPUT, P, AB, AC)

  If NDATA(2) = 0 Then GoTo Q20 Else
  If NDATA(2) = 1 Then GoTo Q30 Else
  If NDATA(3) > 0 Then GoTo Q40

'  OUTPUT

  For I1 = 1 To 4
    Cells(20 + I1, 1) = A(I1)
  Next I1

  Cells(26, 1) = OUTPUT(1)

End Sub        ***************************************************
```

Table 2 Input File for Example 2 Twoeq1

```
 Sub MCP()   ******************************************************

' Set For XSOLVE CALL
' Problem Twoeq1 from Polymath

  Dim KK, NN, I1  As Integer
  KK = 2
  NN = 2

  Dim DATA(16), OUTPUT(6) As Double
  Dim NDATA(26) As Integer

  ReDim A(2 * KK), AV(KK), AMAX(KK), AMIN(KK), Y(NN), Z(2 * NN), _
        PJ(NN), P(KK * (NN + 2) + NN), AB(KK, KK + 2), AC(KK, KK + 2) As Double

   NDATA(1) = 1

  For I1 = 1 To 5
    DATA(I1) = 0#
  Next I1

  DATA(4) = 0.00000001

  For I1 = 1 To KK
    AV(I1) = 1#
  Next I1

    AMAX(1) = 0.2
    AMAX(2) = 0.2

    AMIN(1) = 0.00001
    AMIN(2) = 0.00001

    Y(1) = 0
    Y(2) = 0

'  Start Values

        A(1) = 0.1
        A(2) = 0.1
Q20:
'  Evaluate Z Vector
    Call Zval(A, Z)

Q40:
    Call XSOLVE(KK, NN, NDATA, DATA, A, AV, AMAX, AMIN, Y, _
```

```
        Z, PJ, OUTPUT, P, AB, AC)

  If NDATA(2) = 0 Then GoTo Q20 Else
  If NDATA(3) > 0 Then GoTo Q40

'  OUTPUT
Q60:

  For I1 = 1 To KK
    Cells(I1 + 3, 2) = A(I1)
    Cells(I1 + 3, 3) = Z(I1)
  Next I1
  Cells(7, 2) = OUTPUT(1)

End Sub     ******************************************************

‘ Zval Subprogram    ************************************************

Sub Zval(A, Z)

Dim dp, L, T, Q, Pi, rho, vis, v, kvis, Re, D, fF, xx1, xx2 As Double

      D = A(1)
      fF = A(2)
      dp = 103000
      L = 100
      T = 25 + 273.15
      Q = 0.0025
      Pi = 3.1416
      rho = 46.048 + T * (9.418 + T * (-0.0329 + T * (0.00004882 - T * 0.00000002895)))
      vis = Exp(-10.547 + 541.69 / (T - 144.53))
      v = Q / (Pi * (D ^ 2) / 4)
      kvis = vis / rho
      Re = v * D / kvis

      Z(1) = -dp / rho + 2 * fF * v * v * L / D
      xx1 = fF - 16 / Re
      xx2 = fF - 1 / (4 * 0.4342944 * Log(Re * (fF) ^ (1 / 2)) - 0.4) ^ 2
      If Re < 2100 Then Z(2) = xx1
      If Re >= 2100 Then Z(2) = xx2

End Sub     ************************************************************
```

Appendix XSOLVE Calling Sequence

The Subprogram is called by:

```
Dim     NDATA(26), DATA(16), OUTPUT (6)

Redim   X(2*K), XV(K), XMAX(K), XMIN(K), Y(N), Z(2*N),
        PJ(N),P(K*(N+2) +N), A(K,K+2), AC(K,K+2)

Call XSOLVE   (K, N, NDATA, DATA, X, XV, XMAX, XMIN, Y, Z, PJ, OUTPUT, P, A, AC)
```

where

K = the number of independent variables (K ≥ 1) (input)
N = The number of equations to be solved (N ≥ K) (input)

NDATA – an integer storage vector

NDATA(1) - used to control the sequence of operations internally. Must be set to 1 on initial entry into XSOLVE. It is reset by XSOLVE after initial entry. (input and output)

Value	Meaning	
1	Must be set on initial entry	(on input)
2	Analytical derivative mode	(on output)
3	Numerical derivative mode	(on output)
4	Search mode	(on output)
5	New base point mode	(on output)
-1	Search cannot continue	(on output)

NDATA(2) - used to determine if function or derivative needs to be calculated or if a new base point is being reported. It is set by XSOLVE (output)

Value	Meaning
0	Calculate the function, Z(X)
1	Calculate the derivative vector of the function with respect to X(J) where J is given in NDATA(4) and put into the PJ vector.
-1	A new base point has been found. (The starting point is a new base point). Examine NDATA(3) for convergence.

NDATA(3) – indicates status of search at new base point.

Value	Meaning
>0	Gives the number of variables not satisfying convergence criterion where

$$\frac{|\Delta x_i|}{|\Delta x_i|+\tau} \leq \varepsilon$$

and where τ and ε are specified in DATA(3) and DATA(4).
Recall XSOLVE

0 All parameters satisfy the convergence criterion.

-1 A new base point has been found but $\lambda > 1$ and $\gamma > 90^o$ and the convergence criteria have not been met. This implies that numerical difficulties are present.

-2 There are more unknowns than equations (N<K)

-3 The total number of variables to be varied is zero as indicated in the XV vector.

-4 The convergence criteria have been met (same as NDATA(3) = 0) but $\lambda > 1$ and $\gamma < 45^o$. This generally means that progress has been very slow due perhaps to the presence of a ridge.

-5 On entry the value of NDATA(1) was zero or negative.

-6 One of the variables was out of the stated range of XMAX and XMIN on entry.

-7 The value of $\lambda > 10^8$ but the convergence criteria have not been met. This implies ε may be too small.

-8 Convergence criterion has been met in equation solving but $\phi \geq 10^{-10}$. This implies existence of a relative minimum which is not an exact solution.

NDATA(4) – index of variable for which analytical partial derivatives are to be calculated. NATA(12) gives the variable index when numerical derivatives are being used (output)

NDATA(5) to NDATA(26) – used internally.

DATA – A storage vector

DATA(1) – initial value of ν. If DATA(1)$\leq$ 0, ν is set to 10 internally. ν is the factor used to change λ by multiplication or division. For a finer one dimensional search set to a smaller value, say 2. (Input)

DATA(2) - initial value of λ. If DATA(2) $\leq$ 0, λ is set to 0.01 internally. This value will automatically change as the computation continues. It may be monitored in OUTPUT(6). λ is the factor that is used to combine the gradient and the Newton-Raphson methods. When λ is large (i.e. ,1.) the search is primarily In the negative gradient direction. When it is small (i.e, 10^{-5}) it is primarily in the Newton-Raphson direction. (input)

DATA(3) - the initial value of τ. If DATA (3) $\leq$ 0, τ is set to 0.001 internally. τ is used in the convergence test. See NDATA(3) above (input).

DATA(4) - the initial value of ε. If DATA(4) $\leq$ 0, ε is set to 0.0002 internally. ε is used In the convergence test (input).

DATA (5) - initial value $\phi_{\min}$. If DATA(5) $\leq$ 0, $\phi_{\min}$ is set to 0.0 internally. When

$\phi \leq \phi_{min}$, the partial derivatives from the previous iteration are used instead of computing them again. (input)

DATA(6) - DATA(9) - after the first call, the values of ν, τ, ε and ϕ_{min}, respectively, used internally (output).

DATA(10) thru DATA(16) - used internally.

X - the vector of the K unknowns. On first entry into the subprogram, initial estimates must be supplied for X(1) to X(K). On each exit, the routine supplies a new improved estimate of the unknowns. On final exit this vector contains the best point found to date. Locations X(K+1) to X(2K) always contain the best point (i. e., base point) found to date.

XV - A vector indicating which of the X variables are actually to be varied by the program. It may be varied after each new base point. (input)
If XV(I) = 0., hold X(I) constant
If XV(I) = 1., allow X(I) to vary by using numerical derivatives.
If XV(I) = -1., allow X(I) to vary by using analytical derivatives calculated by the user in the calling program.

XMAX - a vector containing the upper bounds on all X variables. It may be varied after each new base point. (input)

XMIN – a vector containing the lower bounds on all X variables. It may be varied after each new base point. Thus XMIN(I) < X(I) < XMAX(I). (input)

Y – a vector of the N desired function values. (input)

Z – a vector of N computed function values calculated in the calling program before first entry and on subsequent requests for functional values. Locations Z(N+1) to Z(2N) contain the functional values corresponding to X(K+1) to X(2K). (input)

PJ- a vector of partial derivatives for the variable J at the last base point found. This is one column of the P matrix. The vector that must be calculated in the calling program is:

$$PJ(1) = \frac{\partial z_1}{\partial x_j}$$

$$PJ(2) = \frac{\partial z_2}{\partial x_j}$$

.
.
.

PJ(N) = $\frac{\partial z_n}{\partial x_j}$

It is used for derivatives input only if analytical derivatives are being used. When equations are being solved it is the calculated scaled factors on output

Output - An output vector of variables which is reported at each new base point (output).

Output(1) - ϕ , the value of the sum of squares at the current base point. The value of the sum of squares at the point corresponding to the value of X on entry, can be found in DATA(15) on output.

Output (2) - Υ , the angle in degrees between the step actually taken and the steepest descent direction at the last base point.

Output(3) - a counter for the number of times a return from XSOVE is made. It is set to zero on first entry and incremented by one on each exit.

Output(4) - a counter for the number of functional evaluations required by XSOLVE. It is set to 1 on initial entry (to count the initial functional evaluation) and incremented by 1 each time a return from a functional evaluation is made.

Output(5) - a counter for the number of derivative evaluations required. It is set to zero on first entry and incremented by one each time a return from a partial derivative request is made.

Output(6) - the value of the parameter λ used to find the current base point.

P - A scratch vector used to hold the values of all the partial derivatives computed in the calling program. The first N * K locations contain the partial derivatives stored columnwise (an N x K matrix):

$$\begin{matrix} \frac{\partial z_1}{\partial x_1} & \cdots\cdots\cdots & \frac{\partial z_1}{\partial x_K} \\ . & & . \\ . & & . \\ . & & . \\ \frac{\partial z_N}{\partial x_1} & \cdots\cdots\cdots & \frac{\partial z_N}{\partial x_K} \end{matrix}$$

The partial derivatives in the P vector will be calculated by finite differences or by the calling program, depending on the XV vector.

A and AC - Scratch matrices used internally