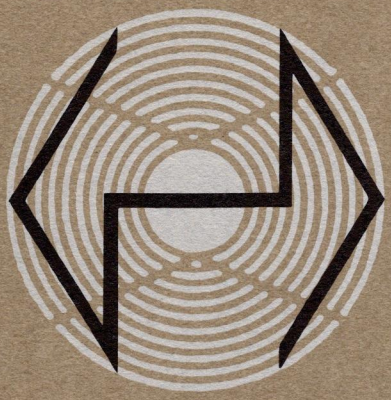
Raabo.



A/S REGNECENTRALEN

SCANDINAVIAN INFORMATION PROCESSING SYSTEMS

RCSL No: 53-M18
Edition: December 1970
Author: Helge Elbrønd Jensen

Title: minimum

Keywords: RC 4000, Software, minimum, calculation of extrema, Algol procedure, ISO Tape

Abstract: The procedure, minimum, calculates extrema of a differentiable function in n variables. 17 pages.

DK-2500 VALBY · BJERREGAARDSVEJ 5 · TELEPHONE: (01) 46 08 88 · TELEX: 64 64 rcinf dk · CABLES: INFOCENTRALEN

1. Function and parameters

Let F denote a real, twice differentiabel function in n variables, and suppose that the first order derivatives of F are given analytically (that is, as expressions depending upon the n variables). Suppose that in a given area the function is bounded below and has a minimum. From a reasonable good starting point the procedure finds this minimum by finding a point at which all the first order derivatives are zero (that is, smaller than a prescribed quantity).

Procedure head:

```
minimum(n, i, x, F, delta, eps, point);
value n;
integer i, n;
real eps, F, delta;
array x, point;
```

Call parameters:

n: the number of variables for the given function.

Call/Return parameters:

point: a real array point(1:n);
at entry point containes the starting point for the procedure;
at exit point containes the coordinates of the point at which the minimum is obtained;

eps: a real quantity affecting the precision to which the minimum is calculated. Consider the norm of the vector consisting of the first order derivatives. If this norm is smaller than eps, then the procedure will stop;
at exit eps containes the norm of the vector described above.

Return parameters:

minimum: the value of the obtained minimum;

Other parameters:

F: a real procedure denoting the given function. In a program in which the procedure minimum is called, F must be declared in the following way:

```
real procedure F(x);
array x;
F:= the given expression;
```

delta: a real procedure delta(i, x) denoting for each i the partiel derivative of F with respect to the variable x(i);
In a program in which the procedure minimum is called, delta must be declared in the following way:

```
real procedure delta(i, x);
integer i;
array x;
delta:= case i of (..., ...,      , ...);
```

In the parenthesis there must be n expressions, where the i-th expression denotes the partiel derivative of F with respect to the variable x(i);

2. The method

Let F denote a function in n variables, and let x denote the n-dimensional point with coordinates (x(1), x(2), ... , x(n)). F is said to have a minimum at a point x0, if there exist a small area including x0, in which the value of F at each point is greater than F(x0).
Most of the various methods for finding a minimum for a function i n variables has one idea in commen: They are all iterative processes based upon a roul, which for each point specifies a certain direction in which the next point of the process is to be found, and for each such direction specifies how to find the next point. Now, suppose that

the function is differentiabel. By the gradient of F at the point x - denoted gradient (x) - we understand the n-dimensional vector, which as the i-th coordinate has the partiel derivative of F with respect to x(i) at the point x.

The method used in the following program is essentially based upon to papers of A.A. Goldstein ((2), (3)). We suppose, that the function is twice differentiabel and that the gradient is given analytically. It is well known, that the gradient will vanish at a minimumpoint.

Let the points of the iterative process be denoted x1, x2, x3, ... xk, ..., where x1 is given by the input array point.

For each k fi(xk) denotes the n-dimensional vector which terminates the new direction.

We choose fi(x1) = gradient (x1).

For each k the number h(k) is defined as:

h(k) = r × norm(n, fi(xk)).

r is calculated at the beginning of the program in such a way that h(1) < 1/5.

norm is denoting the ordinary n-dimensional Euklidian norm.

Then the algorithm, at each point xk, consists of the following two steps:

1. DIRECTION:

We compute an n × n matrix, which is an approximation to the matrix consisting of the second order derivatives of F.

For each j let F(j) denote the vector, which has the j-th coordinate equal to 1 and the others equal to zero.

We then compute the matrix Q(xk) which has the j-th column equal to

(gradient(xk + h(k) × F(j)) - gradient(xk)/h(k).

If the matrix Q(xk) is singular (it is has no inverse) then we define the new direction fi(xk) by

fi(xk) = gradient(xk).

Suppose now, that Q(xk) has an inverse, which we denote P(xk).

If (gradient(xk), P(xk) × gradient(xk)) > 0

(where (,) denotes the ordinary innerproduct) then we define fi(xk) by

fi(xk) = P(xk) × gradient(xk).

If (gradient(xk), P(xk) × gradient (xk)) $\leq$ 0
then we define fi(xk) by
fi(xk) = gradient(xk).

2. KONSTANT:

The next point in the process is now obtained on the form
xk - gk × fi(xk)
where gk is a constant calculated as follows:
Let product = (gradient(xk), fi(xk)).
Let f1 = F(xk).
Let f2 = F(xk - gk × fi(xk)).
Then gk is calculated such that
f2 < f1 and (f1 - f2) < gk × product.

It can be proved, by using the Taylor formula, that such a gk always exists, and that xk calculated in this way will converge to a minimum-point for F.((2), (3)). From a numerical point of view however, gk might fail to exist, and in this case the procedure will stop.

3. Accuracy, Time and Storage Requirements

Accuracy: As measure of accuracy we use the norm of the gradient. If the procedure succeeds, then at the end this norm is smaller than the call parameter eps.

Time: This depends on the wanted accuracy and first of all on the problem in question, so it is not possible to give general rules for this. (See 4. Test and Discussion).

Storage requirements: 10 segments of program
Typographical length: 248 lines.

4. Test and Discussion

The procedure have been tested on several functiones among which we describe the two most difficult problems:

1. Minimising the function in two variables

   F = 100 × (x(2) - x(1)××2) ×× 2 + (1 - x(1)) ×× 2

2. Finding a solution to the following three non-linear equations:

   sin(x(1)××2) + exp(x(2)) × x(3) - 4 = 0

   x(1) + x(2) + x(3) - 3 = 0

   x(1) + x(2)××2 + x(3)××3 - 14 = 0

   This is done by minimising the square-sum of the three equations.

First we consider the problem 1:

The function F has minimum at the point (1, 1) with functionvalue 0.

Starting at the point x(1) = -1.2 and x(2) = 1 and using different values of the term eps, the following results were obtained:

	Value of eps			
	$_{10}$-4	$_{10}$-6	$_{10}$-8	$_{10}$-10
Minimum	0.999999592 0.999999183	0.999999592 0.999999183	1.000000000 1.000000000	1.000000000 1.000000000
Fc.-value	0.000000000	0.000000000	0.000000000	0.000000000
Gr.-norm	7.3_{10}-5	4.2_{10}-7	5.1_{10}-8	1.7_{10}-10
Ex.-time	0.76	0.73	0.75	0.75

(the execution time is in seconds).

It follows, that the procedure succeeds in all 4 situations, and that smaller values of eps does not affect the execution time. This last observation however can not be stated in general, (see below under problem 2).

Using eps = $_{10}$-8 and using different starting points the following results were obtained:

Starting-point	-1.200000000 1.000000000	0.000000000 1.000000000	-0.500000000 -0.500000000	2.000000000 0.250000000
Execution-time	0.75	0.49	0.71	0.80

In all 4 situations the minimum was obtained at the point:

1.000000000
1.000000000

with the functionvalue 0.000000000 and gradient norm 5.1_{10}-8.
Again the procedure succeeds in all 4 situations.

Next, consider the problem 2 in three variables. Starting at the point x(1) = 0, x(2) = 0, x(3) = 2.5 and using different values of the term eps, the following results were obtained:

	Value of eps			
	$_{10}$-4	$_{10}$-6	$_{10}$-8	$_{10}$-10
Minimum	0.097831561 0.512917627 2.389250732	0.097830233 0.512919004 2.389250762	0.097830224 0.512919014 2.389250762	0.097830224 0.512919014 2.389250762
Fc.-value	0.000000000	0.000000000	0.000000000	0.000000000
Gr.-norm	5.8_{10}-5	3.9_{10}-7	3.2_{10}-8	3.2_{10}-8
Ex.-time	1.96	2.55	3.81	3.97

It follows, that the procedure succeeds in the first three situations, but that it is not possible to make the gradient norm smaller than 3.2_{10}-8, so in this sense the procedure does not succeed in the last situation. In this case smaller values of eps gives greater execution time, even if the obtained minimumpoints are practically the same in the last three cases.

Using eps = $_{10}$-8 and using different starting points the following results were obtained:

Starting-point	0.000000000 0.000000000 2.500000000	0.000000000 0.000000000 1.000000000	0.500000000 1.000000000 2.000000000	1.000000000 1.000000000 1.000000000
Execution-time	3.81	1.97	3.09	2.45

In all 4 situations the minimum was obtained at the point:

0.097830223
0.512919014
2.389250762

with the functionvalue 0.000000000 and gradient norm 3.2_{10}-8
It follows, that the procedure succeeds in all 4 situations.

Example

Consider the function

F = 100 × (x(2) - x(1)××2) ×× 2 + (1 - x(1)) ×× 2

Starting at the point x(1) = -1.2 and x(2) = 1 the following program might be used to find the minimum of F:

Testprogram

```
begin
   integer i, j;
   real a, eps;
   array x, point(1:2);

   real procedure F(x);
   array x;
   F:= 100 × (x(2) - x(1)××2) ×× 2 + (1 - x(1)) ×× 2;

   real procedure delta(i, x);
   integer i;
   array x;
   delta:= case i of (-400××x(1) × (x(2) - x(1)××2) - 2 × (1 - x(1)),
                      200 × (x(2) - x(1)××2));

   point(1):= -1.2; point(2):= 1; eps := 10-8;
   a:= minimum(2, i, x, F(x), delta(i, x), eps, point);
   write(out, <:Minimum obtained at the point <10>:>);
   for j:= 1 step 1 until 2 do
   write(out, <:<10>:>, <<-dddd.ddddddddd>, point(j));
   write(out, <:<10><10> Minimumvalue =:>,<<-dddd.ddddddddd>, a);
   write(out, <:<10><10> Gradient norm=:>, <<-d.d10-dd>, eps);
end;
```

This will give the following output:

```
Minimum obtained at the point
   1.000000000
   1.000000000

Minimumvalue = 0.000000000
Gradient norm = 1.0' -8
end
```

In the program we use a boolean procedure inverse to find the inverse (if it exist) of an n × n matrix.
The procedure is based upon Simpel Gaussian illimination and is only introduced in order to make the program complete. One could use any other procedure of this sort, for ex. decompose-solve from RC mathematical procedure library.

Since a minimum of the function F is a maximum of the function -F, the procedure will of course be able to find maximum as well as minimum.

5. References

(1) D. Fletcher and M.J.D. Powell:
A rapidly convergent descent method for minimisation.
Comput. Journal 6 p. 163-168 (1963)

(2) A.A. Goldstein: On steepest descent.
Journal Siam Control Vol. 3 No 1 p 147-151 (1965)

(3) A.A. Goldstein and J.F. Puce:
An effective algorithm for minimisation
Numerische Mathematik 10 p. 184-189 (1967)

(4) E. Isaacson and H.B. Keller:
Analyses of numerical Methods
John Wiley and Sons, Inc. (1966)

6. Algol text

```
minimum = set 10
minimum = algol
external

 real procedure minimum(n,i,x,F,delta,eps,point);
 value n;
 integer i,n;
 real eps,F,delta;
 array x,point;

begin
 integer j;
 real h,g,g1,gamma,r,f1,f2,f3,product,k,s;
 array psi,y,z,b(1:n),p,q(1:n,1:n);



 real procedure norm(n,a);
 value n;
 integer n;
 array a;
 begin
  comment this is the ordinary norm in the n-dimensional Euklidian
  space;
  real h;
  h:=0;for i:=1 step 1 until n do h:=h+a(i)××2;
  norm:=sqrt(h);
 end;



 real procedure innerproduct(n,a,b);
 value n;
 integer n;
 array a,b;
 begin
  comment this is the ordinary innerproduct in the n-dimensional
```

```
 Euklidian space;
 real h;
 h:=0; for i:=1 step 1 until n do h:=h+a(i)×b(i);
 innerproduct:=h;
end;



procedure equal(n,a,b);
value n;
integer n;
array a,b;
begin
 comment the procedure identifies two arrays;
 for i:=1 step 1 until n do b(i):=a(i);
end;



boolean procedure inverse(n,a,b);
value n;
integer n;
array a,b;
 comment the procedure finds the inverse ( if it exists ) of the
 matrix a by Gaussian illimination.If the inverse exist,it is
 stored in b.If the inverse does not exist,inverse is false;
begin
 integer i,j,k,m,pivotnr;
 real pivot,s;
 array c(1:n,1:n),x(1:n),d(1:n);

 inverse:=true;
 for m:=1 step 1 until n do
 begin
  comment for each m one is solving the linear system,which on the
  wright side has the m-th column in the unit-matrix,and on the left
  side the given matix as coefficientmatrix and the m-th column in
  the wanted inverse as unknown;
```

```
for j:=1 step 1 until n do
for i:=1 step 1 until n do c(i,j):=a(i,j);
for i:=1 step 1 until n do d(i):=(if i=m then 1 else 0);
for k:=1 step 1 until n-1 do
begin
 comment among the last n-k+1 equations one is finding the equation,
 which has the numerical largest coefficient in x(k);
 pivot:=0; pivotnr:=0;
 for i:=k step 1 until n do if abs(c(i,k))>pivot then
 begin  pivot:=c(i,k); pivotnr:=i;  end;
 if pivot=0 then begin inverse:=false; goto END;end;
 comment if pivot=0 then the given matrix has determinant 0 and
 consequently no inverse;
 if pivotnr<>k then
 begin
  comment equation number k is replaced by equation number pivotnr
  and vica versa;
  s:=d(k); d(k):=d(pivotnr); d(pivotnr):=s;
  for j:=k step 1 until n do
  begin
   x(j):=c(k,j); c(k,j):=c(pivotnr,j); c(pivotnr,j):=x(j);
  end;
 end if pivotnr<>k;
 for i:=k+1 step 1 until n do
 begin
  comment x(k) is calculated from the k-th equation,and the
  expression inserted in the following n-k equations;
  d(i):=d(i)-d(k)×c(i,k)/c(k,k);
  for j:=k+1 step 1 until n do
  c(i,j):= c(i,j)-c(i,k)×c(k,j)/c(k,k);
 end;
end k;
if c(n,n)=0 then begin inverse:=false; goto END;end else
x(n):=d(n)/c(n,n);
for i:=n-1 step -1 until 1 do
begin
 comment for each i x(i) is calculated from the equation
 c(i,i)×x(i) + c(i,i+1)×x(i+1) + . . . +c(i,n)×x(j) = d(i),
```

```
  where x(i+1),. . . x(n) are known;
  s:=0; for j:=n step -1 until i+1 do s:=s+c(i,j)×x(j);
  x(i):=(d(i)-s)/c(i,i);
 end;
 for i:=1 step 1 until n do b(i,m):=x(i);
end m;
END: end;



procedure search(n,g,y,psi,f2);
value n,g;
integer n;
real g,f2;
array y,psi;
begin
 comment the procedure finds the value of the function to be
 minimised,that is k×F,at the point obtained from y by going the
 distance g in the direction -psi;
 for i:=1 step 1 until n do x(i):=y(i)-g×psi(i);
 f2:=k×F;
end;



equal(n,point,x);equal(n,x,y); k:=1;
for i:=1 step 1 until n do psi(i):=delta;
comment psi is the gradient of F at the starting point;
equal(n,psi,b); h:=product:=norm(n,psi);
if h<1 then r:=1/5 else r:=1/(5×h);

KONSTANT:
; comment at each step of the iterativ process the procedure will
 goto KONSTANT and run through the following.A point y and a
 direction psi is given,and the problem is to find a konstant g
 such that the point y-g×psi can be used as the next point;
 h:=norm(n,psi); equal(n,y,x);
 if h/product<1/10 then
```

```
begin
 comment psi is too small relativ to the gradient which implies,
 that the greatest possible progress is too small.We therefore
 consider the function k×F,where k is defined below;
 k:=(h/product)××(1/n);
 for i:=1 step 1 until n do psi(i):=(1/k)××n×psi(i);
 for i:=1 step 1 until n do b(i):=k×delta;
 h:=norm(n,psi);
 if h<1 then r:=1/5 else r:=1/(5×h);
end;
h:=r×h; f1:=k×F;
comment h is used below as the small quantity in the approximation
of the second order derivatives of F,r is introduced in order to
insure,that this quantity is not too big at the beginning;
product:=innerproduct(n,b,psi);
g:=1; g1:=0;
search(n,1,y,psi,f2);
if f1-f2>=1/4×product then
begin
 f1:=f2; equal(n,x,y); goto DIRECTION;
end;
comment in this case we use g=1/4 and the next point is
therefore obtained as y-1/4×psi;
s:=( if s<1 then 10-10 else 1/s×10-10 );
for g:=g/2 while f1<=f2 do
begin
 search(n,g,y,psi,f2);
 if g<s  then begin equal(n,y,x); goto END; end;
end;
comment if g is smaller than s (see the definition of this term)
then the next point of the process will be practically equal to
the present,and we must therefore conclude,that the procedure is
unable to make further progress;
g:=2×g; equal(n,x,z);
if (f1-f2)<g×product then goto SECOND else
begin
 comment in this case the functionvalue at y-g×psi is smaller
 than f1,but the condition f1-f2<g×product is not satisfied and
```

```
therefore g is too small;
g1:=g; g:=2×g;
FIRST:
 g:=(g1+g)/2;
 search(n,g,y,psi,f2);
 if f1<f2 then goto FIRST else
 begin
  if (f1-f2)<g×product then
  begin equal(n,x,y); f1:=f2; goto DIRECTION; end else
  begin g:=2×g-g1; g1:=(g+g1)/2; goto FIRST; end;
 end;
end;

SECOND:
; comment in this case the functionvalue at y-g×psi is smaller than
 f1 and the condition f1-f2<g×product is satisfied.We therefore
 look for a smaller g for which this condition is satisfied and
 with a smaller functionvalue than before;
 g:=(g1+g)/2; search(n,g,y,psi,f3);
 if f2<=f3 then
 begin equal(n,z,x);equal(n,x,y); f1:=f2; goto DIRECTION;
 end else
 begin
  if (f1-f3)<g×product then
  begin f2:=f3; equal(n,x,z); goto SECOND;end
  else goto THIRD;
 end;

THIRD:
; comment in this case the functionvalue is smaller than before,
 but the condition mentioned before is not satisfied,so g is
 too small;
 g:=2×g-g1; g1:=(g+g1)/2; g:=(g+g1)/2;
 search(n,g,y,psi,f3);
 if (f1-f3)>=g×product then goto THIRD else
 begin
  if f3>=f1 then goto THIRD else
  begin equal(n,x,y); f1:=f3; goto DIRECTION; end;
 end;
```

```
DIRECTION:
; comment at each step of the iterativ process the procedure will
 goto DIRECTION and run through the following.A point x is given
 and the problem is to determine the direction in which the next
 point is to be found;
 for i:=1 step 1 until n do b(i):=k×delta;
 product:=norm(n,b);
 if product<k×eps or product<10-10 then goto END;
 comment if product<k×eps then the wanted accuracy is obtained.
 if product<10-10 then in most situations it will be meaningless
 to look for further progress;
 for j:=1 step 1 until n do
 begin
  comment an approximation to the matrix consisting of the second
  order derivatives of k×F is calculated and the result stored in q;
  for i:=1 step 1 until n do x(i):=(if i=j then y(i)+h else y(i));
   for i:=1 step 1 until n do p(i,1):=k×delta;
   for i:=1 step 1 until n do q(i,j):=(p(i,1)-b(i))/h;
  end j;
  if -,inverse(n,q,p) then goto STEEPEST else
  begin
   comment if the inverse of q exist,then the vector psi is
   obtained by multiplying the inverse matrix with the gradient;
   for i:=1 step 1 until n do
   begin
    psi(i):=0; for j:=1 step 1 until n do
    psi(i):=psi(i)+p(i,j)×b(j);
   end;
  end;
  if innerproduct(n,psi,b)<=0 then goto STEEPEST else
  goto KONSTANT;
  comment if innerproduct(n,psi,b)<=0 then we can not be sure to
  find a point with smaller functionvalue in the direction psi,
  and therefore psi can not be used.If the innerproduct is >0
  then psi is the new direction;

STEEPEST:
 equal(n,b,psi); goto KONSTANT;
```

```
    comment the gradient is used as the new direction;

  END:
  ; comment the present value of the relevant quantities are stored
   in the return parameters;
   for i:=1 step 1 until n do b(i):=delta;
   minimum:=F; eps:=norm(n,b);
   for i:=1 step 1 until n do point(i):=x(i);
  end;
end;
```