Algorithm AS 47: Function Minimization Using a Simplex Procedure

Author(s): R. O'Neill

Source: *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 1971, Vol. 20, No. 3 (1971), pp. 338-345

Published by: Wiley for the Royal Statistical Society

Stable URL: https://www.jstor.org/stable/2346772

```
      W2=0.0
   25 D(ICOL)=W2
      I4=I4+N
      W1=1.0
      GO TO 60
   30 W1=0.0
      W3=D(ICOLW)
      IF(W3.EQ.0.0) GO TO 40
      W1=W2/W3
   40 I2=I1
      DO 50 IROW=1,N
      I2=I2+1
      I4=I4+1
   50 V(I2)=V(I2)-W1*V(I4)
   60 IC=IC+1
      C(IC)=W1
   70 CONTINUE
      RETURN
      END
```

Algorithm AS 47

Function Minimization using a Simplex Procedure

By R. O'Neill

University of Bath

Language

ANSI Standard Fortran

Description and Purpose

The minimum is found of a user-specified function of N variables. The algorithm used is due to Nelder and Mead (1965). On exit, the minimum value of the function is in *YNEWLO*, its N co-ordinates in *MIN*, an $(N \times 1)$ array and the number of function evaluations performed in *ICOUNT*.

On entry we construct a simplex, i.e. $(N+1)$ points in N dimensions.

The size, shape and orientation of the simplex are determined by *START*(I) containing the N co-ordinates of the guessed starting point and the vector *STEP*(I). The values used in *STEP*(I) will depend on the required size of the simplex and the relative magnitudes of the units for each variable. Optimum values of *STEP* and *START* exist but since they imply knowledge of the position of the minimum they obviously cannot be determined. Fortunately, the algorithm will work successfully for all values of *STEP* and *START* but those far from optimum will need more iterations to converge.

The algorithm will work for functions of any number of variables (here limited to 20); our only assumptions about the function are that it is continuous and one-valued.

We illustrate the algorithm with an example of the minimization of a function of two variables.

Let $F(A)$ be the function value of the point $A = (X_A, Y_A)$ being a point in the space of the two variables.

Suppose $F(A) >$ both $F(B)$ and $F(C)$ then we replace A by its reflection, E, in D, the centroid of B and C. If E is not successful, i.e. $F(E) \geqslant F(B)$ and $F(C)$, then we contract to G or H depending whether $F(A)$ or $F(E)$ is lower. In the unusual case that $F(G) > F(A)$ (or equivalently $F(H) > F(E)$) then the simplex is contracted around B or C according as $F(B)$ or $F(C)$ is lower.

If, however, E is successful and $F(E) < F(B)$ and $F(C)$ we extend to point J. We then keep E or J as a replacement for A according as $F(E)$ or $F(J)$ is lower.

If neither contraction nor extension is necessary we replace A by E.

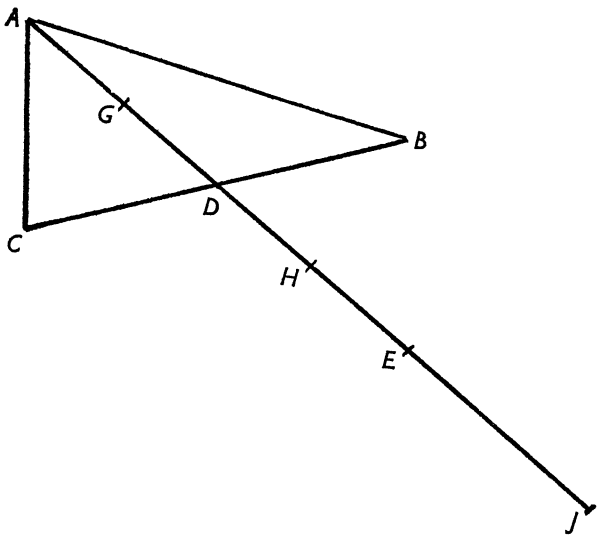

FIG. 1.

Iteration then continues with B, C and the replacement for A, the simplex distorting in shape according to the slopes it encounters.

In the program the "reflection" (ED/DA), "extension" (JD/DA) and "contraction" ($HD/DA = GD/DA$) coefficients are set at 1·0, 2·0 and 0·5 respectively. These are the values recommended by Nelder and Mead as being best for a general situation.

The method for terminating the algorithm is to calculate the variance of the $(N+1)$ function values occurring at the simplex vertices. If this is less than the user-supplied *REQMIN* then the algorithm is terminated. On input *ICOUNT* contains an upper limit to the number of function evaluations. If this is exceeded the algorithm will be terminated.

To save time we need not make the convergence check at each iteration. The frequency can be controlled using the parameter *KONVGE*.

When the algorithm terminates a check is made to verify whether the point is a local minimum. The value *DEL* is set at $0{\cdot}001 \times STEP(I)$, the original step-length.

The function value is calculated at the suspected minimum $\pm$ *DEL* along the axis corresponding to each of the N variables.

Only if all $2N$ of these values are greater than the proposed minimum is the latter accepted.

If a false minimum is found the simplex is contracted around the lowest point and restarted. For each restart the value of *ICOUNT* is augmented by 10,000. Note that the value of *ICOUNT* does not include function evaluations used to set up the simplex or to check the minimum.

The method is in essence opportunist, only current information being used. However, the simplicity of the algorithm and lack of assumptions about the function or its derivatives make it of very general application.

Structure

SUBROUTINE NELMIN (N, START, XMIN, YNEWLO, REQMIN, STEP, KONVGE, ICOUNT)

Formal parameters

N	Integer	input:	the number of variables over which we are minimizing.
START(N)	D–P real array	input:	contains the co-ordinates of the starting point.
XMIN(N)	D–P real array	output:	contains the co-ordinates of the minimum.
YNEWLO	D–P real array	output:	the minimum value of the function.
REQMIN	D–P real	input:	the terminating limit for the variance of function values.
STEP(N)	D–P real array	input:	determines the size and shape of initial simplex. The relative magnitudes of its *N* elements should reflect the units of the *N* variables.
KONGVE	Integer	input:	the convergence check is carried out every *KONVGE* iterations.
ICOUNT	Integer	input:	maximum number of function evaluations.
		output:	function evaluations performed + 10,000 times number of restarts. Negative *ICOUNT* value identifies input parameter fault(s).

Auxiliary algorithm

DOUBLE PRECISION FUNCTION FN(A)

Formal parameter

A(N)	D–P real array	input:	contains the co-ordinates of the point at which we wish to evaluate the function.

Being a function subprogram *FN* must contain at least one statement of the form

$$FN = \ldots\ldots .$$

The subprogram calculates the function value at the point defined by the elements of *A*.

Example

If we are minimizing the sum of the cubes of two variables our subprogram would read:

DOUBLE PRECISION FUNCTION FN(A)

```
DOUBLE PRECISION A(2)
FN = A(1) ** 3 + A(2) ** 3
RETURN
END
```

Restrictions and Error Messages

If any of the following restriction are contravened then control will return to the main program with a negative value of *ICOUNT*.

REQMIN must be greater than zero (*ICOUNT* = −1),
N must be less than 21 (*ICOUNT* = −10),
KONGVE must be greater than 0 (*ICOUNT* = −100).

If more than one restriction is violated then *ICOUNT* will contain the sum of the error values.

The data values *RCOEFF* (reflection), *ECOEFF* (extension) and *CCOEFF* (contraction) can be changed by rewriting the *DATA* card.

The actual parameters corresponding to the formal parameters *START*, *XMIN*, *STEP*, *YNEWLO* and *REQMIN* must be declared in the calling program as double precision. The arrays must have at least *N* locations.

The array of co-ordinates in the function subprogram must be declared as double precision with at least *N* locations.

Test Results

The program was tested on an ICL 4–50 computer using four test functions commonly quoted in literature. Each has a function minimum of zero.

(1) Rosenbrock's parabolic valley (Rosenbrock, 1960):
$y = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$, starting point $(-1{\cdot}2, 1)$.

(2) Powell's quartic function (Powell, 1962):
$y = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$, starting point $(3, -1, 0, 1)$.

(3) Fletcher and Powell's helical valley (Fletcher and Powell, 1963):
$y = 100\{x_3 - 10\theta(x_1, x_2)\}^2 + \{\sqrt{(x_1^2 + x_2^2)} - 1\}^2 + x_3^2$
where $2\pi\theta(x_1, x_2) = \arctan(x_2/x_1), x_1 > 0$
$= \pi + \arctan(x_2/x_1), x_1 < 0,$
starting point $(-1, 0, 0)$.
N.B. Since arctan (0/0) is not defined neither is the function on the line $(0, 0, x_3)$. This line was excluded by assigning a very high value (10,000) to the function there.

(4) $y = \sum_{i=1}^{10} x_i^4$, starting point $(1, 1, \ldots, 1)$.

Results

The parameters were set to be:

$REQMIN = 10^{-16}$: far more accurate than usually necessary.
$STEP(I) = 1{\cdot}0$: side-length of the initial simplex.
$KONVGE = 5$: i.e. convergence tested on every fifth function evaluation.
$ICOUNT = 1{,}000$: limit on number of function evaluation.

These gave:

Function	*Function evaluations performed*	*No. of restarts*	*Value at minimum*
1	148	0	$3{\cdot}19 \times 10^{-9}$
2	209	0	$7{\cdot}35 \times 10^{-8}$
3	250	0	$5{\cdot}29 \times 10^{-8}$
4	474	1	$3{\cdot}80 \times 10^{-7}$

TIME

Execution time is dependent on the number of function evaluations and particularly on the complexity of the function. As an indication of central processor unit time the first two functions (total 357 iterations) took 3·34 seconds while the more involved functions 3 and 4 (total 724 iterations) took 22·25 seconds (*ICL* 4–50).

ACKNOWLEDGEMENTS

I would like to thank Dr J. A. Nelder for his advice and the referees for their useful comments.

This algorithm forms part of a project financed by a grant from the Science Research Council.

REFERENCES

FLETCHER, R. and POWELL, M. J. D. (1963). A rapidly convergent descent method for minimization. *Computer J.*, **6**, 163–168.

NELDER, J. A. and MEAD, R. (1965). A simplex method for function minimization. *Computer J.*, **7**, 308–313.

POWELL, M. J. D. (1962). An iterative method for finding stationary values of a function of several variables. *Computer J.*, **5**, 147–151.

ROSENBROCK, H. (1960). An automatic method for finding the greatest or least value of a function. *Computer J.*, **3**, 175–184.

```
      SUBROUTINE NELMIN(N,START,MIN,YNEWLO,REQMIN,STEP,KONVGE,ICOUNT)
C
C        ALGORITHM AS 47  APPLIED STATISTICS (J.R.STATIST.SOC C),
C        (1971) VOL.20, NO.3
C
C     THE NELDER-MEAD SIMPLEX MINIMISATION PROCEDURE
C
C
C        PURPOSE :: TO FIND THE MINIMUM VALUE OF A USER-SPECIFIED
C        FUNCTION.
C
C
C        FORMAL PARAMETERS ::
C
C             N :  INPUT : THE NUMBER OF VARIABLES OVER WHICH WE ARE
C                        : MINIMISING
C         START :  INPUT : ARRAY; CONTAINS THE COORDINATES OF THE
C                        : STARTING POINT.
C           MIN : OUTPUT : ARRAY; CONTAINS THE COORDINATES OF THE
C                        : MINIMUM.
C        YNEWLO : OUTPUT : THE MINIMUM VALUE OF THE FUNCTION.
C        REQMIN :  INPUT : THE TERMINATING LIMIT FOR THE VARIANCE OF
C                        : FUNCTION VALUES.
C          STEP :  INPUT : ARRAY; DETERMINES THE SIZE AND SHAPE OF THE
```

```
C                          : INITIAL SIMPLEX. THE RELATIVE MAGNITUDES OF
C                          : ITS N ELEMENTS SHOULD REFLECT THE UNITS OF
C                          : THE N VARIABLES.
C         KONVGE :  INPUT  : THE CONVERGENCE CHECK IS CARRIED OUT EVERY
C                          : KONVGE ITERATIONS.
C         ICOUNT :  INPUT  : MAXIMUM NUMBER OF FUNCTION EVALUATIONS.
C                  OUTPUT  : FUNCTION EVALUATIONS PERFORMED + 10,000
C                          : TIMES NUMBER OF RESTARTS. NEGATIVE ICOUNT
C                          : VALUE IDENTIFIES INPUT PARAMETER FAULT(S).
C
C         ALL VARIABLES AND ARRAYS ARE TO BE DECLARED IN THE CALLING
C         PROGRAM AS DOUBLE PRECISION.
C
C
C         AUXILIARY ALGORITHM :: THE DOUBLE PRECISION FUNCTION
C         SUBPROGRAM FN(A) CALCULATES THE FUNCTION VALUE AT POINT A.
C         A IS DOUBLE PRECISION WITH N ELEMENTS.
C
C
C         REFERENCE :: NELDER,J.A. AND MEAD,R.(1965). A SIMPLEX METHOD
C         FOR FUNCTION MINIMIZATION. COMPUTER J.,VOL.7,308-313
C
C*************************************************************************
C
      DOUBLE PRECISION START(N),MIN(N),YNEWLO,REQMIN,STEP(N),
     1   P(20,21),PSTAR(20),P2STAR(20),PBAR(20),Y(20),
     2   DN,DNN,Z,SUM,SUMM,YLO,RCOEFF,YSTAR,ECOEFF,Y2STAR,CCOEFF,
     3   CURMIN,DEL,FN
C
      DATA RCOEFF/1.D0/,ECOEFF/2.D0/,CCOEFF/5.D-1/
C        REFLECTION,EXTENSION AND CONTRACTION COEFFICIENTS.
C
C        VALIDITY CHECKS ON INPUT PARAMETERS.
C
      KCOUNT=ICOUNT
      ICOUNT=0
      IF(REQMIN.LE.0.D0) ICOUNT=ICOUNT-1
      IF (N .GT. 20) ICOUNT=ICOUNT-10
      IF (KONVGE .LE. 0) ICOUNT=ICOUNT-100
      IF (ICOUNT .LT. 0) RETURN
C
      JCOUNT=KONVGE
      DN=DFLOAT(N)
      NN=N+1
      DNN=DFLOAT(NN)
      DEL=1.D0
C
C        CONSTRUCTION OF INITIAL SIMPLEX
C
 1001 DO 1 I=1,N
    1 P(I,NN)=START(I)
      Z=FN(START)
      Y(NN)=Z
      SUM=Z
      SUMM=Z*Z
      DO 2 J=1,N
      START(J)=START(J)+STEP(J)*DEL
      DO 3 I=1,N
    3 P(I,J)=START(I)
      Z=FN(START)
      Y(J)=Z
      SUM=SUM+Z
      SUMM=SUMM+Z*Z
    2 START(J)=START(J)-STEP(J)*DEL
```

```
C
C        SIMPLEX CONSTRUCTION COMPLETE
C
C        FIND HIGHEST AND LOWEST Y VALUES. YNEWLO ( =Y(IHI) ) INDICATES
C        THE VERTEX OF THE SIMPLEX TO BE REPLACED.
C
 1000 YLO=Y(1)
      YNEWLO=YLO
      ILO=1
      IHI=1
      DO 5 I=2,NN
      IF(Y(I) .GE. YLO) GO TO 4
      YLO=Y(I)
      ILO=I
    4 IF (Y(I) .LE. YNEWLO) GO TO 5
      YNEWLO=Y(I)
      IHI=I
    5 CONTINUE
      SUM=SUM-YNEWLO
      SUMM=SUMM-YNEWLO*YNEWLO
C
C        CALCULATE PBAR, THE CENTROID OF THE SIMPLEX VERTICES
C        EXCEPTING THAT WITH Y VALUE YNEWLO.
C
      DO 7 I=1,N
      Z=0.D0
      DO 6 J=1,NN
    6 Z=Z+P(I,J)
      Z=Z-P(I,IHI)
    7 PBAR(I)=Z/DN
C
C        REFLECTION THROUGH THE CENTROID.
C
      DO 8 I=1,N
    8 PSTAR(I)=(1.D0+RCOEFF)*PBAR(I)-RCOEFF*P(I,IHI)
      YSTAR=FN(PSTAR)
      ICOUNT=ICOUNT+1
      IF (YSTAR .GE. YLO) GO TO 12
C
C        SUCCESSFUL REFLECTION, SO EXTENSION
C
      DO 9 I=1,N
    9 P2STAR(I)=ECOEFF*PSTAR(I)+(1.D0-ECOEFF)*PBAR(I)
      Y2STAR=FN(P2STAR)
      ICOUNT=ICOUNT+1
C
C        RETAIN EXTENSION OR CONTRACTION.
C
      IF (Y2STAR .GE. YLO) GO TO 19
   10 DO 11 I=1,N
   11 P(I,IHI)=P2STAR(I)
      Y(IHI)=Y2STAR
      GO TO 900
C        NO EXTENSION.
   12 L=0
      DO 13 I=1,NN
      IF(Y(I).GT.YSTAR) L=L+1
   13 CONTINUE
      IF (L .GT. 1) GO TO 19
      IF (L .EQ. 0) GO TO 15
C
C        CONTRACTION ON THE REFLECTION SIDE OF THE CENTROID.
C
      DO 14 I=1,N
```

```
   14 P(I,IHI)=PSTAR(I)
      Y(IHI)=YSTAR
C
C        CONTRACTION ON THE Y(IHI) SIDE OF THE CENTROID.
C
   15 DO 16 I=1,N
   16 P2STAR(I)=CCOEFF*P(I,IHI)+(1.D0-CCOEFF)*PBAR(I)
      Y2STAR=FN(P2STAR)
      ICOUNT=ICOUNT+1
      IF (Y2STAR .LE. Y(IHI)) GO TO 10
C
C        CONTRACT WHOLE SIMPLEX
C
      SUM=0.D0
      SUMM=0.D0
      DO 18 J=1,NN
      DO 17 I=1,N
      P(I,J)=(P(I,J)+P(I,ILO))*0.5D0
   17 MIN(I)=P(I,J)
      Y(J)=FN(MIN)
      SUM=SUM+Y(J)
   18 SUMM=SUMM+Y(J)*Y(J)
      ICOUNT=ICOUNT+NN
      GO TO 901
C        RETAIN REFLECTION
   19 DO 20 I=1,N
   20 P(I,IHI)=PSTAR(I)
      Y(IHI)=YSTAR
  900 SUM=SUM+Y(IHI)
      SUMM=SUMM+Y(IHI)*Y(IHI)
  901 JCOUNT=JCOUNT-1
      IF (JCOUNT .NE. 0) GO TO 1000
C
C        CHECK TO SEE IF MINIMUM REACHED.
C
      IF ( ICOUNT .GT. KCOUNT ) GO TO 22
      JCOUNT=KONVGE
      CURMIN=(SUMM-(SUM*SUM)/DNN)/DN
C
C        CURMIN IS THE VARIANCE OF THE N+1 FN VALUES AT THE VERTICES.
C
      IF (CURMIN .GE. REQMIN) GO TO 1000
C
C        FACTORIAL TESTS TO CHECK THAT YNEWLO IS A LOCAL MINIMUM
C
   22 DO 23 I=1,N
   23 MIN(I)=P(I,IHI)
      YNEWLO=Y(IHI)
      IF ( ICOUNT .GT. KCOUNT ) RETURN
      DO 24 I=1,N
      DEL=STEP(I)*1.D-3
      MIN(I)=MIN(I)+DEL
      Z=FN(MIN)
      IF (Z .LT. YNEWLO) GO TO 25
      MIN(I)=MIN(I)-DEL-DEL
      Z=FN(MIN)
      IF (Z .LT. YNEWLO) GO TO 25
   24 MIN(I)=MIN(I)+DEL
      RETURN
C
C        RESTART PROCEDURE
C
   25 DO 26 I=1,N
   26 START(I)=MIN(I)
      DEL=1.D-3
      ICOUNT=ICOUNT+10000
      GO TO 1001
      END
```