A COMPARISON OF METHODS FOR THE OPTIMIZATION OF A NON-LINEAR FUNCTION SUBJECT TO NON-LINEAR CONSTRAINTS

by

Janine Asaadi

M.Sc.

September 1971

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Mr. G. R. Lindfield, Dr. K. P. Wong, currently lecturer at the University of Singapore, formerly senior research associate at the University of Birmingham, and staff of the Computer Centre, the University of Aston in Birmingham.

SUMMARY

This thesis deals mainly with a comparison of certain computational techniques used for the solution of non-linear constrained mathematical programming problems. The three techniques being considered here are:

(a) Sequential Unconstrained Minimisation Technique, (S.U.M.T.), by Fiacco and McCormick;

(b) Kowalik, Osborne and Ryan's method;

(c) Powell's method for constrained problems.

They all convert the problem into a sequence of unconstrained problems, that is to say the objective function and the constraints of the original problem are transformed to define a new objective function called an auxiliary or penalty function.

By gradually changing the effects of the constraints in the penalty function, a sequence of unconstrained problems is generated.

As the penalty function is being minimised at each step of the sequence, an efficient unconstrained minimisation algorithm had to be found.

Three unconstrained algorithms have been compared:

A direct search method (Simplex);
Two conjugate direction methods:

(a) Powell (64)'s method not requiring the calculation of derivatives;

(b) Fletcher and Powell's method requiring the calculation of derivatives.

All the methods have been written in the same language, ICL Algol 60, and have been tested with the same set of well-known standard test problems and some larger ones.

All the methods have been described followed by their respective results.

For overall comparison, the best results from each algorithm are considered and tabulated in function of the total number of function evaluations and in function of computer time.

We can, then, draw two conclusions:

(a) if, as some people suggest, the total number of function evaluations is more important, the Powell's method could be the more efficient of the methods considered;

(b) if computer time is more important, then S.U.M.T. could be the more efficient method.

CONTENTS

I

INTRODUCTION

This thesis deals mainly with a comparison of certain computational techniques used for the solution of non-linear constrained problems.

Recently, methods for solving constrained minimization problems by considering sequences of unconstrained problems have attracted considerable attention, the three methods considered here are of that type. They are:

(a) Sequential Unconstrained Minimization Technique, (S.U.M.T.), by Fiacco and McCormick;

(b) Kowalik, Osborne and Ryan's method;

(c) Powell's method for constrained problems.

The objective function and the constraints of the original problem are transformed to define a new objective function called auxiliary or penalty function.

By gradually removing the effect of the constraints in the penalty function, a sequence of unconstrained problems is generated that has solutions converging to a solution of the original problem.

As the penalty function is being minimized at each step of the sequence, an efficient unconstrained minimization algorithm had to be found. Therefore the first part of this thesis begins with a comparison of unconstrained algorithms.

(a) A direct search method (Simplex).

(b) A conjugate direction method not requiring the calculation of derivatives (Powell 64).

(c) A conjugate direction method requiring the calculation of derivatives (Fletcher and Powell).

Work has been considerably delayed at that point for the following reason.

Powell (64)'s method was proved to be a relatively efficient and quick method, attractive too as it did not require the calculation of derivatives so the constrained minimization algorithms had been implemented using it as a sub-routine.

Everything went fine as long as the standard test-problems were considered.

These problems are known to have an awkward behaviour but they are small in size.

Where larger problems were considered (eight variables and more) Powell (64)'s method failed and the programs had to be re-written using the Fletcher and Powell's method.

The largest problem solved was 20 x 17 and this was proved to be satisfactory.

II

UNCONSTRAINED OPTIMIZATION TECHNIQUES

2.1 MATHEMATICAL DESCRIPTION

2.1.1 Simplex Method

The Simplex method was introduced by Himsworth Spendley and Hest[19] in 1962 and developed by Nelder and Mead.[15]

It is a direct search method that is to say it compares the values of the objective function at a set of (n+1) vertices of a simplex.

A simplex is a geometric figure defined as follows: a set of (n+1) points in n-dimensional space forms a simplex. When the points are equidistant the simplex is said to be regular.

In the case n= 2 the corresponding figure is an equilateral triangle while n=3 is a tetrahedron. The principal idea of the method is that we can easily form a new simplex from the current one by reflecting one point in the hyperplane spanned by the remaining points.

If we reflect the point which gives the highest value to the objective function by another one we can expect that at the reflected vertex the function value will be lower, and we go on until the minimum or a sufficiently good approximation to the minimum is found. The problem is to minimize:

$$y = f(x) \text{ where } x \text{ is } 1 \times n \text{ vector}$$

Let us introduce the following notation: h is the suffix such that x_h is the vertex corresponding to $f(x_h) = \max f(x_i)$ $i=1, \ldots, n+1$; l is the suffix such that x_l is the vertex corresponding to $f(x_l) = \min f(x_i)$; x_o is the centroid of the points x_i with $i \neq h$.

At each stage in the process x_h is replaced by a new point. Three basic operations are used in the method:

(a) reflection;
(b) expansion;
(c) contraction.

A. The Reflection Operation

We generate the new point x_r as follows:

$$x_r = (1 + \alpha)x_o - \alpha x_h$$

where α, the reflection coefficient, is greater than unity. (or equal to unity)

Thus, x_r is on the line joining x_h and x_c, on the far side of x_o from x_h and α is the ratio of the distance $[x_r x_o]$ to $[x_h x_o]$.

If $f(x_r)$ lies between $f(x_h)$ and $f(x_l)$ then x_h is replaced by x_r and we start again with the new simplex (see footnote).

If $f(x_r) < f(x_l)$ then $f(x_r)$ is the new minimum; therefore we expand x_r to x_e.

B. The Expansion Operation

x_e is obtained by using the following relation:

$$x_e = \gamma x_r + (1 - \gamma) x_o$$

The expansion coefficient γ which is greater than unity is the ratio of the distance $[x_e x_o]$ to $[x_r x_o]$.

If $f(x_e) < f(x_l)$ we replace x_h by x_e but if $f(x_e) > f(x_l)$ then the expansion has failed, therefore we replace x_h by x_r and in either case we restart the process. If, after reflection, we find that $f(x_r) > f(x_i)$ then we define a new x_h to be either x_h or x_r whichever has the lowest $f(x)$ value and we make a contracting move.

C. The Contraction Operation

We generate x_c as follows:

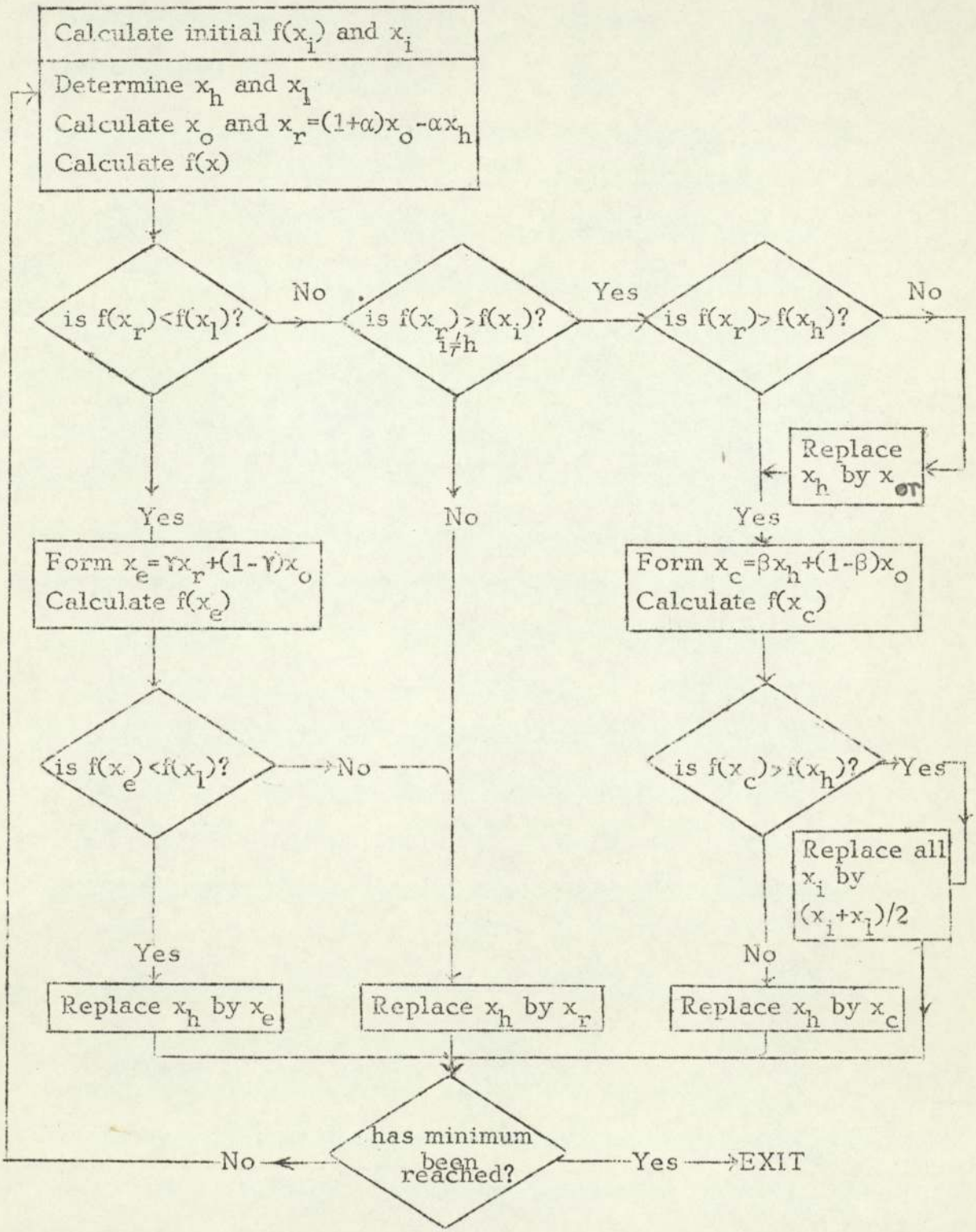
$$x_c = \beta x_h + (1 - \beta) x_o$$

It is necessary to say that the ultimate convergence criterion is tested each time before restarting the whole process.

The contraction coefficient β lies between 0 and 1 and is the ratio of the distance $\left[x_c x_o\right]$ to $\left[x_r x_o\right]$.

If $f(x_h) > f(x_c)$ x_h is replaced by x_c but if the contracted point is worse, that is to say $f(x_h) < f(x_c)$ then we replace all x_i by $\frac{1}{2}(x_i + x_l)$ and in either case we restart the whole process.

The whole algorithm can easily be schematised in the following flow diagram.



Ultimate Convergence Criterion

The stopping criterion suggested by Nelder and Mead is concerned with the variations in the function values over the simplex rather than with changes in the x's.

It takes the standard error form:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n+1}(f(x_i) - f(x_o))^2} < \varepsilon$$

where ε is a small preset positive value.

Comments on the Program

Various parameters are used in the program. The main ones are α, β and γ.

Nelder and Mead have carried out many experiments with different combinations. They find best results are achieved when $\alpha=1$, $\beta=\frac{1}{2}$, $\gamma=2$.

Four other parameters are also used.

Criter: stands for ε and is set at 10^{-8}.

Conver: value of the standard error of the $f(x_i)$

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n+1}(f(x_i) - f(x_o))^2}$$

whenever conver < criter the program stops.

Colimit: preset value for the maximum number of function evaluations.

Count: number equal to the number of function evaluations. Whenever count > colimit then the program stops.

Orp(1:n): Array of feasible starting point.

2.1.2 Powell (64)'s Method (1964)[16]

The key advantage of this method is that it does not require explicit evaluation of derivatives.

It is a quadratically convergent process which generates conjugate directions of search and will thus find the minimum of a quadratic form in a finite number of steps.

It is based on the following theorem: if x_1 is the minimum in a space containing the direction v, and x_2 is also the minimum in such a space, then the direction $(x_1 - x_2)$ is conjugate to v. Let f(x) be a general quadratic function:

$$f(x) = xAx + bx + c$$

By definition $\frac{\delta}{\delta\lambda}\left\{f(x_o + \lambda v)\right\} = 0$ at $\lambda = 0$. Therefore:

$$2\lambda vAv + v(2Ax_1 + b) = 0, \quad \lambda = 0$$

Also:

$$2\lambda v\,Av + v(2Ax_2 + b) = 0 \quad \lambda = 0$$

whence subtracting:

$$vA(x_1 - x_2) = 0$$

The directions v and $(x_2 - x_1)$ are conjugate (Fig. 1).

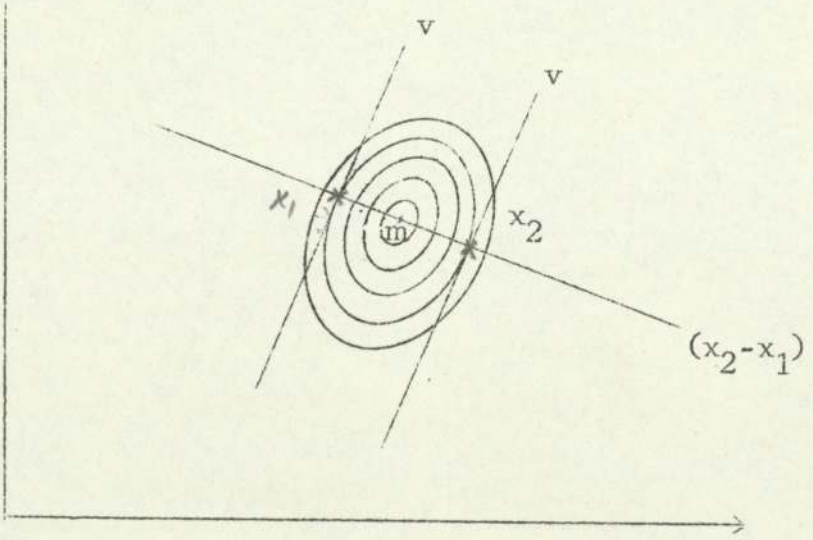


Fig. 1

This means that in the presented space, the minimum M of the quadratic function must be found along the vector (x_2-x_1). Hence it is only necessary to search for the minimum along the two v directions and finally along the vector (x_2-x_1).

For more general functions which are not quadratic the procedure is iterative and can be described as follows. We first assume that n independent directions $v_1, v_2, \ldots, v_n$ are given (for example the co-ordinate directions.)

1. For p=1, 2, ..., n, calculate λ_p so that $f(x_{p-1} + \lambda_p v_p)$ is a minimum and define $x_p = x_{p-1} + \lambda_p v_p$.

2. For p=1, 2, ..., n-1 replace v_p by v_{p+1}.

3. Replace v_n by (x_n-x_o).

4. Find λ so that $f(x_n + \lambda(x_n - x_o))$ is a minimum and replace x_o by $x_o + \lambda(x_n - x_o)$ where x_o is an arbitrary feasible starting point.

5. Repeat the procedure.

This procedure is illustrated in Fig. 2 in the case where n=2.

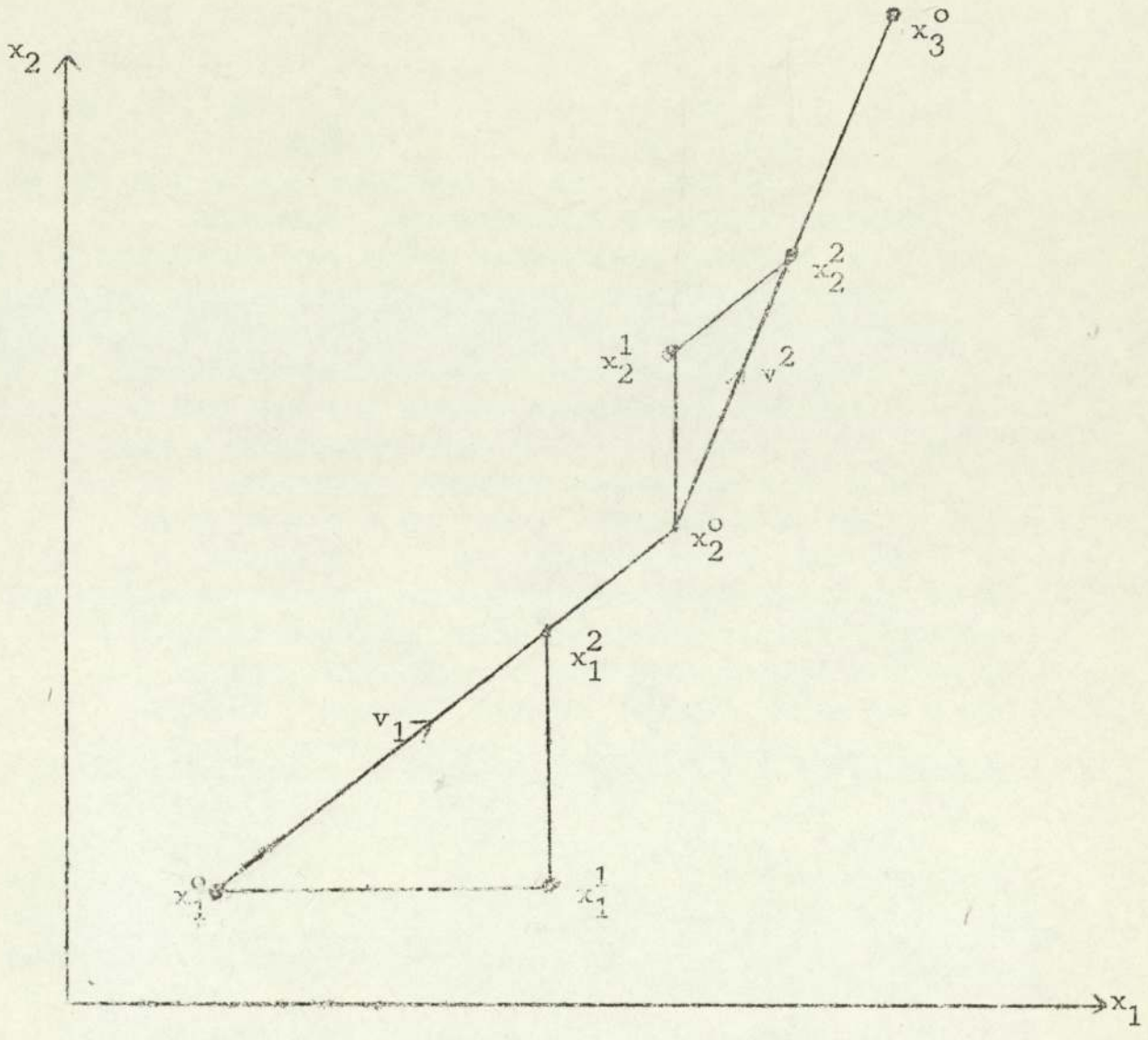


Fig. 2

x_1^o starting point and x_1^1 minimum moving along the x_1 co-ordinate.
x_1^2 minimum from x_1^1 along the x_2 co-ordinate.
x_2^o minimum point along the direction $v_1 = (x_1^2 - x_1^o)$ and repeat.

The final point x_3^o must be the minimum of the quadratic function since v_1 and $v_2 = (x_2^2 - x_2^o)$ are conjugate.

Theoretically this procedure converges to the minimum of a quadratic function in n iterations. Practical applications have shown, however, that there is a need to modify the basic procedure

in order to achieve a satisfactory rate of convergence. The basic procedure may occasionally select directions which are nearly dependent and do not span full parameter space. Powell has introduced a modification which ensures that the efficiency of the directions $v_1, v_2, \ldots, v_n$ is never less than that of the original independent co-ordinate system.

For this purpose he uses a certain criterion. He rejects the direction generated at the current stage if the criterion fails and he computes another cycle of descent steps using the current directions otherwise he accepts this new direction.

The procedure described above is then modified as follows:

1. for $p=1, 2, \ldots, n$ calculate λp so that $f(x_{p-1} + \lambda_p v_p)$ is a minimum and define $x_p = x_{p-1} + \lambda_p v_p$;

2. find the integer m, $1 \leq m \leq n$, so that $\{f(x_{m-1}) - f(x_m)\}$ is a maximum and define $\Delta = f(x_{m-1}) - f(x_m)$;

3. Calculate $f_3 = f(2x_n - x_o)$ and define $f_1 = f(x_o)$ and $f_2 = f(x_n)$;

4. if either $f_3 \geq f_1$ and/or $(f_1 - 2f_2 + f_3).(f_1 - f_2 - \Delta)^2 \geq (f_1 - f_3)^2$ use the old directions $v_1, v_2, \ldots, v_n$ for the next iteration and use x_n for the next x_c otherwise;

5. defining $v = (x_n - x_o)$ calculate λ so that $f(x_n + \lambda(x_n - x_o))$ is a minimum use $v_1, v_2, \ldots, v_{m-1}, v_{m+1}, v_{m+2}, \ldots, v_n, v$, as the directions and $x_n + \lambda v$ as the starting point for the next iteration.

If this modification is used, a conjugate direction is thrown away and more than n iterations are required to find the exact minimum. Nevertheless, it was necessary whenever large problems are solved.

To justify this criterion, Powell uses the following theorem:

Let vectors $p_1, \ldots, p_n$ be scaled so that $p_i^T A p_i = 1$ $i = 1, 2, \ldots, n$. Let X be the matrix whose columns are the vectors p_i. Then the determinant of X is maximum if and only if the directions are mutually conjugate.

The consequence of this is that $v_1, v_2, \ldots, v_n$ should be chosen to make the determinant as large as possible.

The criterion is applied by using the new direction v, defined by an iteration, if it causes the determinant to increase, or by rejecting the direction, the replacement of which causes the new determinant to be largest.

It will now be proved that the direction which should be discarded, if any, is v_m, $1 \leq m \leq n$ where m is such that $\{f(x_{m-1}) - f(x_m)\}$ is a maximum.

As $f(x_i)$ is a minimum in the direction v_i, if v_i is scaled so that

$$v_i^T A v_i = 1$$

the displacement from x_{i-1} to x_i is:

$$\sqrt{\left[f(x_{i-1}) - f(x_i)\right]} \cdot v_i = \alpha_i v_i$$

The direction defined by the iteration is:

$$x_n - x_o = \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n$$

So if $x_n - x_o = \mu v_p$ where $v_p^T A v_p = 1$, the effect of replacing the column vector v_i by the vector v_p is to multiply the determinant of directions by α_i/μ.

Consequently the direction to be discarded, if any, is that for which α_i is largest and this is the direction v_m.

This replacement should be made only if $\alpha_m \geqslant \mu$ and μ is calculated by the means of f_1, f_2 and f_3 (values defined in the description of the procedure). The predicted stationary value of the function along the new direction is:

$$f_s = f_2 - \frac{1}{8}\frac{(f_1 - f_3)^2}{(f_1 - 2f_2 + f_3)}$$

f_s is a minimum if:

$$f_1 - 2f_2 + f_3 > 0$$

If the above second difference term is negative, a new direction should certainly be defined, otherwise:

$$\mu = \sqrt{(f_1 - f_s)} \pm \sqrt{(f_2 - f_s)}$$

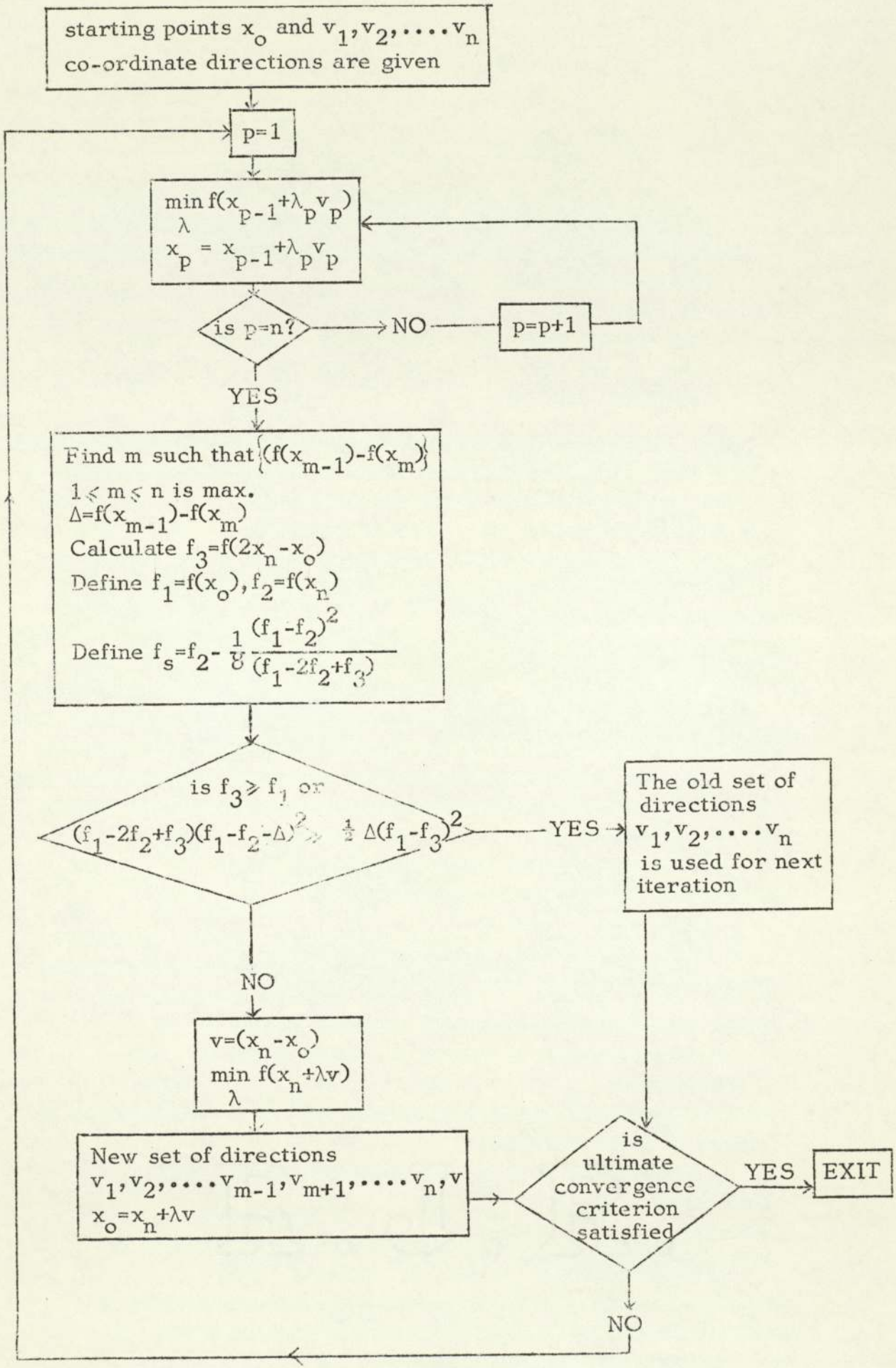
+ or - sign depending on whether f_3 is greater or less than f_1. In the former case it is obvious that the old directions should be used again, in the latter case new directions should be defined only if:

$$\sqrt{\Delta} \geqslant \sqrt{(f_1 - f_s)} - \sqrt{(f_2 - f_s)}$$

The above results have been condensed in the criterion that $x_n - x_o$ should not be used for the next iteration if and only if, either $f_3 \geqslant f_1$ and/or $(f_1 - 2f_2 - f_3)(f_1 - f_2 - \Delta)^2 \geqslant \frac{1}{2}\Delta(f_1 - f_3)^2$.

Because the modified procedure cannot cause the determinant of directions to decrease the efficiency of the direction of search $v_1, v_2, \ldots v_n$ is never less than that of the original co-ordinate directions. If these are poor, improved directions will be found easily.

The whole procedure can easily be schematised in the following diagram.


starting points x_o and $v_1, v_2, \ldots v_n$ co-ordinate directions are given
p=1
$\min_{\lambda} f(x_{p-1}+\lambda_p v_p)$
$x_p = x_{p-1}+\lambda_p v_p$
is p=n?
NO
p=p+1
YES
Find m such that $\{f(x_{m-1})-f(x_m)\}$
$1 \leq m \leq n$ is max.
$\Delta=f(x_{m-1})-f(x_m)$
Calculate $f_3=f(2x_n-x_o)$
Define $f_1=f(x_o)$, $f_2=f(x_n)$
Define $f_s=f_2-\frac{1}{2}\frac{(f_1-f_2)^2}{(f_1-2f_2+f_3)}$
is $f_3 \geq f_1$ or $(f_1-2f_2+f_3)(f_1-f_2-\Delta)^2 \geq \frac{1}{2}\Delta(f_1-f_3)^2$
YES
The old set of directions $v_1, v_2, \ldots v_n$ is used for next iteration
NO
$v=(x_n-x_o)$
$\min_{\lambda} f(x_n+\lambda v)$
New set of directions $v_1, v_2, \ldots v_{m-1}, v_{m+1}, \ldots v_n, v$
$x_o=x_n+\lambda v$
is ultimate convergence criterion satisfied
YES
EXIT
NO

Ultimate Convergence Criterion

Powell's convergence criterion is more concerned with the changes over the variables.

1. The procedure is applied until the change on each variable is one tenth of the required accuracy. This point is called a.

2. Each variable is then increased by ten times the required accuracy.

3. The procedure is applied again until the change on each variable is one tenth of the required accuracy. This point is called b.

4. The minimum on the line through a and b is then found, it is called c.

5. If (a-c) and (b-c) are less than one tenth of the required accuracy in the corresponding variables then ultimate convergence has been reached, else,

6. include the direction (a-c) in place of v_1 and start again.

Comments on the Program

The procedure Powell 64 has eight parameters, defined as follows:

1. X: array composed of the initial feasible starting points.

2. E: array composed of the required accuracy value for each variable in this case it is set at 10^{-8}.

3. N: number of variables.

4. F: value of function.

5. Escale: an integer number which defines the step for each linear search so X will not be changed by more than Escale x E.
   Difficulties arose when choosing Escale. Not knowing the behaviour of the function it was difficult to determine the step and therefore a method of trial and error was used.

It was found that Escale should be at least equal to one fourth of the inverse of the accuracy.

6. Iprint: controls the printing.

   (a) Iprint = 0 no printing

   (b) Iprint = 1 the variables and the function will be printed after every search along a line

   (c) Iprint = 2 the variables and the function will be printed after every iteration (n+1) searches along a line.

7. Icon: provides an alternative convergence criterion. Usually it is satisfactory if Icon=1. However, if a low accuracy is required, Icon is set equal to two but the execution time might be increased by as much as 30%.

8. Maxit: maximum number of iterations required. The routine will be left regardless after Maxit iterations have been completed.

Zangwill Modification of the Powell (64)'s Method

In 1967 Zangwill[22] published a paper suggesting a modification to the Powell procedure. A counter example was found which reveals that Powell's method does not converge to the minimum of a quadratic in a finite number of iterations but will not converge in any number of iterations.

This new procedure, based upon Powell's theorems can be written as follows:

Let $v_1, v_2, \ldots, v_n$ be the co-ordinate directions and assume they are normalised to unit length. The starting points $x_1^o, \ldots, x_n^o$, and the normalised directions $\xi_1^1, \xi_2^1, \ldots, \xi_n^1$ are given. Calculate λ_n^o to minimise $f_1(x_n^o + \lambda_n^o \xi_n^1)$ and let $x_{n+1}^o = x_n^o + \lambda_n^o \xi_n^1$.
Set t=1 and go to iteration k with k=1. Iteration k: x_{n+1}^{k-1}, ξ_r^k, r=1, ..., n and t are given.

Step (i) Find α to minimise $f(x_{n+1}^{k-1} + \alpha v_t)$

Update t by:

$$t \leftarrow \begin{cases} t+1 & \text{if } 1 \leq t < n \\ 1 & \text{if } t = n \end{cases}$$

If $\alpha \neq 0$ let $x_o^k = x_{n+1}^{k-1} + \alpha v_t$.

If $\alpha = 0$ repeat step (i). Should step (i) be repeated n times in succession, then stop; the point x_{n+1}^{k-1} is optimal.

Step (ii) for $r=1, \ldots, n$ calculate λ_r^k to minimise $f(x_{r-1}^k + \lambda_r^k \xi_r^k)$ and define $x_r^k = x_{r-1}^k + \lambda_r^k \xi_r^k$.

Let $\xi_{n+1}^k = (x_n^k - x_{n+1}^{k-1}) / \| x_n^k - x_{n+1}^{k-1} \|$

Determine λ_{n+1}^k to minimise $f(x_n^k + \lambda_{n+1}^k \xi_{n+1}^k)$ and set $x_{n+1}^k = x_n^k + \lambda_{n+1}^k \xi_{n+1}^k$

Define $\xi_r^{k+1} = \xi_{r+1}^k \quad r = 1, 2, \ldots, n$

Go to iteration k where k becomes k+1

No results are available yet to enable an assessment to be made of the rate of convergence of this procedure relative to the Powell (64)'s procedure.

2.1.3 Fletcher and Powell's Method, 1965

This method is also sometimes called the method of Davidon, Fletcher and Powell[8] as the idea is based upon a procedure described by Davidon (1959).[2]

Like the previous method described, it uses conjugate directions and has quadratic convergence, therefore gets to the optimum at a finite number of steps (for quadratic function).

It also requires the gradient vector $g_i = \frac{\delta f}{\delta x_i}$ to be defined analytically at each point.

We know that two directions v_i and v_j are said to be conjugate with respect to A if $v_i^T A v_j = 0$ for $i \neq j$.

A being a positive definite matrix. Is it possible to define A^{-1} as a function of these conjugate directions?

Consider the matrix:

$$\sum_{i=1}^{p} \alpha_i v_i v_i^T$$

We have for s=1, 2, ..., p

$$\left(\sum_{i=1}^{p} \alpha_i v_i v_i^T\right) A v_s = \alpha_s v_s v_s^T A v_s = v_s$$

provided $\alpha_s = 1/v_s^T A v_s$

This gives for p=n:

$$A^{-1} = \sum_{i=1}^{n} \frac{v_i v_i^T}{v_i^T A v_i}$$

This suggests an iterative scheme in which the best approximation to the inverse $A^{-1} = H$ is used to define the next direction of search by:

$$v_{i+1} = - H_i \nabla F(x^{i+1})$$

and the results of this search are then used to improve the approximation to the inverse, but in the mean time the successive directions generated must stay conjugate.

Knowing the matrix $A_{ij} = \frac{\delta^2 f}{\delta x_i \delta x_j}$ the displacement between the point x_i and x_{i-1} is $v_i = - A_{i-1}^{-1} g_i$, then $x_{i+1} = x_i + \lambda_i v_i$ and updating we have:

$$H_i = H_{i-1} + \frac{v_i v_i^T}{v_i^T A v_i} + B_i$$

where B_i is a correcting term and H_i an approximation of A^{-1}.

If $v_1, \ldots, v_i$ are conjugate:

$$v_s^T A H_{i-1} = v_s^T \qquad s = 1, 2, \ldots, i-1$$

we also know that if $v_1, \ldots v_i$ are conjugate:

$$-v_s^T A v_i = v_s^T A H_{i-1} g_i = v_s^T g_i = 0$$

therefore we must choose B_i to satisfy $v_s^T A H_i = v_s^T$ $s = 1, 2, \ldots, i$.

For i=s we have:

$$v_i^T A(H_{i-1} + \frac{v_i v_i^T}{v_i^T A v_i} + B_i) = v_i^T$$

whence:

$$(g_{i+1} - g_i)^T (H_{i-1} + B_i) = 0$$

Writing $y_i = g_{i+1} - g_i$:

$$B_i = - \frac{H_{i-1} y_i y_i^T H_{i-1}}{y_i^T H_{i-1} y_i}$$

To update H_{i-1} in a general function we need to modify the term $v_i^T A v_i$. We know that:

$$v_i^T A v_i = \frac{(x_{i+1} - x_i)^T}{\lambda_i} A v_i$$

It is also equal to:

$$\frac{(g_{i+1} - g_i)^T}{\lambda_i} v_i \quad \text{or} \quad \frac{1}{\lambda_i} g_i^T H_{i-1} g_i$$

$$v_i^T A v_i = \frac{1}{\lambda_i} g_i^T H_{i-1} g_i$$

Setting:

$$A_i = \frac{v_i v_i^T}{v_i^T y_i} \quad \text{where } y_i = g_{i+1} - g_i$$

the updated H matrix now takes the form:

$$H_i = H_{i-1} + A_i + B_i$$

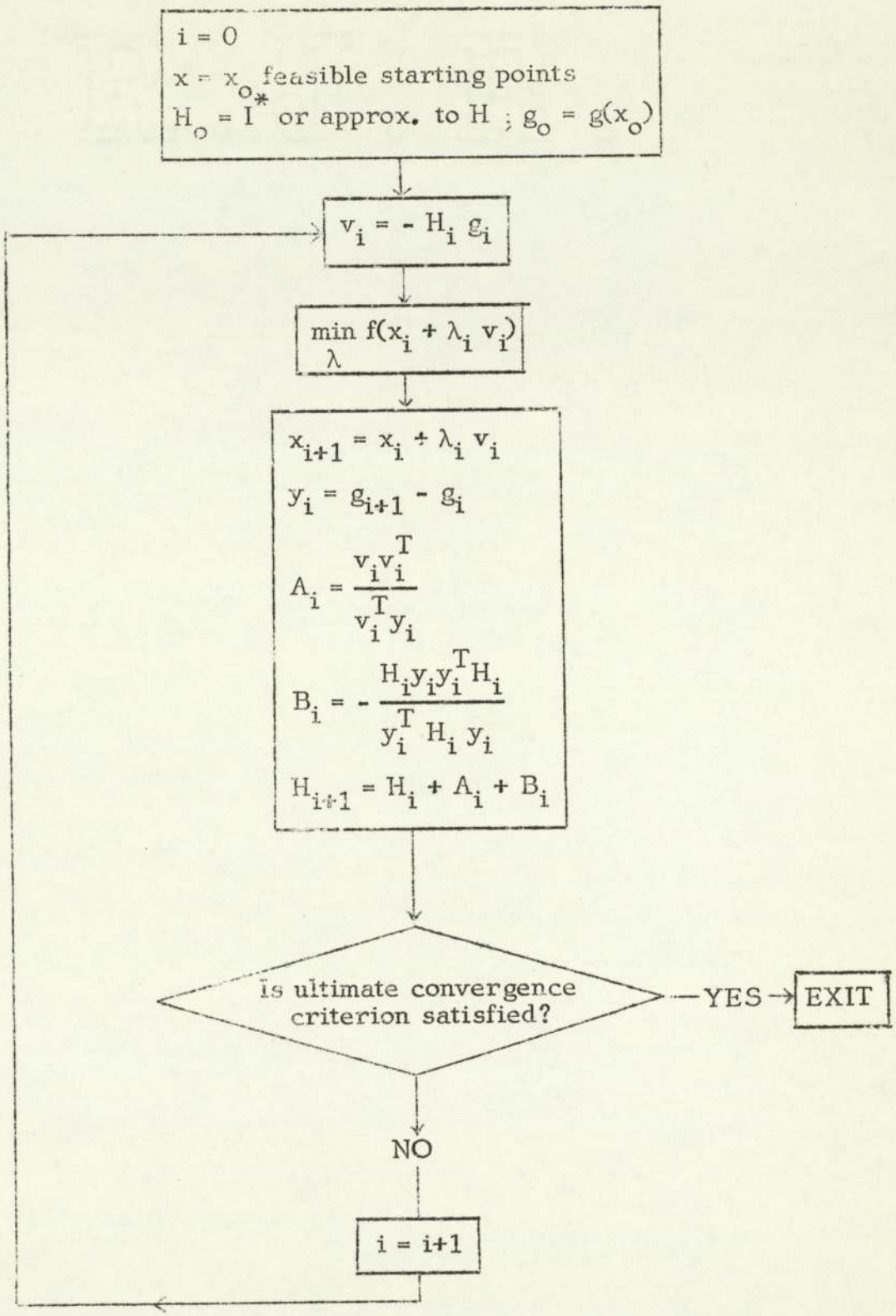
We can now state the procedure as follows:

Given x_o and $g_o = g(x_o)$

1. Compute $v_i = - H_{i-1} g_i$
2. Compute λ_i to minimise $f(x_i + \lambda_i v_i)$
3. Set $x_{i+1} = x_i + \lambda_i v_i$
4. Compute $H_i = H_{i-1} + A_i + B_i$

In the first step of the iteration it is customary to set $H_o = I$. However, if an estimate of A^{-1} is known, this can be used provided it is positive definite. If H_o is positive definite initially it has been proved that all subsequent H_i are also positive definite. As $H_n = A^{-1}$ by construction it must be positive definite so that if H_o does not satisfy this condition there is the possibility of a breakdown in the calculation.

This algorithm is easily schematised in the following flow chart.



* H_o is usually set up equal to an identity matrix at the beginning, unless an approximation to the Hessian matrix is known.

Convergence Criterion

The process is terminated whenever two successive values of f are equal or if a new value of f is larger than the previous one (due to rounding errors) or when the first derivative of f nearly vanishes.

Comments on the Program

The procedure Flepomin[20] is a nine parameter procedure.

N = number of variables

X = array of feasible starting points

F = function value

EPS = tolerance used in terminating the procedure when the first derivative of f nearly vanishes. Therefore it is set in a very small quantity. In this case it is set to 10^{-8}.

FUNCT = procedure calculating the value of the function and the derivatives.

CONV = Boolean variable equal to true if convergence exists or false otherwise.

LIMIT = Integer variable defining the number of iterations required. CONV will be equal to false whenever the number of iterations has exceeded LIMIT and the process will be terminated regardless.

H = array $\frac{n(n+1)}{2}$

LOADH = Boolean variable indicating whether or not an approximation to the inverse of the matrix of second derivatives is available.

Recent Variations of the Fletcher and Powell's Algorithm[9]

One of the main features of the algorithm described earlier is that an approximation to H is kept and updated using the formula:

$$H_{i+1} = H_i + \frac{v_i v_i^T}{v_i^T y_i} - \frac{H_i v_i v_i^T H_i}{y_i^T H_i y_i}$$

where $v_i = - \lambda H_i g_i$

$y_i = g_{i+1} - g_i$

The correction v_i is taken as a multiple λ of a direction of search $s_i = H_i g_i$.

Though this algorithm proved a very powerful one so far, it has some inconvenient features that Fletcher, in his recent article, tried to overcome. The main one is the need to solve the sub-problem of finding the optimum λ at each iteration, ie. the linear search. As it requires the evaluation of the function and the gradient for a number of different value of λ and interpolates according to some strategy, until a sufficiently accurate minimum is obtained, considerable computer time is needed.

The linear search also has another disadvantage because of the special circumstances which can arise, eg. a minimum may not exist.

So it would be convenient to find λ other than by finding λ which minimises $f(x_i + \lambda s_i)$, bearing in mind that the main importance of the optimum linear search is that it generates conjugate directions leading to the property that for a quadratic function convergence occurs within less than n iterations.

So Fletcher tried to find out if that property could be kept for variable metric algorithms not requiring optimum linear searches but based upon a revised formula for updating H.

The only solution, however, was to abandon the property of quadratic convergence and to replace the linear search by another process ensuring an efficient decrease of the value of the function at each iteration and this could be produced by the retention of the positive definiteness in H.

So Fletcher suggested a new updating formula for H which guarantees positive definiteness:

$$H_{i+1} = H_i - \frac{vy^T H_i}{v^T y} - \frac{H_i y v^T}{v^T y} + \left(1 + \frac{y^T H_i y}{v^T y}\right) \frac{vv^T}{v^T y}$$

This formula can only be used under certain conditions. If:

$$A: \quad y^T A^{-1} y \geq y^T H y$$

then H is smaller than A^{-1} and the new formula for updating H is used. If, however, $y^T A^{-1} y < y^T H y$ then H is larger than A^{-1} and the original updating formula is used.

The new formula used whether or not the inequality (A) holds has been inserted into the Flepomin procedure.

Another modification has also be implemented. We usually start with H set equal to I (the unit matrix) and sometimes this proved to be quite inefficient as H can be much greater than the local A^{-1} then any direction which reduces F would be considerably less than -Hg and a considerable number of extra function evaluations would be required at each iteration.

This only occurs at up to and including the n^{th} iteration after which a step of -Hg is almost always successful. In practice, a step length λ has been kept, derived from the previous iteration and used to generate an initial $v = -\lambda Hg$.

However, the program reverts to the basic algorithm after the n^{th} step.

We shall see from the tables that the results obtained, once these modifications have been included, are far better than those obtained with the original Fletcher and Powell algorithm.

2.2 Computational Results

Each algorithm has been tested with the following test problems:

Function 1

Rosenbrock (banana shape) function

minimise $f = 100 (x_2 - x_1^2)^2 + (1 - x_1)^2$

starting points $x = (-1.2, 1)$

Function 2

Fletcher and Powell's function with quartic singular Hessian

$$\text{minimise } f = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$

starting points x=(3, -1, 0, 1)

Function 3

Fletcher and Powell's (hellical valley) function

$$\text{minimise } f = \left[100\ x_3 - 100\theta(x_1, x_2)\right]^2 + \left(\sqrt{(x_1^2 + x_2^2)} - 1\right)^2 + x_3^2$$

where $2\pi\theta(x_1, x_2) = \text{Arctan}(x_2/x_1)$, if $x_1 > 0$ or $\pi + \text{Arctan}(x_2/x_1)$ if $x_1 < 0$. Starting points $x = (-1, 0, 0)$.

Function 4

Four dimensional banana shaped function (Colville)

$$\text{minimise } f = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 + 90(x_3^2 - x_4)^2 + (1 - x_3)^2 + 10.1\left[(x_2 - 1)^2 + (x_4 - 1)^2\right] + 19.8(x_2 - 1)(x_4 - 1)$$

starting points x = (-3, -1, -3, -1)

Function 5

Box's function[8]

$$\text{Minimise } f(a_1, a_2, a_3) = \sum_x \left[a_3(e^{-a_1 x} - e^{-a_2 x}) - a_3(e^{-x} - e^{-10x})\right]^2$$

where summation is over the values x = 0.1(0.1)1. Nine sets of starting points were used.

Function 6

Watson's function

$$\text{minimise } f = \sum_{i=1}^{30} \left\{ \sum_{j=1}^{m} (j - 1) x_j y_i^{j-2} - \left(\sum_{j=i}^{m} x_j y_i^{j-1} \right)^2 - 1 \right\}^2 + x_1^2$$

where $y_i = (i - 1)/29$.

m has been chosen equal to 6. Starting points x = (0, 0, 0, 0, 0, 0).

This function results from an attempt to approximate to the solution of the differential equation:

$$\frac{dz}{dx} - z^2 = 1 \qquad z(0) = 0$$

in $0 \leq x \leq 1$ by a polynomial of degree m, by minimising the sum of squares of the residuals at selected points.

When running a program, the total mill time is given. This includes the compilation time, the consolidation time, the program run and the operating system administration. As the operating system administration may vary quite substantially - for reasons difficult to explain - the total mill time is quite unreliable if a comparison is to be made in function of time. Therefore, here the comparison is made as function of the run time of the program excluding compilation and consolidation. Though, again, this might be quite unreliable for programs taking very little run time (up to 20 seconds). The first tables show the number of function evaluations necessary to obtain the required accuracy. This is shown for the four different methods we considered. The other tables show the run time and the number of function evaluations for each problem and for each algorithm compared.

N. B. The same accuracy 10^{-8} has been used in the three different algorithms.

SIMPLEX METHOD

$\alpha = 1, \quad \beta = 0.5, \quad \gamma = 2$

	Function 1	Function 2	Function 3	Function 4	Function 5	Function 6
EV	f	f	f	f	f	f
1	24.2	215	2500	19192	2.087	30
20	3.81	7.36	637	79.9	0.034	1.56
40	1.61	8.16	70.6	10.01	0.026	1.51
60	1.04	3.34	16.77	7.95	0.025	0.0875
80	0.47	0.62	10.73	7.73	0.014	0.027
100	0.694	0.138	7.41	7.33	9.09×10^{-4}	0.015
120	0.001	0.005	3.08	7.23	58×10^{-5}	0.013
140	58×10^{-8}	0.005	1.31	7.02	2.54×10^{-6}	8.4×10^{-3}
160		8.99×10^{-4}	0.10	6.44	6.08×10^{-8}	7.5×10^{-3}
180		5.35×10^{-5}	3.86×10^{-3}	5.46		6.9×10^{-3}
200		47×10^{-8}	27×10^{-6}	4.41		6.81×10^{-3}
300				2.57		6.72×10^{-3}
400				0.81		5.52×10^{-3}
500				4.56×10^{-3}		2.37×10^{-3}
600				6.3×10^{-8}		2.288×10^{-3}
2000						*2.28×10^{-3}

* To achieve an accuracy of 10^{-8}

The Simplex method has been successful with any set of starting points for the Box function. However, for clarity the results for the set (0, 20, 1) only are shown in this table. The number of function evaluations is approximate to the nearest twenty, or hundred if it is more than 200.

	Function 1	Function 2	Function 3	Function 4	Function 5	Function 6
EV	f	f	f	f	f	f
1	24.2	215	2500	19192	2.087	30
20	3.7	6.1	129	9120	9.14×10^{-4}	2.3
40	2.4	4.13	10.9	62	3.35×10^{-5}	0.53
60	1.07	0.75	0.3	25	9×10^{-11}	0.19
80	0.51	0.029	4.63	12	9×10^{-18}	0.12
100	0.10	5.3×10^{-3}	1.42	7.88		0.037
120	5.8×10^{-3}	3.58×10^{-3}	0.32	7.87		0.027
140	9×10^{-7}	2.1×10^{-5}	0.028	7.87		0.014
160	2.04×10^{-19}	1.53×10^{-6}	9×10^{-7}	7.87		0.014
180		2.40×10^{-7}	8×10^{-11}	7.84		0.011
200		1.95×10^{-7}	6.66×10^{-16}	7.80		0.011
300		1.161×10^{-13}		7.08		2.35×10^{-3}
400				4.58		2.28×10^{-3}
500				2.27		2.28×10^{-3}
600				0.82		
700				0.01		
800				5.1×10^{-12}		

Here again the number of function evaluations is approximate to the nearest twenty or hundred if it is more than 200. The set (0, 20, 1) of starting points has been considered for the function 5. More details of the results obtained with the same function are given now.

It was mentioned earlier on in the description of function 5 that nine sets of starting points were used. They are as follows:

Set	a_1	a_2	a_3	f
1	0	20	1	2.087
2	2.5	10	10	275.881
3	0	0	10	306.401
4	0	10	1	1.885
5	0	10	10	213.673
6	0	10	20	1031.154
7	0	20	0	9.706
8	0	20	10	209.280
9	0	20	20	1021.655

The optimum $a_1 = 10$, $a_2 = 1$, $a_3 = -1$ has never been obtained with any combination of methods and starting points tried to date.

There is, however, the continuum of optima $f = 0$, corresponding to $a_3 = 0$, $a_1 = a_2$ on which various of the methods found solutions with some starting points. This was rarely the case with the nine starting points quoted above and in the majority of these cases the desired optimum was found by making an alternative selection of initial step-lengths.

Powell (64)'s method was not successful with every set of starting points as we can see from the following table.

Set	2	3	5	6	8	9
	f 275.881 F	f 306.401 F	f 213.673 F	f 1031.154 F	f 209.280 F	f 1021.655 F

Set	1	4	7
Ev			
1	2.087	1.885	9.706
20	9.14×10^{-4}	2.8×10^{-4}	$1. \times 10^{-3}$
40	3.35×10^{-5}	6.5×10^{-6}	1.2×10^{-8}
60	9×10^{-11}	1×10^{-11}	1.1×10^{-8}
80	9×10^{-18}	1.19×10^{-22}	1.08×10^{-8}
100			1.52×10^{-15}

For the sets (2, 3, 5, 6, 8, 9) this method has failed. We can notice that these sets give the highest starting values for f. They produced the following solution:

$$a_1 \simeq 0.61 \quad a_2 \rightarrow \infty \quad a_3 \simeq 1.32 \quad f \simeq 0.076$$

ie. regarding the problem as one of curve fitting, this method has effectively eliminated a_2 from the problem and then endeavoured to fit one exponential and a multiplicative factor to the data. This failure stems from the fact that the method set out to locate the minimum along a line too precisely.

Any method which does not find the minimum along a line, for instance the Simplex Method, could not fail in this way.

For the sets 1 and 4 the same minimum values were obtained for the objective function and the variables, that is to say, respectively: 0, 1, 1, 1. For the set 7 there is the continuum of optima f = 0 corresponding to $a_1 = a_2$, $a_3 = 0$.

FLETCHER AND POWELL's METHOD (FLEPOMIN)

	function 1	function 2	function 3	function 4	function 5	function 6
ev	f	f	f	f	f	f
1	24.2	215	2500	19192	2.087	30
20	3.50	0.037	15.4	5.69	0.0226	1.76
40	0.122	3.50×10^{-4}	2.46	0.846	2.8×10^{-3}	9.88×10^{-3}
60	0.0522	5.33×10^{-9}	4.56×10^{-6}	0.364	5.2×10^{-5}	7.84×10^{-3}
80	0	1.8×10^{-13}	5.05×10^{-35}	0.075	9.79×10^{-22}	6.81×10^{-3}
100		3.68×10^{-22}		1.16×10^{-10}		6.45×10^{-3}
120						5.95×10^{-3}
140						5.17×10^{-3}
160						4.56×10^{-3}
180						3.63×10^{-3}
200						3.01×10^{-3}
300						2.57×10^{-3}

The number of function evaluations is approximate to the nearest twenty or hundred if it is more than two hundred and for clarity concerning function 5, the results for the set (0, 20, 1) only are shown in this table.

FLETCHER AND POWELL's METHOD MODIFIED BY FLETCHER (Flepomin Modified)

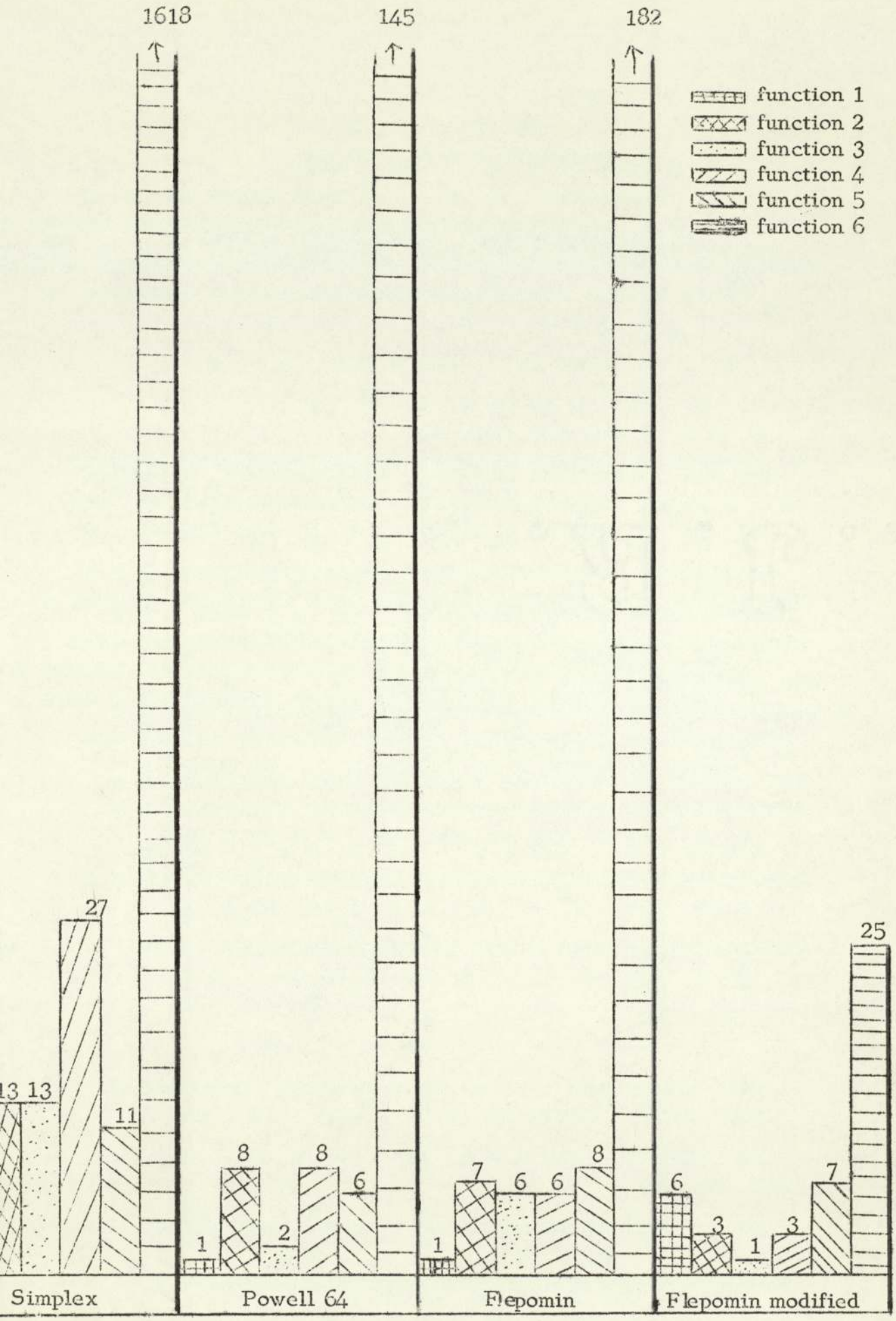
	function 1	function 2	function 3	function 4	function 5	function 6
ev	f	f	f	f	f	f
1	24.2	215	2500	19192	2.087	30
10	3.88	154.36	9.986	22.50	0.026	0.053
20	3.07	3.24×10^{-3}	2.89	14.02	0.0132	9.67×10^{-3}
30	2.84	2.55×10^{-4}	0.0936	0.750	7.06×10^{-3}	257×10^{-3}
40	1.307	2.29×10^{-8}	2.22×10^{-22}	0.727	26×10^{-6}	
50	1.12	2.80×10^{-10}		0.702	2.1×10^{-21}	
60	0.109	5.20×10^{-14}		1.19×10^{-3}		
70	0.020			2.11×10^{-20}		
80	1.129×10^{-11}					
85	2.11×10^{-22}					

Here again the number of functions evaluations is approximate to the nearest 10.

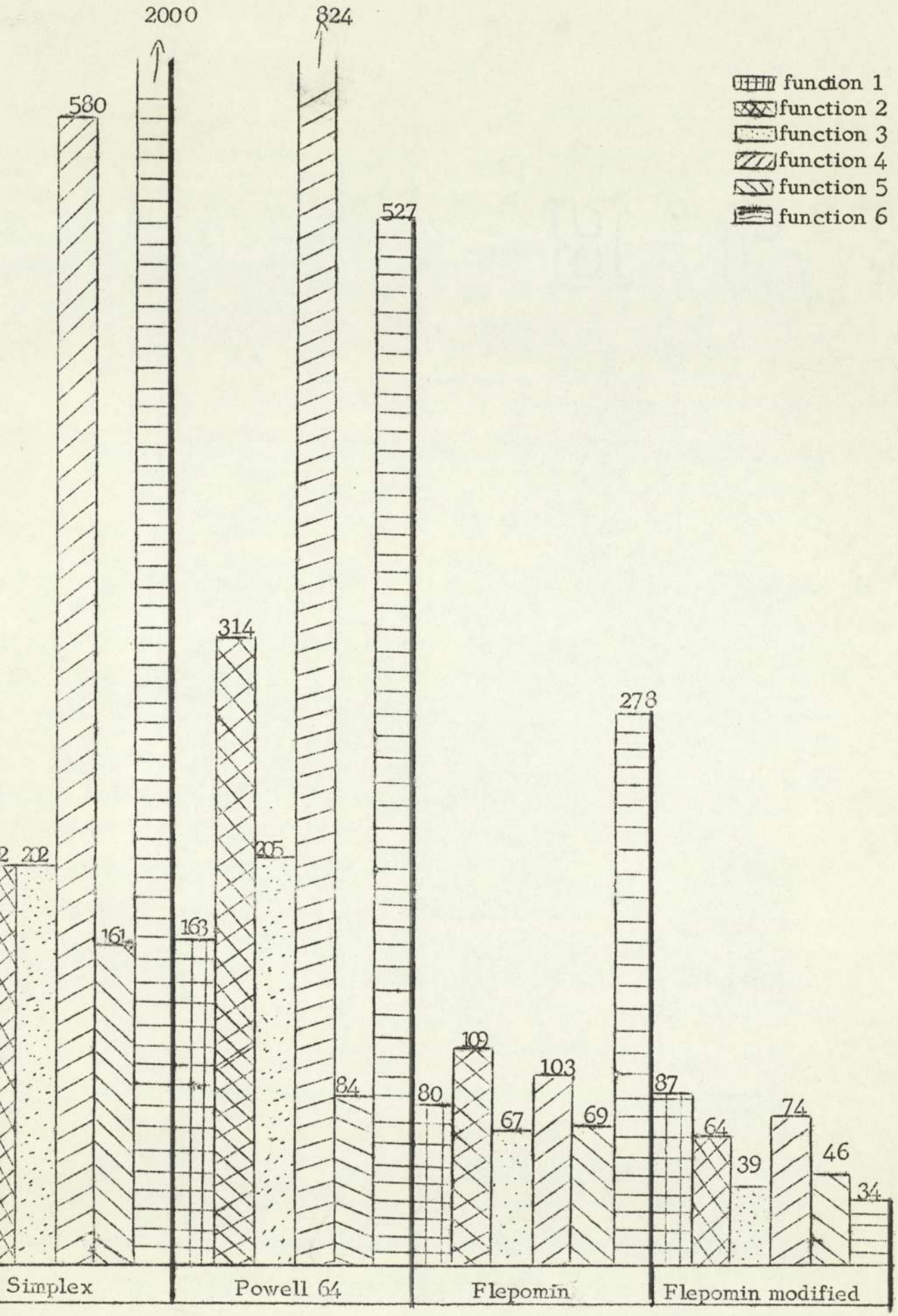
- upper corner : Actual run time of the program
- lower corner : Number of function evaluations

	Simplex Method	Powell 64	Flepomin	Flepomin modified
Function 1	3 / 147	1 / 163	1 / 80	6 / 87
Function 2	13 / 202	8 / 314	7 / 109	3 / 64
Function 3	13 / 202	2 / 205	6 / 67	1 / 39
Function 4	27 / 580	8 / 324	6 / 103	3 / 74
Function 5	11 / 161	6 / 84	8 / 69	7 / 46
Function 6	1618 / 2000	145 / 527	182 / 278	25 / 34

Run time in seconds



ber of function evaluations



2.3 Conclusions

The main purpose of this study of algorithms for unconstrained non-linear optimisation problems was to find the most efficient method which could be used as a sub-routine when writing programs for solving non-linear constrained optimisation problems.

At a glance, from the last tables, we can see that the Simplex method is a rather lengthy process considering the number of function evaluations and the run time, both are higher than those of the two other methods. However, we must say that the method will converge in some cases where the others failed as they try to locate the minimum along a line too precisely, eg. function 5.

The results obtained by the Fletcher and Powell's method and the Powell (64)'s method, both performed remarkably similarly and both possess quadratic convergence, ie. the property that they will converge to the minimum of a quadratic function in a finite number of steps and although such functions rarely occur in practice, it is nevertheless found that methods with this feature converge more rapidly, particularly, of course, in the vicinity of the optimum.

However, one advantage of the Powell (64)'s method over the Fletcher and Powell's method is that it does not require the calculation of the derivatives and this is why, at first, the Powell (64)'s method was chosen to be the common sub-routine for the programs solving the constrained non-linear optimisation problems.

Unfortunately, it will be shown in the next chapter, that the Powell (64)'s method failed when trying to solve larger constrained non-linear optimisation problems (eight variables and more).

The modified version of the Fletcher and Powell's method would have then been the obvious choice for an alternative sub-routine, but the modification suggested by Fletcher for the Fletcher and Powell's method[9] had not been published then. Therefore the only choice left was to rewrite the programs using the Fletcher and Powell's method.

III

CONSTRAINED OPTIMIZATION TECHNIQUES

3.1 MATHEMATICAL DESCRIPTION

3.1.1 Kowalik, Osborne and Ryan's Technique[12]

This recent method due to Kowalik, Osborne and Ryan is in fact a method combining a modification of a method due to Morrison[13] and a method due to Schmit and Fox[18] to bracket the optimal value of the function.

First of all let us consider Morrison's method. Consider the problem:

Minimise f(x)

where f is a scalar function and x a vector $x_1, x_2, \ldots, x_n$ subject to $g_i(x) = 0$; $i=1, 2, \ldots, p$, $p < n$.

A solution to this problem is assumed to exist and is denoted by $\bar{x}$, that is $g(\bar{x}) = 0$ and if $g(x) = 0$ then $f(x) \geq f(\bar{x})$. The problem is then transformed into a sequence of unconstrained minimisation problems using a parameter X_k and takes the following form:

$$(A) \quad \underset{x}{\text{minimise}} \; F(x, X_k) = \left[f(x) - X_k\right]^2 + \sum_{i=1}^{p} g_i(x)^2$$

Let x_k denote the solution to the problem. Morrison has proved:

1. if $X_k \leq f(\bar{x})$ then $f(x_k) \leq f(\bar{x})$
2. $f(x_k)$ is a monotonic non-decreasing function of X_k
3. if X^M_{k+1} is defined equal to $X_k + \sqrt{F(x_k)}$ and if $X_k \leq f(\bar{x})$ then $X^M_{k+1} \leq f(\bar{x})$
4. the sequence $\{X^M_j\}$ for $j = k+1, k+2, \ldots \to f(\bar{x})$ as $j \to \infty$ and the sequence $\{X^M_j\}$ approaches the optimal solution from below.

X^M_k is called the Morrison's parameter.

The second method, used by Kowalik (et al.), is the tangent parameter sequence suggested by P. Wolfe in relation to Morrison's article.

If we denote X^T_{k+1} as being the tangent parameter, then we define:

$$X^T_{k+1} = X_k + F(x_k, X_k) / \left\{ F(x_k, X_k) - \sum_{i=1}^{p} g_i(x_k)^2 \right\}^{\frac{1}{2}}$$

The sequence $\{ X^T_k \}$ also approaches the optimum from below under certain conditions but in general at a faster rate. The justification of its use is as follows:

Justification

Consider the (f, g) space (Fig. 1). P is found by minimising $F(x, X_k)$ then from geometric considerations: $X_k + \sqrt{F(x, X_k)}$ is the point on the f axis (o, X^M_k), clearly this is closer to the optimum than X_k.

Wolfe noticed that the tangent at P gives even a better point $(0, X_k)$.

The formula is derived from the fact that the circle has for equation:

$$g^2 + (f - X_k)^2 = F(x_k, X_k)$$

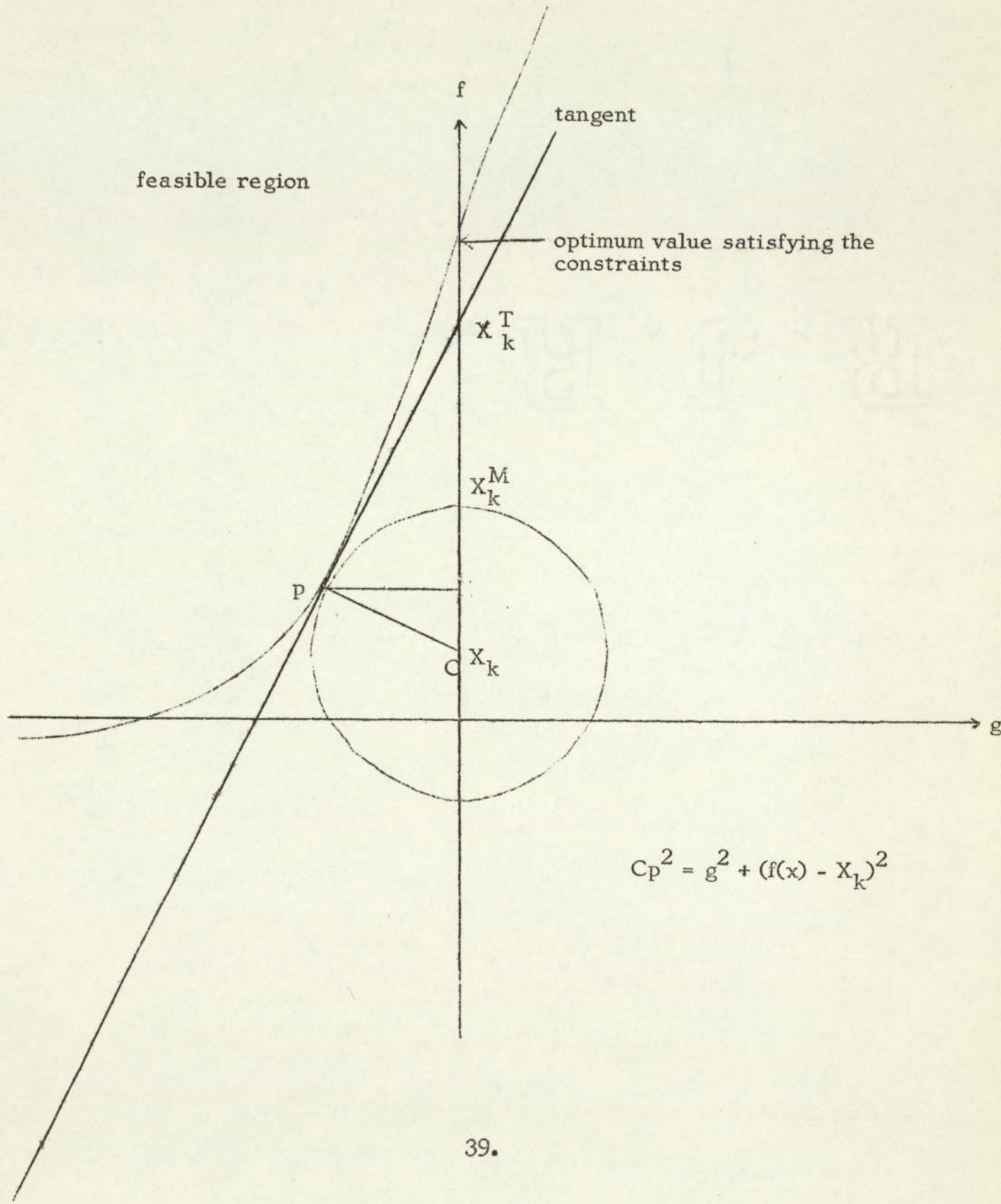
Writing the equation of the tangent and putting g=0 to get the intersection of the tangent with the f axis, we have:

$$F(x_k, X_k) = 0 + (f(x_k) - X_k)(f(x) - X_k)$$

for f we have:

$$f = X_k + \frac{F(x_k, X_k)}{f(x_k) - X_k}$$

which is the required result.


f
tangent
feasible region
optimum value satisfying the constraints
X_k^T
X_k^M
p
C
X_k
g
$Cp^2 = g^2 + (f(x) - X_k)^2$

Now, having justified the use of

$$X^T_{k+1} = X_k + F(x_k, X_k) / \left\{ F(x_k, X_k) - \sum_{i=1}^{p} g_i(x_k)^2 \right\}^{\frac{1}{2}}$$

provided $X_k \leq f(\bar{x}) \Rightarrow X^T_{k+1} \leq f(\bar{x})$ then we can use the parameter sequence $\{X_j\} = \{X^T_j\}$ $j = k+1, k+2, \ldots, \infty$ in (A) and X^T_{k+1} being greater or equal to X^M_{k+1} it should converge to the optimum more rapidly.

We saw earlier on, in Morrison's method, that we require an X_k subject to $X_k \leq f(\bar{x})$ therefore we need to find a lower bound for X_k to initiate the process.

The third method, used by Kowalik (et al.) is Schmit and Fox's method which enables us to bracket the minimum. This method proceeds as follows:

Let $\Gamma_{\bar{x}} = \Gamma \{ x : g(x) = 0$ and let $Y = \max_{x \in \Gamma} f(x)$; then if $f(\bar{x}) \leq X \leq Y$, then X offers an upperbound for $f(\bar{x})$. $f(\bar{x}) \leq X \leq Y$ will hold if $\min_x F(x, X) = 0$. However, if $\min_x F(x, X) \neq 0$ then $X \leq f(\bar{x})$ consequently X offers a lower bound for $f(\bar{x})$. Therefore we examine the value of $F(x, X)$ for the sequence of values $X = \{X_k\}$ $k = 1, 2, \ldots$. When we reach an X_k subject to $F(x, X_k) \neq 0$ then X_k is a lower bound for $f(\bar{x})$ and X_{k-1}, since $\min_x F(x, X_{k-1}) = 0$ must be an upper bound.

Another modification developed by Kelley[10] is used in this method to deal with all types of constraints. This approach converts the inequality constraints into equality constraints by using the Heaviside function H(t) defined as:

$$H(t) = 1 \text{ if } t > 0$$
$$H(t) = 0 \text{ if } t \leq 0$$

Example

Using the Heaviside function, h(x) is transformed into:

$$f(x) = h(x) \quad H\left[-h(x)\right] = 0 \qquad h(x) \neq 0$$

Theorem for Convergence

For $x \in \Gamma$ (Γ closed bounded region) we assume:

1. $f(x)$, $g(x)$ are continuous
2. there exists a solution to the problem
3. $F(x_k) = \min_x F(x, X_k)$ can be found by any methods of unconstrained minimisation and $F(x_k) > 0 \Rightarrow f(x_k) < f(\bar{x})$ then we can say there is convergence and the limit point of the set x_k for $k = 0, 1, \ldots$ is a solution to the problem.

Proof

From assumption 3 we conclude that there exists an upper bound and lower bound for $f(\bar{x})$ and they can be found easily.

There is an increasing sequence of X_j for $j = k+1, k+2, \ldots$ bounded by $f(\bar{x})$. As the sequence X_j converges, therefore X^M_j converges so that:

$$F(x_j) \to 0 \text{ as } j \to \infty$$

giving:

$$\sum_{i=1}^{p} g_i(x_j)^2 \to 0 \text{ as } j \to \infty \quad \text{(I)}$$

$$|f(x_j) - X_j| \to 0 \text{ as } j \to \infty \quad \text{(II)}$$

Let x_L be a limit point of x_j. There is a sequence of points $x_{\sigma j}$ converging to x_L. From assumption 2 and equation (I) we have $g(x_L)=0$; therefore $f(x_L) \geq f(\bar{x})$. We know too that $f(x_{\sigma j}) \leq f(\bar{x})$ and this gives $f(x_L) \leq f(\bar{x})$.

Remarks

The problem might not terminate in a finite nu mber of steps. Keeping an upper bound for $f(\bar{x})$ we ensure that X^T_{k+1} cannot be accepted if $X \geq Y$. This could also give $\min_x F(x, X) > 0$.

This is unacceptable as X would tend towards $+\infty$. In such circumstances we have a stationary point

$$-\nabla f(x) + (\tfrac{1}{2}X)\nabla f(x, 0) = 0$$

so there are two possibilities:

(a) x tends to an unconstrained local minimum of f

(b) f(x) tends to $+\infty$ as X tends to $+\infty$.

The whole algorithm can be divided into two phases:

1. Phase I sets the bounds;
2. Phase II is the actual minimisation process. We can now give a step by step description of the whole algorithm.

Phase I

I Set $x_o \in \Gamma$ (Γ closed bounded region);
Set $X_o = f(x_o)$;
Set k = 1;

II Minimise $F(x, x_k)$ to find x_k;
$X_k = X_{k-1}$ - step;

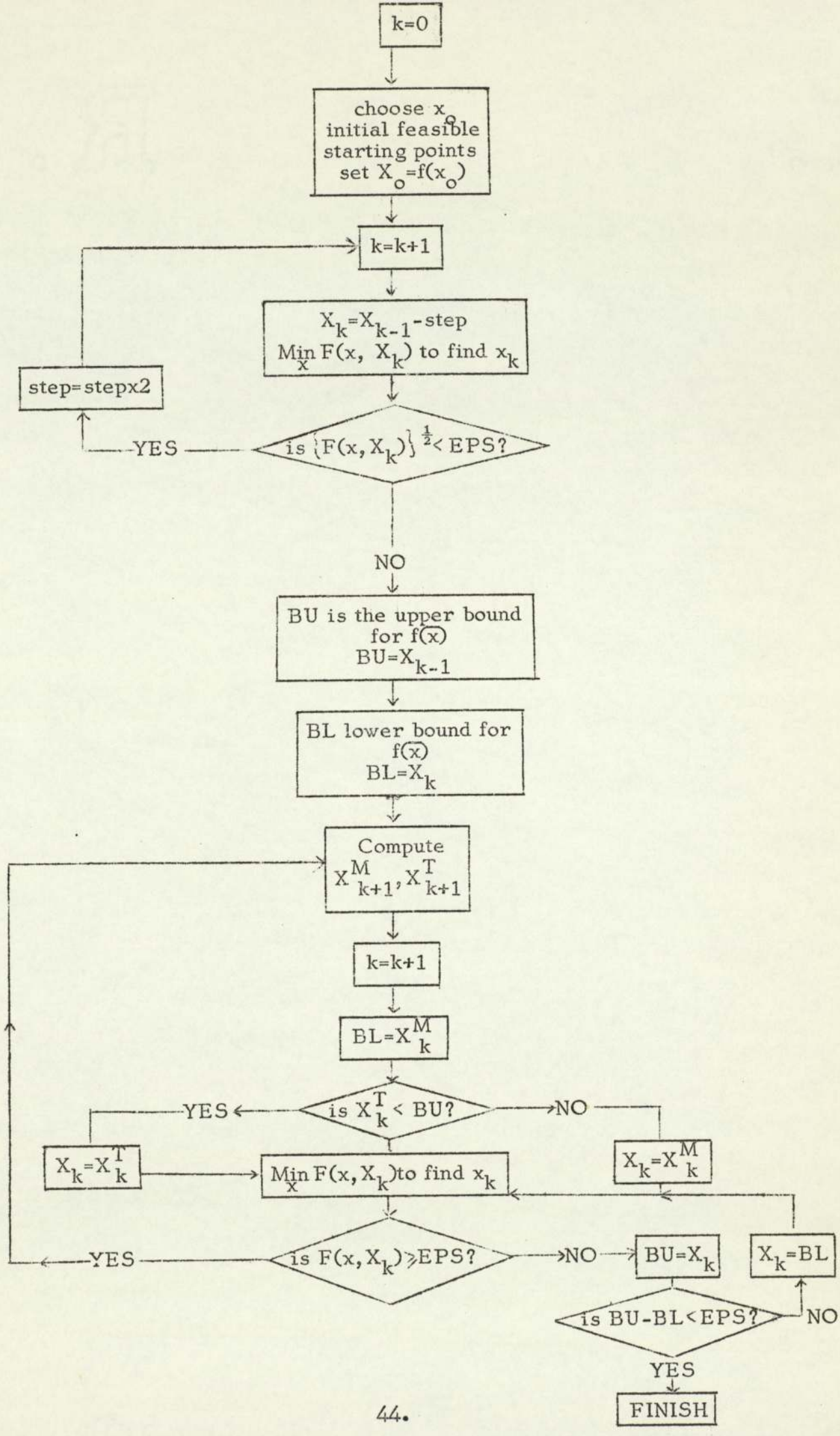
III IF $F(x_k) <$ EPS then set 1) step = 2 * step,
2) k = k+1
goto II else BU = X_{k-1} (BU is an upper bound for $f(\overline{x})$)
goto IV;

Phase II

IV Set BL = X_k (BL is a lower bound for $f(\overline{x})$);
V Compute X^M_{k+1}, X^T_{k+1};
VI Set k = k+1
Set BL = X^M_k
IF X^T_k < BU then $X_k = X^T_k$ else $X_k = X^M_k$;
VII Minimise $F(x, X_k)$ to find x_k;

VIII If $\sqrt{F(x_k)} \geq$ EPS then goto V else BU = X_k;
If BU - BL <EPS then goto FINISH else X_k = BL and goto VII.

FINISH: END OF PROGRAM


k=0
choose x_o initial feasible starting points set $X_o=f(x_o)$
k=k+1
$X_k=X_{k-1}$ - step
$\min_x F(x, X_k)$ to find x_k
step=stepx2
YES
is $\{F(x, X_k)\}^{\frac{1}{2}} < EPS$?
NO
BU is the upper bound for $f(x)$
$BU=X_{k-1}$
BL lower bound for $f(x)$
$BL=X_k$
Compute X^M_{k+1}, X^T_{k+1}
k=k+1
$BL=X^M_k$
YES
is $X^T_k < BU$?
NO
$X_k=X^T_k$
$\min_x F(x, X_k)$ to find x_k
$X_k=X^M_k$
YES
is $F(x, X_k) \geq EPS$?
NO
$BU=X_k$
$X_k=BL$
is BU-BL<EPS?
NO
YES
FINISH

Ultimate Convergence Criterion

In this case the values of the lower bound and the upper bound of f(x) are considered.

If the difference between the two is less than EPS (a small preset positive value) then the program stops.

Comments on the Program

The program has been written in such a way that it could be run with any minimisation sub-routine.

It has been tested using the Powell (64)'s method and the Fletcher and Powell's method.

By using the Fletcher and Powell's method, two external procedures are required:

1. Procedure TEMPCAL (Temp, x)
   where temp is an arry (0:m)
   Temp(0) being the objective function and Temp (1), Temp (2) .... Temp (m) are the constraints. This procedure calculates the value of the constraints and also the sum of square of the constraints set equal to a real number t, and where x is an array (1:n);

2. Procedure TEMPDCAL (Tempd, x)
   where tempd is a matrix (1:m, 1:n).
   This is the matrix of partial derivatives, and where x is an array (1:n).

It is obvious that the procedure TEMPDCAL is not necessary when using the Powell (64)'s method as it does not require the derivatives.

The procedure OPTKOV is a seven-parameter procedure. They are as follows:

X	array (1:n) containing the feasible starting point;
XK	real number calculated inside the program but originally equal to the value of the objective function of the constrained problem;
XKT	real number calculated inside the program representing the value of the tangent parameter;
XKM	real number calculated inside the program representing the value of the Morrison parameter;
STEP	real number by which XK is decreased each time;
F	real number representing the value of the penalty function minimised in the sub-routine;
EPS	representing the stopping criterion, usually EPS is a very small positive value.

The main problem in preparing data for the program is to determine the step by which XK is decreased.

This is difficult to determine not knowing the behaviour of the problem.

A method of trial and error has been used to solve the different problems and it showed that the step could vary between 0.125 and 2. and a wrong choice of the step could give a wrong optimum answer.

If the lower and upper bounds to the problem are known then this program could easily be used feeding in BU and BL and starting the process at Phase II.

3.1.2 Powell's Method for Constrained Problems[17]

This method deals with a general non-linear programming problem of the form: Minimise f(x) where f is a scalar function and x a vector $x_1, x_2, \ldots, x_n$ subject to $g_i(x)=0$; $i=1, 2, \ldots, m$.

First of all we assume the problem has a solution and that the given functions have continuous second derivatives.

The problem is then converted into a sequence of unconstrained problems having the property that the successive solutions of the unconstrained problems converge to the optimal answer.

The method depends on two sets of parameters $(\theta_1 \ldots \theta_m)$, $(\sigma_1 \ldots \sigma_m)$, for which we calculate the vector of variables x, to minimise:

$$\phi(x, \sigma, \theta) = F(x) + \sum_{i=1}^{n} \sigma_i \left[g_i(x) + \theta_i \right]^2$$

Computation experience has shown that the required solution can be obtained for moderate values of the parameters - consequently avoiding difficulties experienced by Fiacco and McCormick method (See. 3.1.3).

The method is based on the following simple theorem.

Theorem I If the values of the variables x which minimise $\phi(x, \sigma, \theta)$ are $\xi(\sigma, \theta)$ then $\xi(\sigma, \theta)$ is a solution to the constrained problem:

minimise f(x)
subject to $g_i = g_i(\xi(\sigma, \theta))$ $i=1, \ldots, m$

Proof If the theorem does not hold and the variable $\xi^*(\sigma,\theta)$ minimises f(x), then $\phi(\xi^*) < \phi(\xi)$ which is a contradiction. This means we just have to obtain the values of the parameters (σ,θ) such that:

$$g_i(\xi(\sigma,\theta))=0 \quad i=1, \ldots, m \qquad (1)$$

so the process is based on an iterative adjustment of the parameters.

For instance if m=1 and σ_1 is fixed, then by adjusting θ_1 we provide a line of points $\xi(\sigma_1, \theta_1)$ in the space of the variables and if this line intersects the surface $g_1(x)=0$ we just have to calculate the value of θ_1.

So we try to satisfy (1) by adjusting θ, σ being fixed.

As the equations are non-linear the adjustment of θ could involve a lot of computations. However, it is found that the following adjustment works well:

$$\theta_i + g_i(\xi) \rightarrow \theta_i \quad i=1, 2, \ldots m$$

Computation experience has verified this works well as long as θ_i is large and Powell[17] has given a theoretical justification for adjustment in this manner.

So in fact this method consists of adjusting the parameters by applying the correction unless it happens that $\max_i |g_i(\xi)|$ fails to converge or converges too slowly to zero when σ is increased.

It is now very easy to write a flow chart for the program, k being the number of iterations, C_k least value of $\max_i |g_i(\xi)|$. At the beginning of the process $c_o = A$ where A is very large positive number exceeding the magnitude of $|g_i(x)|$ all i. If switch is 'down', it means we have just chosen a new value for σ but if switch is 'up' it means the correction has been applied in the previous iteration.

We go on applying the correction as long as we get a convergence: for instance $c_k = \frac{1}{4} c_{k-1}$, otherwise we increase σ.

If σ_i is increased, we adjust θ_i so that the product $\sigma_i \theta_i$ is unchanged.

Let us call ξ^*, σ^*, θ^* the optimal values of ξ, σ, θ then from the condition:

$$\left[\frac{\partial \emptyset x}{\partial x_j}\right]_{x=\xi} = 0 \quad j=1,2,\ldots.n$$

we derive that:

$$\left[\frac{\partial Fx}{\partial x_j} + 2\sum_{i=1}^{m} \sigma_i^* \theta_i^* \frac{\partial g_i(x)}{\partial x_j}\right]_{x=\xi^*} = 0$$

Therefore the final gradient vector of F(x) is a linear combination of the final gradient vectors of the functions $g_i(x)$ and the appropriate linear factors are $-2\sigma_i^* \theta_i^*$ i=1,2 .... m. From there we derive another theorem.

Theorem II If our problem has a unique solution, and at this solution the gradient vectors of the functions $g_i(x)$ are linearly indepen dent, then for i=1,2, ... m, the final value of $\sigma_i \theta_i$ is independent of the parameters.

The algorithm is described step by step and schematised in the following flow diagram.

Flow Chart

I Set $k = 0$;
Set $\bar{\sigma}_i = 1$;
Set $\theta_i = 0$;
Set $c_o = A$;
Switch down.

II $k = k+1$;
Calculate ξ to minimise $\phi(x)$;
$c_k = \max_i |g_i(\xi)|$.

III IF c_k is small enough then goto EXIT else goto IV.

IV IF $c_k \geq c_{k-1}$ then goto V else goto VI.

V Set $c_k = c_{k-1}$.
If switch is up goto VII else goto VIII.

VI If switch is up then goto IX else goto X.

VII Set $\theta_i = \bar{\theta}_i$ goto XIII.

VIII Goto XIII.

IX Goto XII

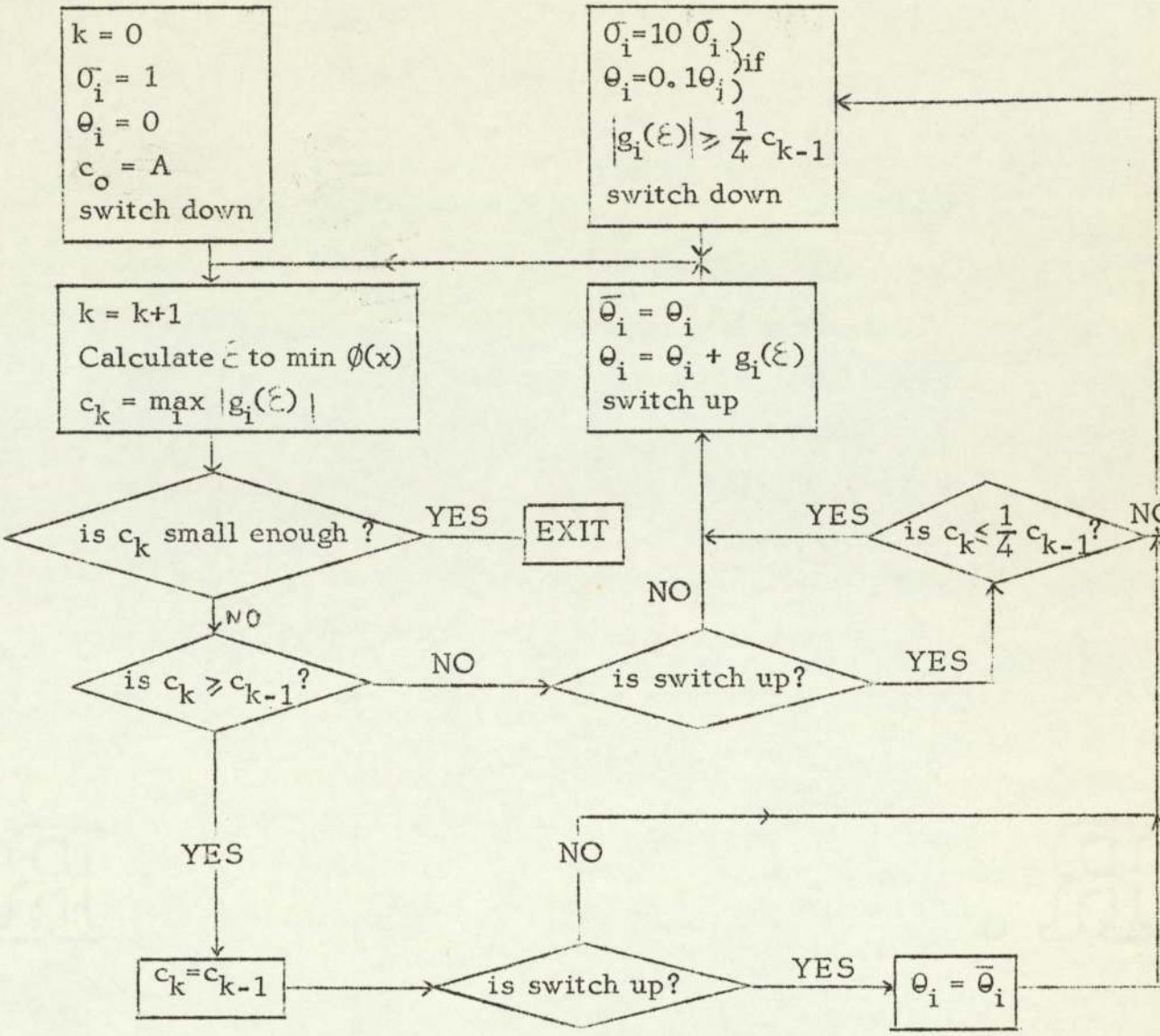
X If $c_k \leq \frac{1}{4} c_{k-1}$ then goto XI else goto XIII.

XI Goto XII.

XII $\bar{\theta}_i = \theta_i$ $\theta_i = \theta_i + g_i(\xi)$ and set switch up. Goto II.

XIII If $|g_i(\xi)| \geq \frac{1}{4} c_{k-1}$ then $\bar{\sigma}_i = 10\,\bar{\sigma}_i$ and $\theta_i = 0.1\,\theta_i$. Set switch down goto II.

EXIT: END OF PROGRAM.


k = 0
$\sigma_i = 1$
$\theta_i = 0$
$c_o = A$
switch down
$\sigma_i = 10\,\sigma_i$)
$\theta_i = 0.1\theta_i$) if
$|g_i(\xi)| \geq \frac{1}{4} c_{k-1}$
switch down
k = k+1
Calculate ξ to min $\phi(x)$
$c_k = \max_i |g_i(\xi)|$
$\bar{\theta}_i = \theta_i$
$\theta_i = \theta_i + g_i(\xi)$
switch up
is c_k small enough ?
YES
EXIT
NO
is $c_k \geq c_{k-1}$?
NO
is switch up?
YES
YES
is $c_k \leq \frac{1}{4} c_{k-1}$?
NO
YES
NO
$c_k = c_{k-1}$
is switch up?
YES
$\theta_i = \bar{\theta}_i$

Ultimate Convergence Criterion

From the flow diagram we notice that a variable c is used. To start with this variable is assigned a value A, where A is a large positive number exceeding the magnitude of $\phi_i(x)$ for i=1,2, ..., m. At each step, c decreases as it is assigned the value of $\max_i \phi_i(\xi)$ and whenever the value of c reaches a certain value ϵ (a preset small positive value) the process is terminated.

Comments on the Program

Similar to Kowalik's method, Powell's method has been written for problems with equality constraints. If problems with inequality constraints have to be dealt with, the Heaviside function H(t) is used (see Kowalik's method).

The program has been written in such a way that it could be run with any minimisation sub-routine.

It has been tested using the Powell (64)'s method and the Fletcher and Powell's method.

By using the Fletcher and Powell's method, two external procedures are required:

1. Procedure PSICAL (psi, x)
   where psi is an array (1:m)
   psi (1), psi (2), ..., psi (m) are the constraints; and where x is an array (1:n). This procedure calculates the value of the constraints.

2. Procedure PSIDCAL (psid, x)
   where psid is a matrix (1:m, 1:n). This is the matrix of partial derivatives, and where x is an array (1:n).

It is obvious that the procedure PSIDCAL is not necessary when using the Powell (64)'s method as it does not require the derivatives.

The use of the Powcon. procedure does not give any problem. It is a four-parameter procedure:

X array of starting point;

EPS small preset positive value determining the stopping criterion;

RATIO integer number (the value suggested by Powell is 4 and used for testing the convergence: the correction is applied as long as $c_k \leq \frac{1}{\text{ratio}} c_{k-1}$);

M integer number equal to the number of constraints;

One point needs to be clarified however: a scaling problem arises when choosing the initial values for σ_i and θ_i. $\sigma_i=1$ and $\theta_i=0$ is a good initial choice.

3.1.3 S.U.M.T.

Sequential Unconstrained Minimisation Technique (SUMT) developed by Caroll, Fiacco and McCormick.[3,4,5,6,7,21]

Originally the problem is as follows: consider a general non-linear problem:

minimise $f(x)$

subject to $g_i(x) \geq 0 \quad i=1, \ldots, m \quad x=(x_1, x_2, \ldots, x_n)$

On applying SUMT this ordinary constrained problem is reduced to a sequence of unconstrained problems of the following form.

1. Minimise $P(x, r_k) = f(x) + r_k \sum_{i=1}^{m} \frac{1}{g_i}(x)$ where $r_k > 0$ and $\frac{1}{g_i}(x)$ defined only if $g_i(x) > 0$. $r_k > r_{k+1}$ $r_k \to 0$ as $k \to \infty$. In order to prove the convergence of the system, ie. that $x(r_k) \to \hat{x}$ and $P(x(r_k), r_k) \to \hat{f}$ we need to set up a certain number of conditions usually attached to non-linear programming.

I P, f are convex;

II $g_i(x)$ are concave;

III $R^o = \{ x \mid g_i(x) > 0,\ i=1, 2, \ldots, m \}$ is non empty;

IV the functions $f, g_1, \ldots, g_m$ are twice continuously differentiable;

V for every finite k, $\{ x \mid f(x) \leqslant k;\ x \in R \}$ is a bounded set, where $R = \{ x \mid g_i(x) \geqslant 0,\ i=1, \ldots, m \}$;

VI the function $P(x, r) = f(x) + r \sum_i 1/g_i(x)$ is for each $r > 0$, strictly convex for $x \in R^o$.

Proof of the Convergence

If the conditions 1-6 are satisfied, then:

(i) each function $P(x, r_k)$ is minimised over R^o at a unique $x(r_k) \in R^o$ where $\nabla x P \left[x(r_k) r_k \right] = 0$.

(ii) $\mathrm{Lim}_{k \to \infty} P \left[x(r_k), r_k \right] = \mathrm{Min}_{x \in R} f(x) = v^o$

Proof for (i)

Let us call x_o the starting value vector $M_o = P(x_o, r_k)$. We now form two sets:

(a) $S_o = \{ x \mid f(x) \leqslant M_o,\ x \in R \}$;

(b) $S_i = \{ x \mid r_k / g_i(x) \leqslant M_o - v_o \}$ $\quad i=1, \ldots, m$

(c) $S = \bigcap_{i=0}^{m} S_i$.

From this it follows that:

$$\inf_{x \in S} P(x, r_k) = \inf_{x \in R^o} P(x, r_k) \geqslant v_o > -\infty$$

But S is non-empty, contains no boundary of R so from condition IV and the construction of $S_1, \ldots, S_m$, $P(x, r_k)$ is continuous in S.

$\wedge$ denotes optimal value.

Since the greatest lower bound of a continuous function, bounded on a compact set is taken on by a point in that set, then at least one $x(r_k)$ exists.

As $P(x, r_k)$ is strictly convex in R_o, there exists only one $x(r_k)$ and also there exists only one local minima to P in R^o.

Proof for (ii)

Let $\varepsilon > 0$ be any positive number. Then select an x^* such as $x^* \in R^o$ and $f(x^*) < v_o + \varepsilon/2$. Select $k^* < \left[\min_i g_i(x^*)\right] \varepsilon/2m$. Then for $m = m^*$:

$$v_o \leq \inf_{x \in R^o} P(x, r_k)$$

$$v_o = P\left[x(r_k), r_k\right] \leq P\left[x(r_k^*), r_k\right] < P\left[x(r_k^*), r_k^*\right]$$

$$v_o \leq P\left[x^*, r_k^*\right] < v_o + \varepsilon/2 + \varepsilon/2 = v_o + \varepsilon$$

This technique can deal only with inequality constraints.

In 1965 the technique was extended to deal simultaneously with inequality and equality constraints.

The ordinary non-linear problem is then:

$$\begin{aligned} &\text{minimise } f(x) \\ &\text{subject to } g_i(x) \geq 0 \quad i=1, \ldots, m \\ &\qquad\qquad h_j(x) = 0 \quad j=1, \ldots, p \end{aligned}$$

and the sequence of unconstrained problems becomes:

$$\text{minimise } P(x, r_k) = f(x) + r_k \sum_{i=1}^{m} 1/g_i(x) + \sum_{j=1}^{p} h^2_j(x)/^{\frac{1}{2}} r_k$$

Primal Problem with Inequality Constraints

Using SUMT, the primal of a problem becomes:

$$\text{Minimise } P(x, r_k) = f(x) + r_k \sum_{i=1}^{m} 1/g_i(x)$$

In order to be able to solve such a problem we need to formulate the dual of this problem.

Dual Problem with Inequality Constraints

The dual may be written as follows:

$$(1) \quad \text{Maximise } \left\{ G(x, u) = f(x) \sum_{i=1}^{m} u_i\, g_i(x) \;\middle|\; \nabla_x G(x, u) = 0 \quad u \geq 0 \right\}$$

or:

$$\text{Maximise } \left\{ f(x) - \sum_{i=1}^{m} u_i\, g_i(x) \;\middle|\; \frac{\partial f}{\partial x} + \sum_{i=1}^{m} u_i \frac{\partial g_i}{\partial x} = 0 \quad u \geq 0 \right\}$$

Any (x, u) which satisfies $\nabla_x G(x, u) = 0$ with $u \geq 0$ is a feasible solution. Expanding $\nabla_x G(x, u)$ we have:

$$\nabla_x f(x) - \sum_{i=1}^{m} u_i\, \nabla_x g_i(x) = 0$$

$$\nabla_x f(x) = \sum_{i=1}^{m} u_i\, \nabla_x g_i(x)$$

A sufficient condition that x be a solution to the penalty problem is that:

$$\nabla_x f(x) - r \sum_{i=1}^{m} \frac{\nabla_x g_i(x)}{g_i^2(x)} = 0$$

$$\nabla_x f(x) = r \sum_{i=1}^{m} \frac{\nabla_x g_i(x)}{g_i^2(x)} \qquad u_i \geq 0$$

Hence:

$$\nabla_x f(x) = \sum_{i=1}^{m} u_i \nabla_x g_i(x) = r \sum_{i=1}^{m} \frac{\nabla_x g_i(x)}{g_i^2(x)}$$

and $u_i = \dfrac{r}{g_i^2(x)}$ which satisfies $u_i \geq 0$ as $r > 0$ and $g_i^2 > 0$.

Thus if $x(r_k)$ is the solution vector to $P(x, r_k)$ then $x(r_k)$, $u(r_k)$ will be a feasible solution of the dual problem (1).

The corresponding value of the dual objective function would be:

$$G(x(r_k), u(r_k)) = f(x(r_k)) - r_k \sum_{i=1}^{m} \frac{1}{g_i(x(r_k))}$$

An important property of dual programming is:

$$f(x) \geq f(\hat{x}) \geq G(x, u)$$

but:

$$P(x, r) \geq f(x) \text{ as } P(x, r) = f(x) + r \sum_{i=1}^{m} \frac{1}{g_i(x)}$$

$$\text{where } r \sum_{i=1}^{m} \frac{1}{g_i(x)} \geq 0$$

Therefore:

$$P(x, r) \geq f(\hat{x}) \geq G(x, u)$$

Using the fact that the P-value and G-value constitute respectively the upper and lower bound to the optimal value $\hat{f}$, and that they would converge to it from opposite directions, we can set a stopping criterion for the termination of the process.

This could be that the difference between $P(x, r)$ and $G(x, u)$ must be smaller or equal to ε (ε being any small value we care to choose).

Primal Problem Including Equality and Inequality Constraints

The problem is then as follows:

$$\begin{aligned} &\text{Minimise } f(x) \\ &\text{subject to } g_i(x) \geq 0 \text{ for } i=1, \ldots, m \\ &\qquad\qquad h_j(x) = 0 \text{ for } j=1, \ldots, p \end{aligned}$$

and the penalty function becomes:

$$P(x, r) = f(x) + r \sum_{i=1}^{m} 1/g_i(x) + \sum_{j=1}^{p} h^2 j(x)/r^{\frac{1}{2}}$$

In order to avoid difficulties we can have when we have local minima, we require that the points satisfying the constraints of the problem should form a convex set. Similarly as for the first problem, there are conditions to be attached to this one.

I The function $f(x)$ is convex;

$\sum_{j=1}^{p} h^2 j(x)$ is convex in R;

II The functions $g_1, \ldots, g_m$ are concave;

III If $Q = \{x \mid h_j(x) = 0, j=1, \ldots, p\}$ and
$R^o = \{x \mid g_i(x) \quad 0, i=1, \ldots, m\} \Rightarrow R^o \cap Q$ non-empty;

IV The functions $f, g_1, \ldots, g_m, h_1, \ldots, h_p$ are continuous;

V For every finite k, and every $r > 0$,

$\{x \mid f(x) + \sum_{j=1}^{p} h^2 j(x)/r^{\frac{1}{2}} \leq k, x \in R\}$ is a bounded set where R is the closure of R^o.

If the equality constraints are linear then condition V is reduced to $\{x \mid f(x) \leq k;\ x \in R \cap Q$ From conditions 3, 4 and 5 we derive that there exists a finite number v_o where $v_o = \inf x \in R \cap Q = \min x \in R \cap Q$ From conditions 1 and 2 we derive that P is convex in R^o

VI The function $P(x, r) = f(x) + r \sum_{i=1}^{m} 1/g_i(x) + \sum_{j=1}^{p} h^2 j(x)/r^{\frac{1}{2}}$ is,

for each $r > 0$, strictly convex for $x \in R^o$.

Proof of Primal Convergence

If all these conditions are satisfied then:

(a) each function $P(x, r_k)$ is minimised at a unique $x(r_k) \in R^o$;

(b) $\text{Lim}_{k \to \infty} P\left[x(r_k), r_k\right] = \min_{R \cap Q} f(x) = v_o$

(c) The unique limit point x^* of the uniformly bounded sequence $\{x(r_k)\}$ is a solution to the primal problem.

Proof for (a)

$x_o \in R^o$ and $M^o = P(x_o, r_k)$.

Form the set $S_o = \left\{x \mid f(x) + \sum h_j^2(x)/r_k^{\frac{1}{2}} \leq M_o;\ x \in R\right.$

$v_k = \inf_{S_o} \left[f(x) + \sum h_j^2(x)/r_k^{\frac{1}{2}}\right] \left[v_k < M_o \text{ as } r_k \sum 1/g_i(x_o) > 0\right]$

Now form the set

$S_1 = \left\{x \mid r_k/g_i(x) \leq M_o - v_k\right\} \quad i=1, \ldots, m$

$S = \bigcap_{i=0}^{m} S_i$

From this we derive:

$$\inf_S P(x, r_k) = \inf_{R^o} P(x, r_k) > -\infty$$

Since $P(x, r_k)$ is continuous in S, it takes in greatest lower bound in S. As $P(x, r_k)$ is strictly convex in R^o we can say that the minimising point $x(r_k)$ is unique in S and also that no other local minimum exists to P in R^o.

Proof for (b) and (c)

Let ε be any positive number and select x^* such that $x^* \in R^o$, $x^* \in Q$, $f(x^*) < v_o + \varepsilon/2$ and k^* such as $r_k^* < \min_i g_i(x^*)[\varepsilon/2m]$ Then for $k > k^*$:

$$\underline{A}: \inf_{R^o} P(x, r_k) = P\left[x(r_k), r_k\right] \leq P(x^*, r_k) \leq P(x^*, r_k^*) <$$

$$v_o + \varepsilon/2 + \varepsilon/2 = v_o + \varepsilon$$

Now suppose we have an integer k greater than k^* and we define T

$$T_k = \left\{ x \mid f(x) + \sum h_j^2(x)/r_k^{\frac{1}{2}} \leq v_o + \varepsilon ; \quad x \in R \right\}$$

$$T_L = \left\{ x \mid f(x) + \sum h_j^2(x)/r_L^{\frac{1}{2}} \leq v_o + \varepsilon ; \quad x \in R \right.$$

where $L > k$. By condition 5, T_k and T_L are bounded sets. Also, since $r_L < r_k$, $T_L \subset T_k$. T_L is not empty since $x\left[r_L\right]$ is contained in it. So:

$$f\left[x(r_L)\right] \geq \min_{x \in T_L} f(x) \geq \min_{x \in T_k} f(x) > -\infty$$

and $\left\{x(r_L)\right\}$ is uniformly bounded. It has a unique limit point since $P\left[x, r_k\right]$ is strictly convex. Then we can rewrite equation A. For any $\varepsilon > 0$, there is a $k^*(\varepsilon)$ such that for $L > k^*(\varepsilon)$.

$$\underline{B}: \quad v_o + \varepsilon > f\left[x(r_L)\right] + r_L \sum 1/g_i \left[x(r_L)\right] + \sum h^2 j \left[x(r_L)\right] / r^{\frac{1}{2}}$$

Because the three terms of the right hand side of the inequality is bounded below it is possible to show:

1. each term is bounded above;

2. $\text{Lim}_{L \to \infty} h_j\left[x(r_L)\right] = 0$ for $j=1, \ldots, p$ and consequently the limit point x of the sequence $\left\{x(r_L)\right\}$ is primal feasible ($x \in R \cap Q$).

If either $r_L \sum 1/g_i\left[x(r_L)\right]$ or $\sum h_j^2\left[x(r_L)\right] / r_L^{\frac{1}{2}}$ has a limiting value greater than 0 then $f(x) = \lim_{L \to \infty} f\left[x(r_k)\right]$ must have values less than v_o (from B). This contradicts the fact that v_o is the smallest value that any primal feasible point can take on. Therefore:

3. $$\text{Lim}_{L \to \infty} r_L \sum 1/g_i\left[x(r_L)\right] = 0,$$

$$\text{Lim}_{L \to \infty} \sum h_j^2\left[x(r_L)\right] / r_L^{\frac{1}{2}} = 0$$

$$\text{Lim}_{L \to \infty} f\left[x(r_L)\right] = v_o$$

4. $$\text{Lim}_{L \to \infty} P\left[x(r_L), r_L\right] = v_o.$$

From this point we assume that the problem functions are differentiable and the condition IV becomes

The functions $f, g_1, \ldots, g_m, h_1, \ldots, h_p$, are twice continuously differentiable.

Since a necessary condition that a point be a local minimum of an unconstrained function is that the first partial derivatives vanish there, we can say from condition IV and from the proof of convergence:

$$\underline{C}: \; V_x P\left[x(r_k), r_k\right] = 0$$

Dual of the Problem

The dual is formulated in different ways depending upon the nature of the equality constraints, if they are linear or not. If they are non-linear then we rewrite the primal problem in the following form:

minimise $f(x)$

subject to $g_i(x) \geq 0 \quad i=1, \ldots, m$

$$-h^2 f(x) \geq 0 \quad j=1, \ldots, p$$

Then the dual is:

$$\max G(x,u,w) = f(x) - \sum_{i=1}^{m} u_i\, g_i(x) + \sum_{j=1}^{p} w_j\, h_j^2(x)$$

subject to $\nabla x\, G(x,u,w) = 0$

$$u_i \geq 0 \quad i=1, \ldots, m$$
$$w_j \geq 0 \quad j=1, \ldots, p$$

If h_j's are non-linear, we do not know if the solution of that dual coincides with v_o, the minimum value of the corresponding primal.

However, it can be proved that the method generates points which are dual feasible and whose G values bound v_o from below.

Let $u_i(r_k) = r_k / g_i^2\left[x(r_k)\right]$ for $i=1, \ldots, m$ and let $w_j(r_k) = r_k^{-\frac{1}{2}}$ for all j. Then from equation C it follows that

$$\nabla_x G\left[x(r_k),\, u(r_k),\, w(r_k)\right] = 0$$

Let x^* be a point in $R \cap Q$ where $f(x^*) = v_o$ and let $\Delta x = x^* - x(r_k)$. Then:

$$v_o = f(x^*) + r_k^{-\frac{1}{2}} \sum h_j^2(x^*) \geq$$

$$f\left[x(r_k)\right] + r_k^{-\frac{1}{2}} \sum h_j^2\left[x(r_k)\right] +$$

$$\left\{ \nabla_x f\left[x(r_k)\right] + r_k^{-\frac{1}{2}} \sum 2h_j\left[x(r_k)\right] \nabla_x h_j\left[x(r_k)\right] \right\}^T \Delta_x$$

by convexity of f and h_j^2 condition I

$$= f\left[x(r_k)\right] + r_k^{-\frac{1}{2}} \sum h_j^2 \left[x(r_k)\right] +$$

$$r_k \sum g_i^{-2} \left[x(r_k)\right] \nabla_x^T g_i \left[x(r_k)\right] \Delta_x \text{ from relation (C)}$$

$$\geq f\left[x(r_k)\right] + r_k^{-\frac{1}{2}} \sum h_j^2 \left[x(r_k)\right] +$$

$$r_k \sum g_i^{-2} \left[x(r_k)\right] \left\{\left[- g_i \left[x(r_k)\right] + g_i(x^*)\right]\right\}$$

by concavity of g_i condition II

$$\geq f\left[x(r_k)\right] + r_k^{-\frac{1}{2}} \sum h_j^2 \left[x(r_k)\right] - r_k \sum 1/g_i \left[x(r_k)\right]$$

(since $g_i(x^*) \geq 0$ as $x^* \in R \cap Q$) $= G\ x(r_k),\ u(r_k),\ w(r_k)$

From now on it will be assumed that the h_j's are linear therefore condition I (in primal problem) is being replaced by the following condition: The function $f(x)$ is convex and the functions $h_j(x)$ $j=1, \ldots, p$ are linear. If this is the case then we can write the primal problem:

$$\begin{aligned} &\text{Minimise } f(x) \\ &\text{subject to } g_i(x) \geq 0 \quad i=1, \ldots, m \\ &\qquad\qquad h_j(x) \geq 0 \quad j=1, \ldots, p \\ &\qquad\qquad -h_j(x) \geq 0 \quad j=1, \ldots, p \end{aligned}$$

Wolfe's theorem proves that if the primal has an optimal solution at a point x^* then the dual problem:

$$\text{Max } G(x,u,v,w) = f(x) + \sum u_i g_i(x) + \sum w_j h_j(x) - \sum v_j h_j(x)$$

$$\text{subject to } \nabla x\ G(x,u,v,w) = 0$$

$$u \geq 0$$
$$v \geq 0$$
$$w \geq 0$$

has a solution at some (x^*, u^*, v^*, w^*) where $G(x^*, u^*, v^*, w^*) = v_o$

Initial Value of R

Two alternative methods of calculating r are suggested by Fiacco and McCormick in.[4]

As they stand they are applicable either when there are no equality constraints or when x^o(initial starting point) satisfies the equality constraints.

(a) $$r_1 = - \nabla f(x^o)^T \nabla p(x^o) / |\nabla p(x^o)|^2$$

$$\text{where } p(x) = \sum_{i=1}^{m} 1/g_i(x)$$

This comes from the condition that P is minimised when the first partial derivatives vanishe so r_1 should be chosen in such a way that it minimises the magnitude of the square gradient of P at x_o. Note that as r_1 must be > 0 then:

$$\nabla f(x^o)^T \nabla p < 0$$

(b) Let us call H_1 the Hessian matrix of f(x) and H_2 the Hessian matrix of p(x), both calculated at x^o. Then the magnitude of the gradient gives an estimate of the amount by which P(x, r) exceeds its minimum value which is:

$$\underline{D}: \nabla p(x^o, r^o)^T \left[H_1 + rH_2\right]^{-1} \nabla P(x^o, r)/2$$

If x_o is near several constraint boundaries, H_1 can be elminated then the value of r for which (D) is minimised is:

$$r = \left(\frac{\nabla f(x^o)^T H_2^{-1} \nabla f(x^o)}{\nabla p(x^o)^T H_2^{-1} \nabla p(x^o)} \right)^{\frac{1}{2}}$$

Approximation and Extrapolation

Experiments showed that the trajectory of x(r) is approximately linear in $r^{\frac{1}{2}}$ as r approaches 0. That is to say, for a small r:

$$x(r) = \bar{x} + ar^{\frac{1}{2}}$$

and

$$x(r/c) = \bar{x} + a(r/c)^{\frac{1}{2}}$$

$\bar{x}$ being a feasible point considered as an estimate of the solution, a being some constant, c being the constant by which r is divided at each iteration.

The first order estimate of the solution is given by eliminating a from the two previous equations:

$$\bar{x} = (c^{\frac{1}{2}}\, x(r/c) - x(r))/(c^{\frac{1}{2}}-1)$$

Then the first order estimate of points for which $P(x, r/c^2)$ is a minimum is obtained by assuming:

$$x(r/c^2) = \bar{x} + a(r/c^2)^{\frac{1}{2}}$$

and we get:

$$x(r/c^2) = x(r/c) + 1/c^{\frac{1}{2}}\left[x(r/c) - x(r)\right]$$

In practice the function $P(x, r/c^2)$ is minimised along the vector connecting the last two minima. This has substantially reduced the effort required to minimise the P function.

Updating the Value of r

Two observations are made with respect to the manner of reducing r after each P minimisation:

1. It is highly advantageous to change r by a constant effort;

2. The overall effort required to obtain a solution is relatively insensitive to the choice of factor, over a wide range of values of this factor.

The value of r, suggested by Fiacco and McCormick, for the $(i+1)^{st}$ minimisation is given by $r_{i+1} = r_i/c$ where $c > 1$.

The whole algorithm, step by step, can be described as follows:

I Select a feasible starting point x^o;

II Calculate a suitable initial value of r;

III Form the function

$$P(x, r_k) = f(x) + r_k \sum_{i=1}^{m} 1/c_i(x) + r_k^{-\frac{1}{2}} \sum_{j=1}^{p} h^2_j$$

IV Find the unconstrained minimum of $P(x, r_1)$ in:

$$W_o = \{x/c_i(x) > 0 \text{ for } i=1, 2, \ldots, p\}$$

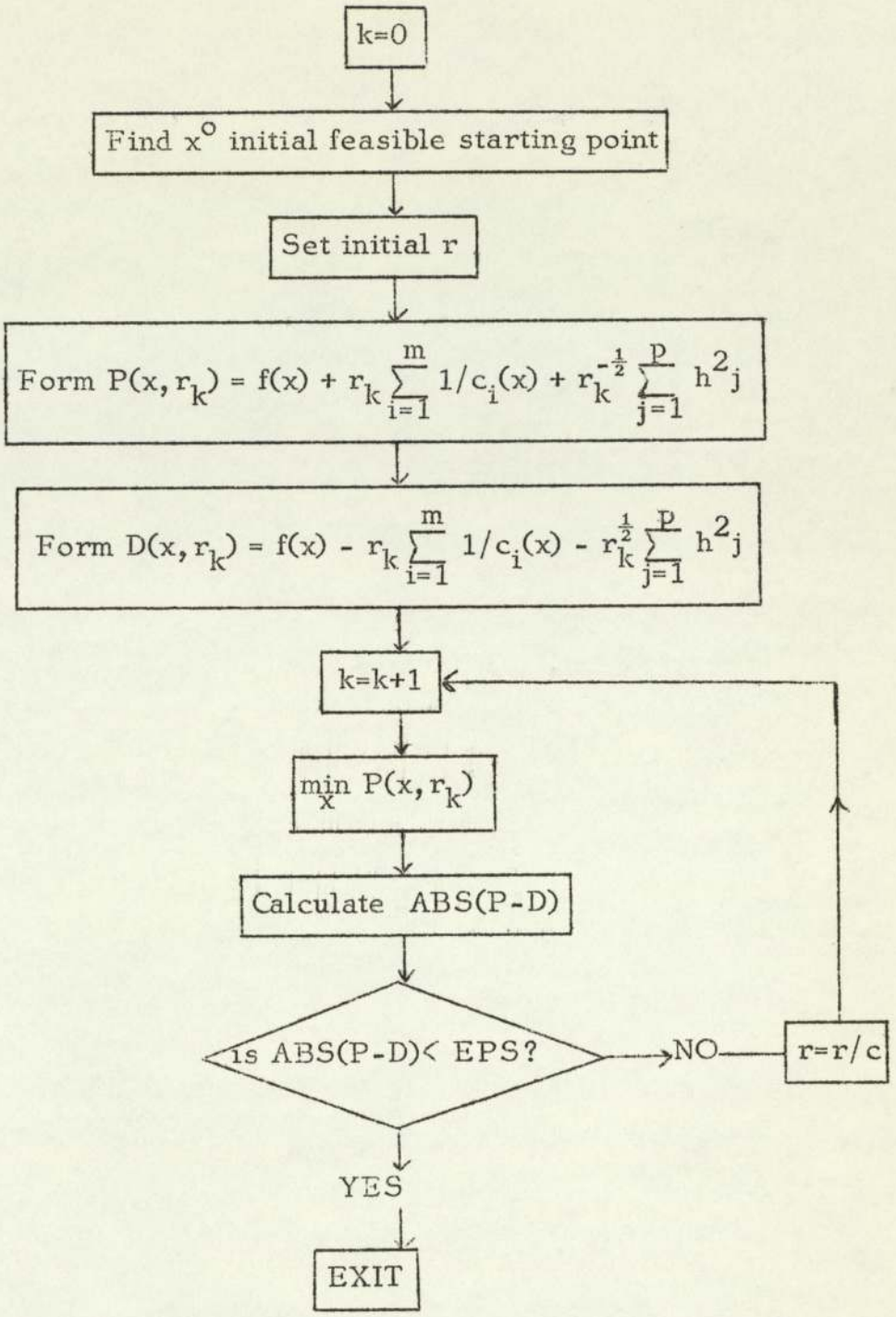
The starting point x^o has to be quasi-feasible ie. $x^o \in W_o$;

V Starting from the minimum of the unconstrained function $P(x, r_1)$ which is called $x(r_1)$ minimise $P(x, r_2)$ where $r_1 > r_2 > 0$;

VI Continue to minimise $P(x, r_k)$ for a monotonically decreasing sequence of values r_k where:

$$\lim r_k = 0 \text{ when } k \to \infty$$

This algorithm can be schematised in the following flow diagram.



The flow diagram might look very simple but in fact difficulties arise when minimising $P(x, r_k)$ as the constraints have to be satisfied at each step.

Full comments will be given in a following paragraph called 'Comments on the Program'.

Ultimate Convergence Criterion

In the description of the algorithm we proved that the primal and the dual of the auxiliary function converge towards the optimal value from opposite directions therefore the most sensible stopping criterion is to evaluate the difference between the Primal and the Dual. Whenever the difference is smaller than ε (ε being any small positive value we care to choose) the program stops.

Comments on the Program

The procedure SUMT has three parameters: X [1:n.] is the array of feasible starting point; EPS is a small positive value (if the difference between the primal and the dual of the function is less than EPS the program stops); RATIO is an integer number defining the value by which r is reduced after each iteration.

The use of the program is simple once the following modification has been included.

After each minimisation of $P(x, r_k)$ we need to check whether or not the constraints are still satisfied. If they are not, this means the step by which x has been decreased in the minimisation procedure is too large, therefore we divide the step by a certain quantity until all the constraints are satisfied. This is applicable in the case of inequality constraints only.

SUMT has been tested with the Fletcher and Powell's method only, and the modification introduced in this method for the constraints to be satisfied at each iteration is shown in Appendix II. Also within the procedure FUNET in Flepomin we calculate not only the penalty function but also its dual $D=f(x) = r \sum_{i=1}^{m} 1/g_i$.

By looking at the structure of $P(x, r_k)$ one could think that difficulties (ie. overflow) might arise after a few iterations when r becomes very small. But this never happened in practical experiences.

3.2 Computational Results

The three programs, OPTKOV, POWCON and SUMT have been tested with the same functions and same convergence criterion so the comparison of the results could be fair.

The functions used are:

Function 1 suggested by Fiacco and McCormick

$$\text{Minimise } f(x) = (x_1 + 1)^3/3 + x_2$$

$$\text{subject to } x_1 - 1 \geq 0$$

$$x_2 \geq 0$$

This function has a minimum at $f = \frac{8}{3}$ and $x = (1, 0)$. The starting point used is $x^0 = (1.125, 0.125)$.

Function 2 Rosen-Suzuki's problem

$$\text{Minimise } f = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4$$

$$\begin{aligned} \text{subject to } & -x_1^2 - x_2^2 - x_3^2 - x_4^2 - x_1 + x_2 - x_3 + x_4 + 8 \geq 0 \\ & - x_1^2 - 2x_2^2 - x_3^2 - 2x_4^2 + x_1 + x_4 + 10 \geq 0 \\ & - 2x_1^2 - x_2^2 - x_3^2 - 2x_1 + x_2 + x_4 + 5 \geq 0 \end{aligned}$$

This function has a minimum at $f = -44$ and $x = (0, 1, 2, -1)$. The starting point used is $x^0 = (0, 0, 0, 0)$.

Function 3 Beale's problem

$$\text{Minimise } f = 9 - 8x_1 - 6x_2 - 4x_3 + 2x_1^2 + 2x_2^2 + x_3^2 + 2x_1x_2 + 2x_1x_3$$

$$\begin{aligned} \text{subject to } & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_3 \geq 0 \\ & - x_1 - x_2 - 2x_3 + 3 \geq 0 \end{aligned}$$

This function has a minimum at $f = \frac{1}{9}$ and $x = (\frac{4}{3}, \frac{7}{9}, \frac{4}{9})$. The starting point used is $x=(0.5, 0.5, 0.5)$.

Function 4 is a problem with equality constraints suggested by Powell[17]

$$\text{Minimise } f(x) = x_1 x_2 x_3 x_4 x_5$$
$$\text{subject to } x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0$$
$$x_2 - x_3 - 5x_4x_5 = 0$$
$$x_1^3 + x_2^3 + 1 = 0$$

This function has a minimum at $f = 2.9197$ and $x = (-1.1712, 1.5957, 1.8272, -0.7636, -0.7636)$. The starting point used is $x = (-2, 1.5, 2, -1, -1)$.

Function 5 This problem and the two following problems have been suggested by Dr. K.P. Wong[21]

$$\text{Minimise } f(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

$$\text{subject to } -2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 + 127 \geq 0$$
$$-7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 + 282 \geq 0$$
$$-23x_1 - x_2^2 - 6x_6^2 + 8x_7 + 196 \geq 0$$
$$-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0$$

This function has a minimum at $f = 680.97$ and $x = (2.30, 195, -0.47, 4.37, 0.51, 1.03, 1.58)$. The starting point used is $x = (1, 2, 0, 4, 0, 1, 1)$.

Function 6

$$\text{Minimise } f(x) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$$

$$\text{subject to } -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0$$

$$-5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 4 \geq 0$$

$$-\tfrac{1}{2}(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0$$

$$-x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0$$

$$-4x_1 - 5x_2 + 3x_7 - 9x_8 + 105 \geq 0$$

$$-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0$$

$$3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0$$

$$8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0$$

This function has a minimum at f = 24.31 and
x = (2.17, 2.36, 8.77, 5.09, 0.99, 1.43, 1.32, 9.82, 8.27, 8.37).
The starting point used is x = (2, 3, 5, 5, 1, 2, 7, 3, 6, 10).

Function 7

$$\text{Minimise } f(x) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + (x_{11} - 9)^2 + 10(x_{12} - 1)^2 + 5(x_{13} - 7)^2 + 4(x_{14} - 14)^2 + 27(x_{15} - 1)^2 + x_{16}^4 + (x_{17} - 2)^2 + 13(x_{18} - 2)^2 + (x_{19} - 3)^2 + x_{20}^2 + 95$$

subject to $-3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0$

$-5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0$

$-\frac{1}{2}(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0$

$-x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0$

$-4x_1 - 5x_2 + 3x_7 - 9x_8 + 105 \geq 0$

$-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0$

$-3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0$

$8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0$

$-x_1 - x_2 - 4x_{11} + 21x_{12} \geq 0$

$-x_1^2 - 15x_{11} + 8x_{12} + 28 \geq 0$

$-4x_1 - 9x_2 - 5x_{13}^2 + 9x_{14} + 87 \geq 0$

$-3x_1 - 4x_2 - 3(x_{13} - 6)^2 + 14x_{14} + 10 \geq 0$

$-14x_1^2 - 35x_{15} + 79x_{16} + 92 \geq 0$

$-15x_2^2 - 11x_{15} + 61x_{16} + 54 \geq 0$

$-5x_1^2 - 2x_2 - 9x_{17}^4 + x_8 + 68 \geq 0$

$-x_1^2 + x_2 - 19x_{19} + 20x_{20} - 19 \geq 0$

$-7x_1^2 - 5x_2^2 - x_{19}^2 + 30x_{20} \geq 0$

This function has a minimum at f = 130.60 and x = (2.04, 2.20, 8.74, 5.06, 0.95, 1.43, 1.33, 9.97, 8.17, 8.46, 2.31, 1.35, 6.10, 14.16, 0.99, 0.49, 1.49, 2.00, 2.64, 2.02). The starting point used is x = (2, 3, 5, 5, 1, 2, 7, 3, 6, 10, 2, 2, 6, 15, 1, 2, 1, 2, 1, 3).

As mentioned earlier on, the procedures OPTKOV and POWCON have been tested using the Powell (64)'s method and the Fletcher and Powell's method as a sub-routine. Results are now given only for the first four functions as Powell (64)'s method failed for the larger problems.

OPTKOV

with

	POWELL 64		FLEPOMIN	
	Function Evaluation	Run Time	Function Evaluation	Run Time
Function 1	125	2	191	8
Function 2	776	25	264	21
Function 3	497	17	324	18
Function 4	1236	42	278	17

POWCON

with

	POWELL 64		FLEPOMIN	
	Function Evaluation	Run Time	Function Evaluation	Run Time
Function 1	329	2	86	1
Function 2	1287	43	65	3
Function 3	303	8	47	1
Function 4	540	17	90	6

Looking at these two tables we can now say that the Fletcher and Powell method is more efficient than the Powell (64)'s method. The difference did not show so much for unconstrained problems, but now the minimisation sub-routines being called several times the difference shows more.

From now on all the results tabulated have been obtained using Flepomin as a sub-routine.

As mentioned earlier on, OPTKOV is divided into two phases; phase I finds the bounds and phase II is the iteration phase.

For clarity the phase I will be considered as being the first loop no matter how often the minimisation sub-routine has been called.

The number of loops is equal to the number of times that sub-routine has been called in the whole process.

Function 1, 2, 3, 4 have been tested with EPS = 10^{-6} and functions 5, 6, 7 with EPS = 10^{-4}.

Function 1

	OPTKOV			POWCON			S.U.M.T.		
Loop	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time
	3.3235			3.3235			3.3235		
1	2.6658	26		0.1295	17				
2	2.666666	49		1.2532	35		2.9667	58	
3	2.6660	57		2.2738	45		2.6966	85	
4	2.6661	62		2.6000	53		2.669	110	
5	2.6662	67		2.6535	64		2.6669	135	
6	2.6663	72		2.664	71		2.6665	161	
7	2.6664	77		2.665	82		2.6666	188	7
8	2.6664	82		2.6664	93				
9	2.6665	87		2.6665	102				
10	2.6665	93		2.6665	117				
11	2.6665	98		2.66666	134	1			
12	2.66660	103							
13	2.66662	108							
14	2.66663	114							
15	2.66663	119	7						

From this table and the two following ones we can see that the optimal solution is reached after the third loop. However, the program did not stop there for the required accuracy (BU - BL) $\leq$ EPS, was not satisfied.

Function 2

	OPTKOV			POWCON			S.U.M.T.		
Loop	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time
	0			0			0		
1	-44.185	107		-44.0974	53		-41.4680	38	
2	-44.0019	133		-43.9307	68	7	-43.7580	74	
3	-44.000	172					-43.9758	113	
4	-44.00	188					-43.9975	143	
5	-44.0000	198					-43.9997	175	
6	-44.000	209					-43.9999	219	
7	-44.000	219					-44.0000	252	14
8	-44.000	232							
9	-44.000	249							
10	-44.000	259							
11	-44.000	269							
12	-44.000	278							
13	-44.000	287	22						

Function 3

	OPTKOV			POWCON			S.U.M.T.		
Loop	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time
	7.29			7.29			7.29		
1	0.1106	71		0.1094	23		0.7036	26	
2	0.1111	133		0.1110	43		0.1548	63	
3	0.1111	168		0.11111	61	23	0.1158	87	
4	0.1111	175					0.11158	108	
5	0.1111	184	10				0.11115	129	
							0.111111	149	4

Function 4

	OPTKOV			POWCON			S.U.M.T.		
Loop	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time
	-6.000			-6.000			-6.000		
1	-4.0063	34		-3.5026	32		-3.5026	32	
2	-3.2615	65		-2.8816	53		-2.9731	61	
3	-3.0244	85		-2.923	66		-2.9249	82	
4	-2.9515	101		-2.9194	78		-2.9022	105	
5	-2.9293	116		-2.9197	90		-2.91975	126	
6	-2.9226	132		-2.9197	101	13	-2.91970	147	14
7	-2.9205	148							
8	-2.9199	167							
9	-2.91978	178							
10	-2.91972	187							
11	-2.91970	196							
12	-2.91970	209	16						

Function 5

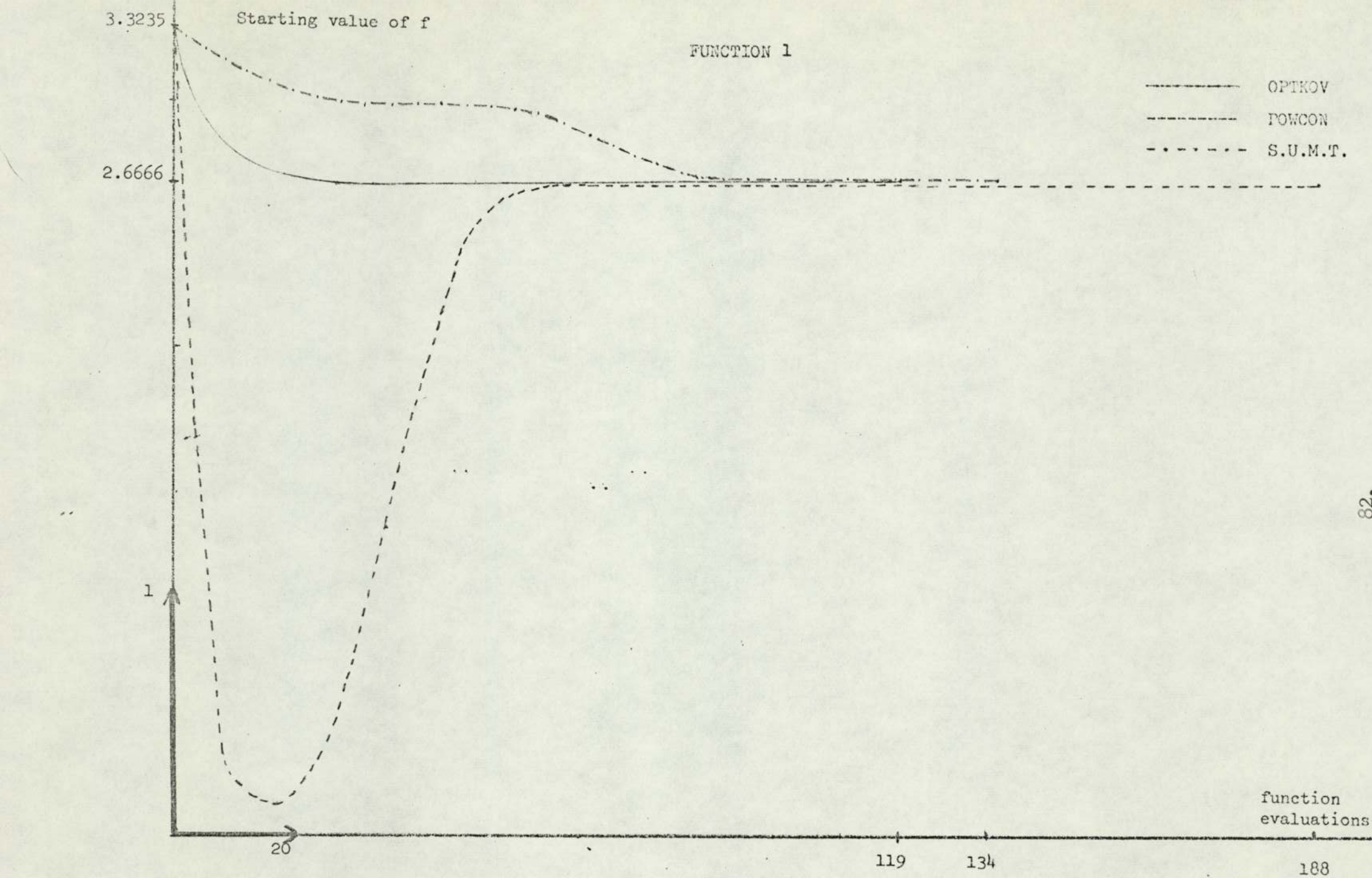
	OPTKOV			POWCON			S.U.M.T.		
Loop	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time
	714			714			714		
1	679	260		679.917	54		682.305	100	
2	680.630	334		680.985	101		680.797	141	
3	680.633	366		680.629	123	15	680.691	151	
4	680.632	381					680.739	160	11
5	680.632	386	38						

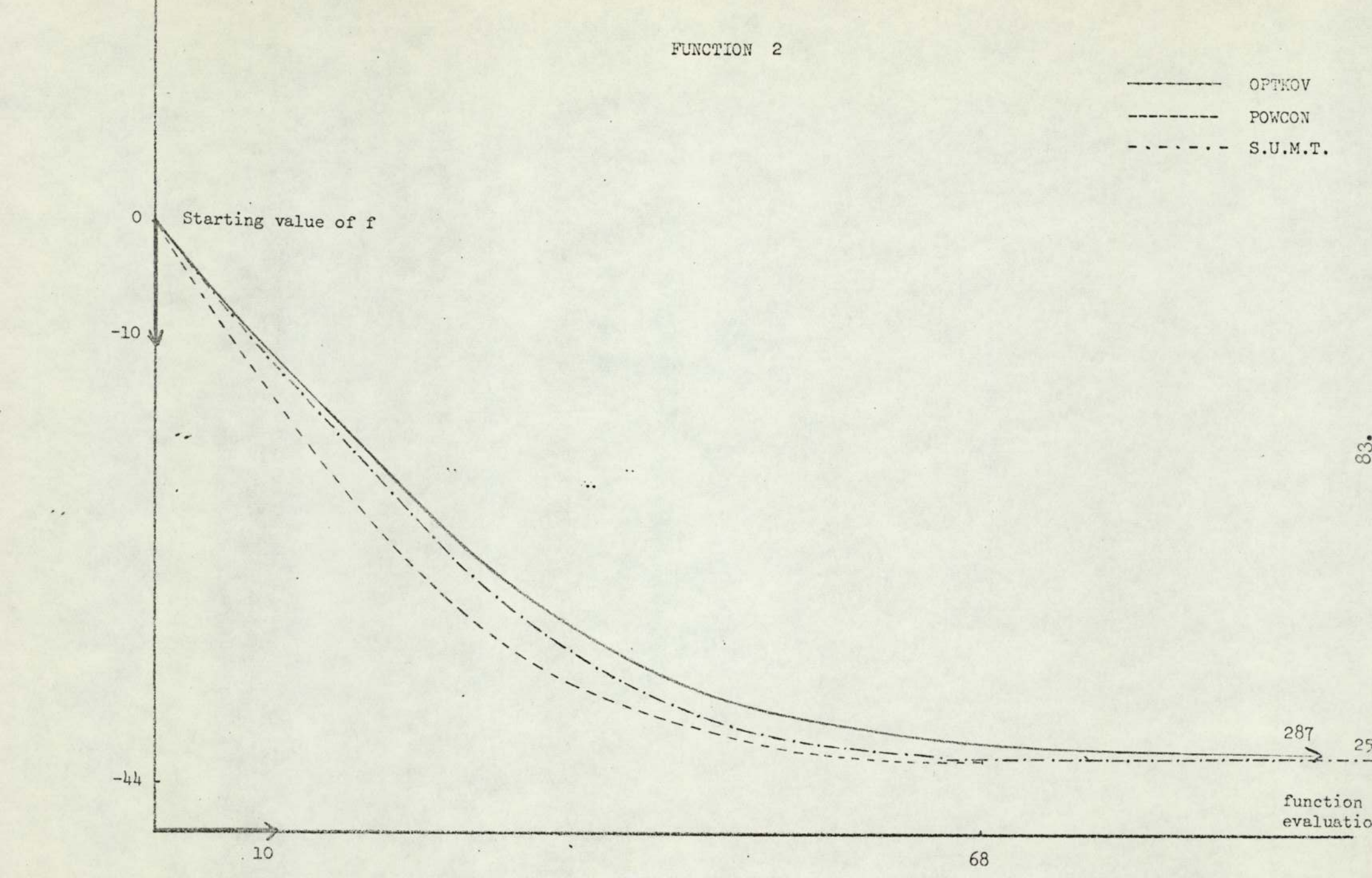
Function 6

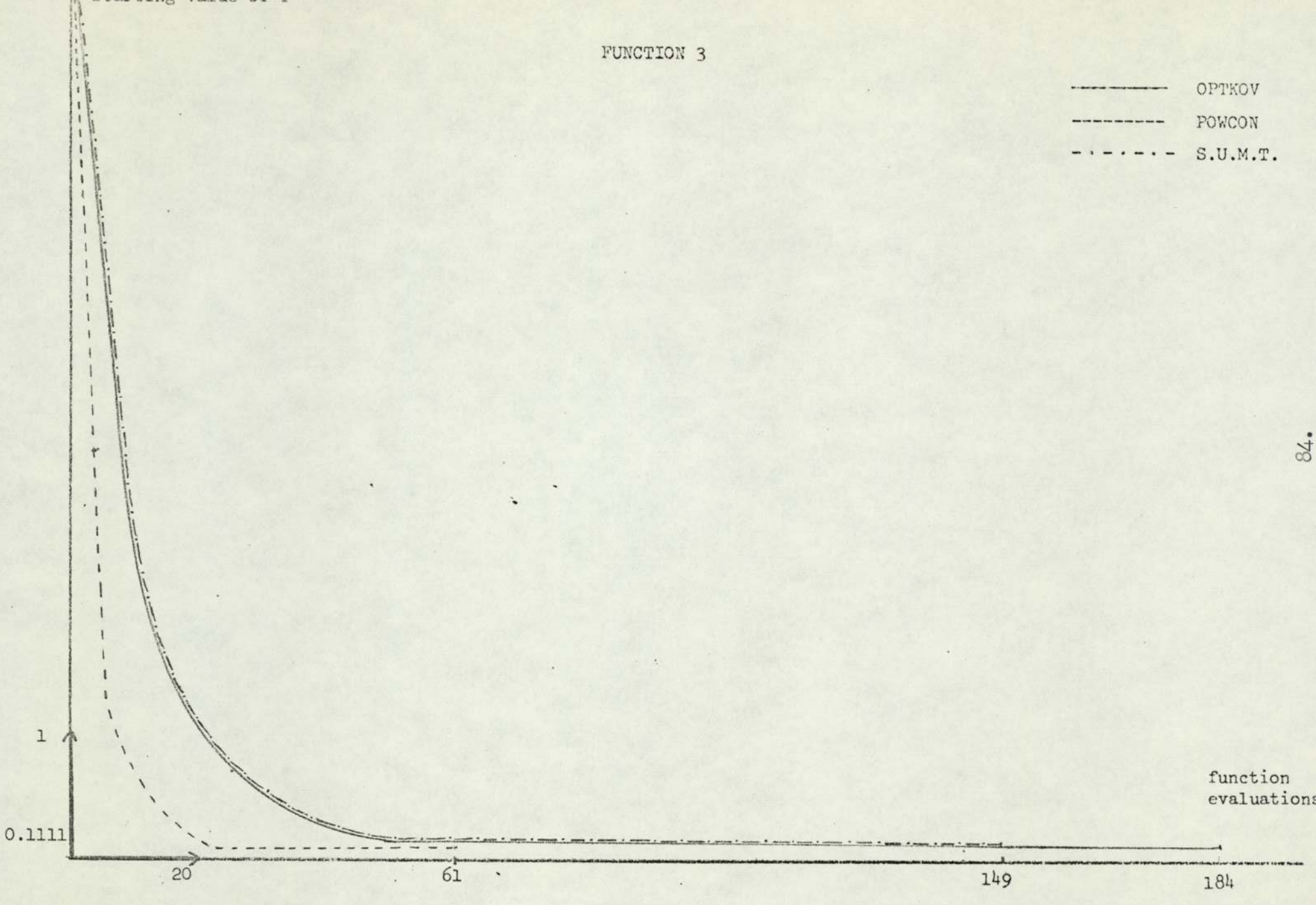
	OPTKOV			POWCON			S.U.M.T.		
Loop	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time
	753			753			753		
1	16.90	368		23.03	147		28.725	139	
2	24.02	468		24.28	193		24.747	233	
3	24.30	582		24.28	199		24.350	347	
4	24.31	621		24.29	237		24.310	532	
5	24.31	654	137	24.29	248		24.307	605	
6				24.30	360		24.307	656	54
7				24.30	272	61			

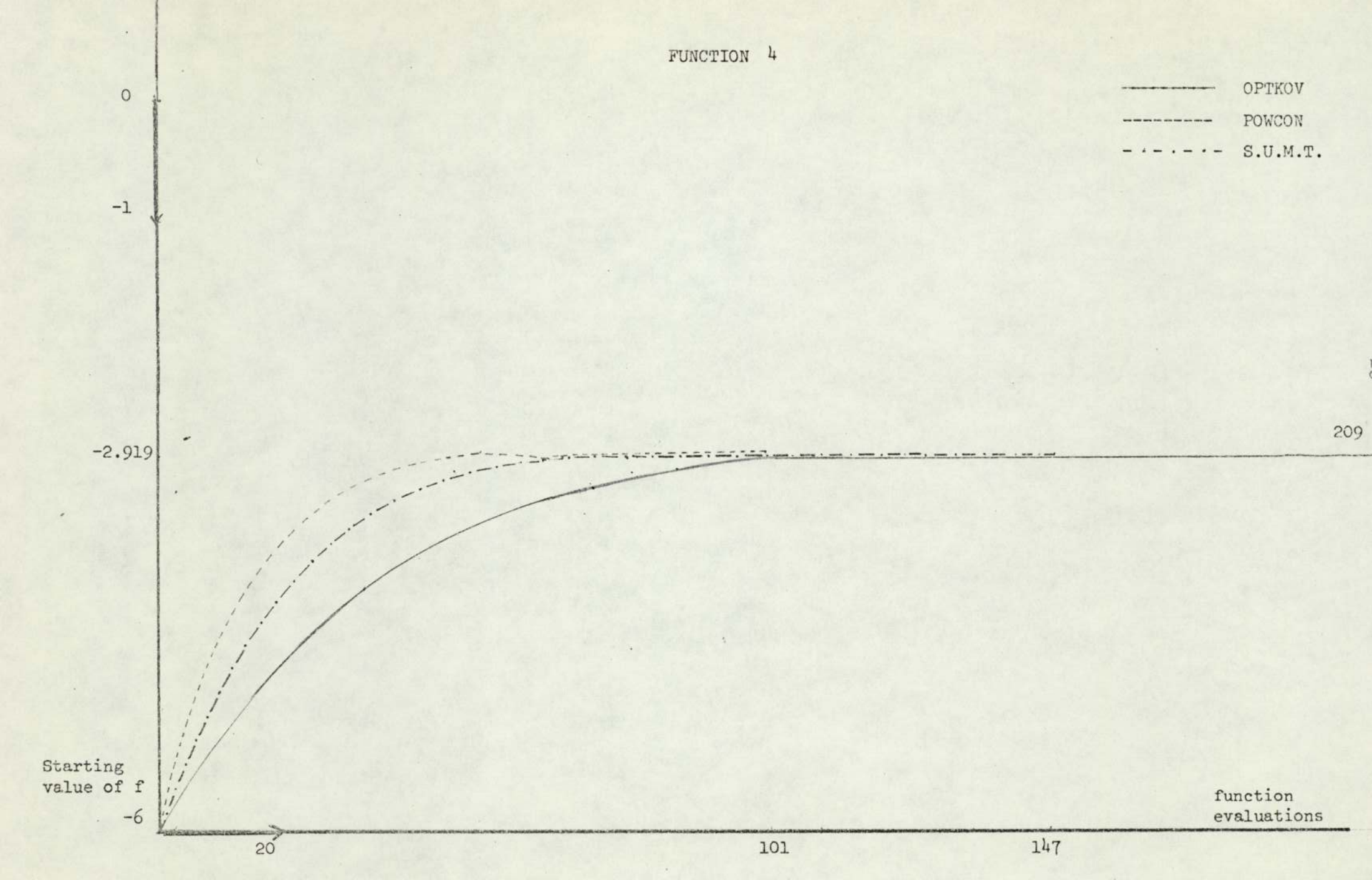
Function 7

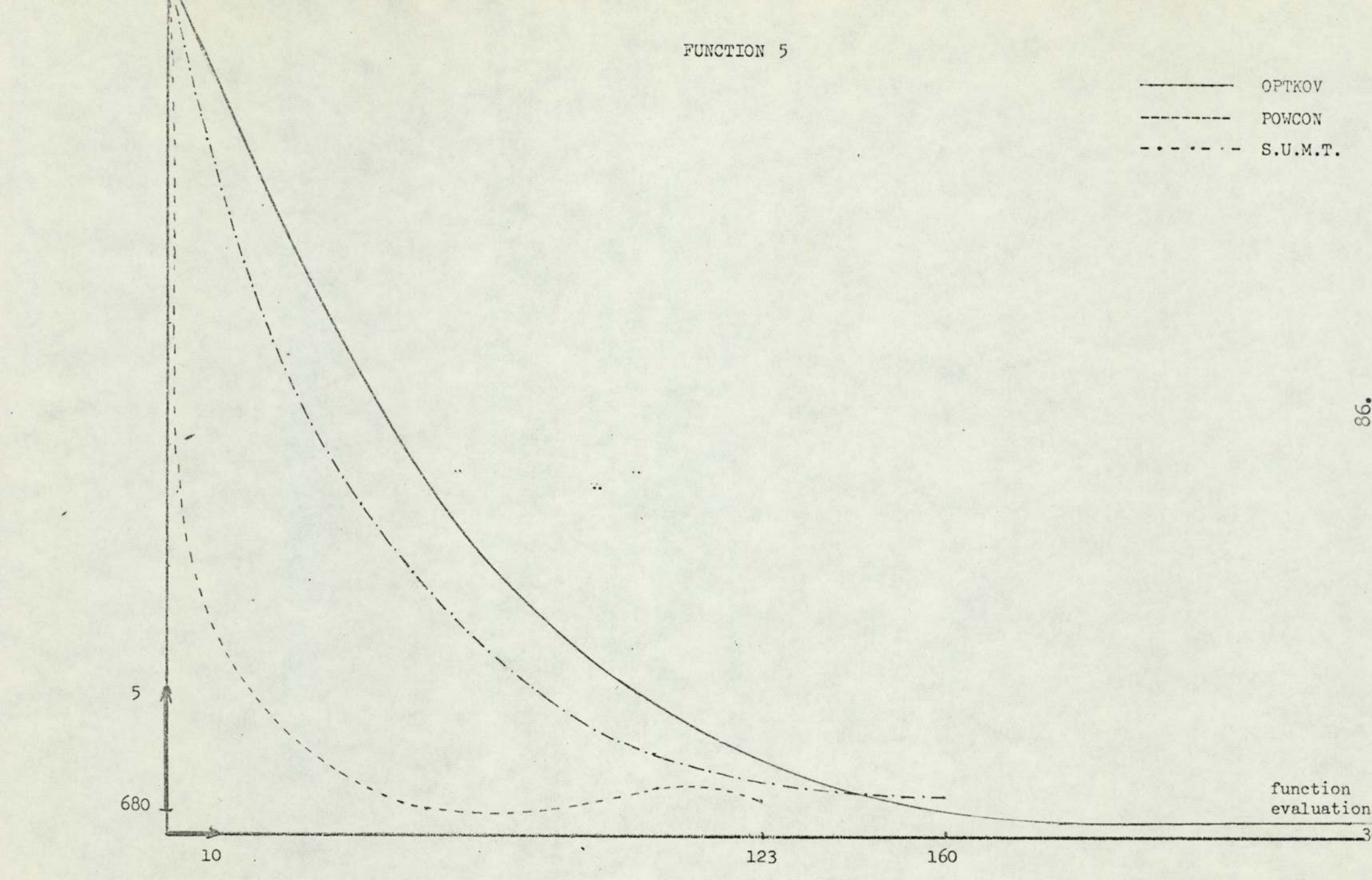
	OPTKOV			POWCON			S.U.M.T.		
Loop	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time	f	Function Evaluation	Run Time
	901			901			901		
1	134.97	498		129.64	178		136.409	244	
2	133.98	583	356	130.23	200		131.073	416	
3				130.77	215		130.547	581	
4				130.77	226		130.515	614	
5				130.70	240		130.513	661	
6				130.248	248		130.489	1120	195
7				130.81	260				
8				130.81	270	207			

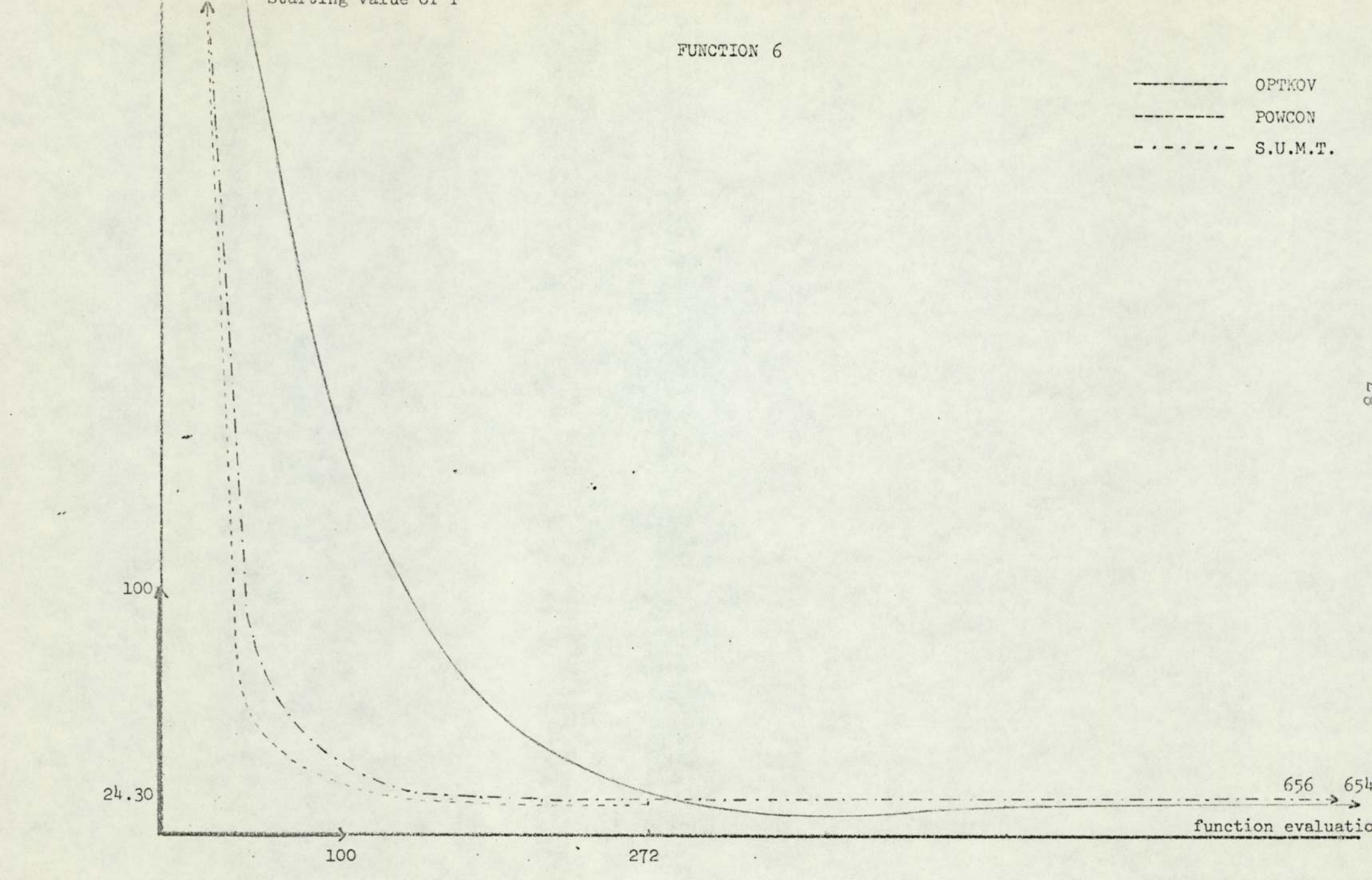

3.3235
Starting value of f
FUNCTION 1
OPTKOV
POWCON
S.U.M.T.
2.6666
1
20
119
134
188
function evaluations

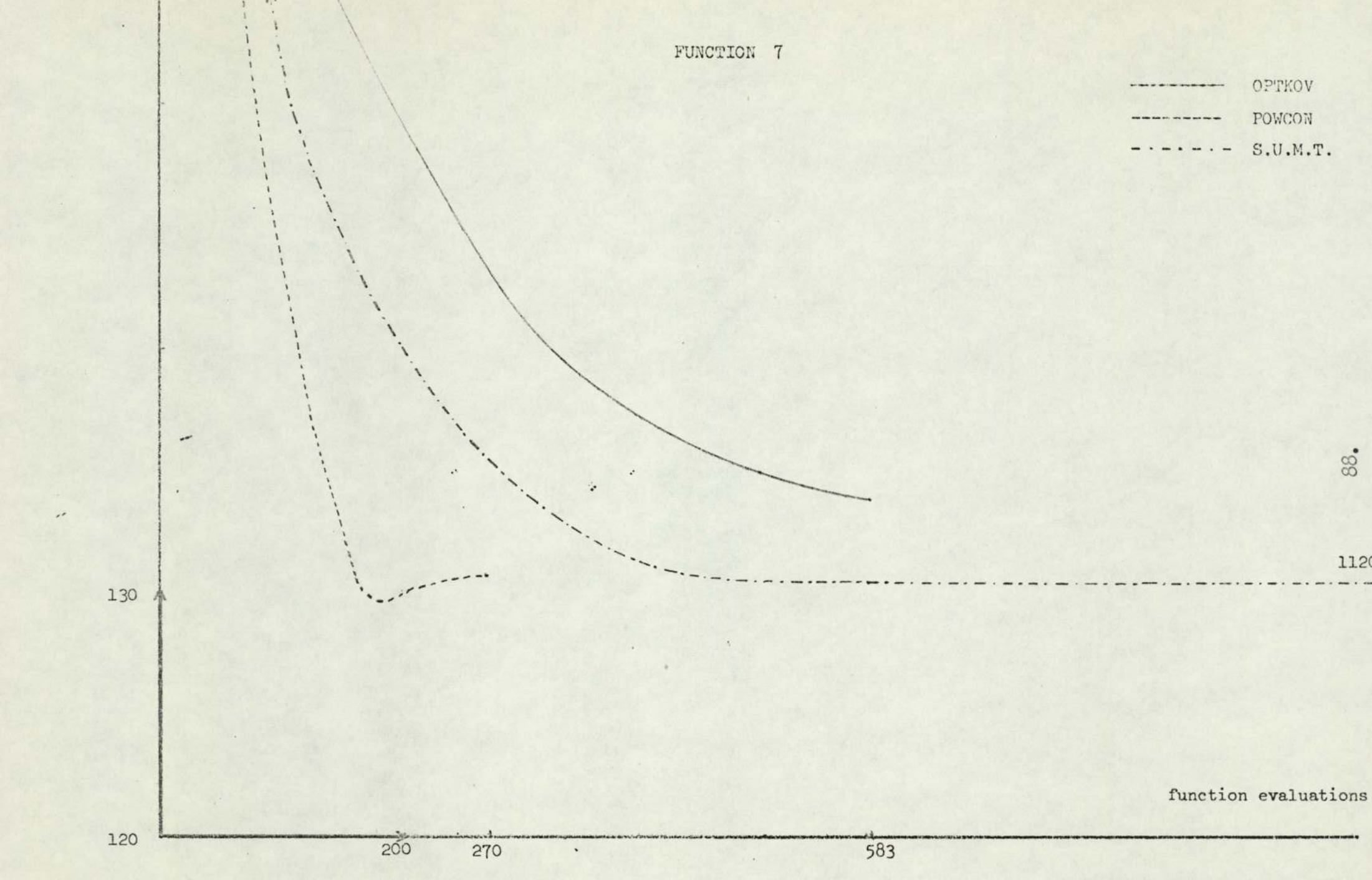

FUNCTION 2
OPTKOV
POWCON
S.U.M.T.
0
Starting value of f
-10
-44
287
25
function
evaluatio
10
68


FUNCTION 3
OPTKOV
POWCON
S.U.M.T.
1
0.1111
20
61
149
184
function evaluations


FUNCTION 4
OPTKOV
POWCON
S.U.M.T.
0
-1
-2.919
209
Starting value of f
-6
function evaluations
20
101
147


FUNCTION 5
OPTKOV
POWCON
S.U.M.T.
5
680
function evaluation
10
123
160
3


Starting Value of F
FUNCTION 6
OPTKOV
POWCON
S.U.M.T.
100
24.30
656
654
function evaluatio
100
272


FUNCTION 7
OPTKOV
POWCON
S.U.M.T.
88.
1120
130
120
200
270
583
function evaluations

3.3 Comparison and Conclusion

It is difficult to say which of run time and function evaluations is the best criterion for judging efficiency.

If we consider the first three previous tables, that is to say problems with less than five variables, the comparison is rather difficult to make as both function evaluations and run time vary a lot.

With larger problems we can notice straight away that OPTKOV becomes relatively inefficient considering both number of function evaluations and run time, even if we look at them from the second loop onwards once the bounds have been found.

If we now look at the results obtained by S.U.M.T. and POWCON we see that the former is relatively good considering the run time on the computer, better than POWCON, though requiring more function evaluations.

Therefore we can draw two conclusions, depending upon which criterion is considered for judging efficiency:

(a) if for some reason the number of function evaluations is more important, then POWCON could be the best method considered here;

(b) otherwise, if the computer run time is the most important criterion, then S.U.M.T. is the best method.

IV

FURTHER RESEARCH WORK

An interesting point discovered during various experiments is worth mentioning.

We have seen that all three methods convert the constrained problem into a series of unconstrained ones. At each iteration, the minimization procedure is called.

Considering Flepomin, we can notice that at the beginning of the program the Hessian matrix is set equal to the unit matrix and an experiment has been made to see what would be the effect of using, from the second iteration onwards, the Hessian matrix obtained at the previous iteration instead of resetting it equal to I.

This experiment proved to be very successful, particularly for S.U.M.T. For instance the results obtained with this modification for function 7 are as follows:

S.U.M.T. (originally) 1120 function evaluations
195 seconds run time;

S.U.M.T. (including the modification) 601 function evaluations
152 seconds run time.

The results obtained with POWCON including the same modification have not improved the results; this can easily be explained as, in this procedure, two parameters σ and θ change at each iteration and this makes the Hessian matrix vary a lot from one iteration to another.

With OPTKOV, when results were obtained, the modification improved the results too. For example, with function 5:

OPTKOV (originally) 386 function evaluations
38 seconds run time

OPTKOV (with modification 215 function evaluations
29 seconds run time.

Therefore, here again the modification brought some improvement. Nevertheless, with function 6 and function 7, no convergence was obtained, the Hessian matrix tending to go singular.

Complete results are given in the following table.

		Function 5		Function 6		Function 7	
		Ev	Run Time	Ev	Run Time	Ev	Run Time
POWCON	Original	123	15	272	61	270	216
	Modified	130	20	250	60	289	259
OPTKOV	Original	386	38	654	137	583	356
	Modified	215	29	NO RESULTS			
SUMT	Original	160	11	656	78	1120	231
	Modified	203	7	317	38	601	152

Therefore we can say that a method using the Hessian matrix can be very much improved by using a good approximation to the Hessian instead of the unit matrix from major iteration to major iteration and further work should be done on how to evaluate this approximation.

We have seen, too, that S.U.M.T. only has two different penalty functions depending upon whether the problem we are dealing with has equality constraints or inequality constraints.

POWCON and OPTKOV have only one penalty function for equality constraints, therefore problems with inequality constraints have to be converted using the Heaviside function (procedure H(t)).

Calling this procedure H(t) each time the procedure FUNCT is called might lengthen the run time of the program. Therefore if we could find, for these two methods, a penalty function dealing with both equality and inequality constraints, this could make the methods more competitive.

REFERENCES

1. BOX, M.J.
'A comparison of several current optimization methods and the use of transformation in constrained problems'
Computer Journal, vol.9, no.1, 1966.

2. DAVIDON, W.C.
'Variable metrix methods for minimization'
Argonne National Laboratory, Report ANL 5990 Rev.
Illinois, 1959.

3. FIACCO and McCORMICK
'Programming under non-linear constraints by unconstrained minimization: A Primal Dual method'
Management Science, vol.10, no.2, January 1964.

4. FIACCO and McCORMICK
'Computational Algorithm for S.U.M.T.'
Management Science, vol.10, no.4, July 1964.

5. FIACCO and McCORMICK
'Slacked Unconstrained Minimization Technique for convex programming'
S.I.A.M. Journal, no.15, 1967.

6. FIACCO and McCORMICK
'S.U.M.T. without parameters'
Operations Research, vol.15, no.5, September/October 1967.

7. FIACCO and McCORMICK
'S.U.M.T. for convex programming with equality constraints'
R.A.C. T.P. 155, April 1965.

8. FLETCHER, R. and POWELL, M.J.D.
'A rapidly convergent method for minimization'
Computer Journal, vol.6, July 1963.

9. FLETCHER, R.
'A new approach to variable metric algorithms'
Computer Journal, vol.13, August 1970.

10. KELLEY, N.J.
'Optimization Techniques with applications to aerospace systems'
George Leitmann (ed.) Academic Press, New York, 1962.

11. KOWALIK, I., OSBORNE, M.R., and RYAN, D.M.
'A new method for constrained optimization problems'
J.O.R.S.A., vol.17, no.6, 1969.

12. KOWALIK, J.
'Non-linear programming procedures and design optimization'
Acta Polytechnica Scandinavica, Mathematics and Computing Machinery Series N.R.13, Trondheim, 1966.

13. MORRISON, D.D.
'Optimization by least squares'
S.I.A.M. Journal, Numerical Analysis, vol.5, no.3, 1968.

14. MURTAGH, B.A. and SARGENT, R.W.H.
'Computational experience with quadratically convergent methods'
Computer Journal, vol.13, no.2, May, 1970.

15. NELDER J.H. and MEAD, R.
'A Simplex Method for function minimization'
Computer Journal, vol.7, 1964.

16. POWELL, M.J.D.
'An efficient method for finding the minimum of a function of several variables without calculating derivatives'
Computer Journal, vol.7, no.4, January 1965.

17. POWELL, M.J.D.
'A method for non-linear constraints in minimization problems'
Optimization, edited by R. Fletcher, Academic Press, London and New York, 1969.

18. SCHMIT, L.A., and FOX, R.L.
'Advances in the integrated approach to Structural Synthesis'
A.I.A.A. 6th Ann. Struct. and Materials Conference, Palm Springs, California, 1965.

19. SPENDLEY, W., HEXT, G.R., and HIMSWORTH, F.R.
'Sequential application of Simplex Designs in Optimization and Evolutionary Operation'
Technometrics, vol.4, no.4, November 1962.

20. (a) WELLS, M.
Electronic Computing Laboratory, University of Leeds,
Collected algorithms from C.A.C.M.,
Algorithm 251,
Function minimization (E4).

(b) FLETCHER, R.

Electronic Computing Laboratory, University of Leeds, Certification of Algorithm 251 (E4)

21. WONG, K.P.

Ph.D. thesis: Decentralized planning by vertical decomposition of an economic system: a non-linear approach Ph.D. in National Economic Planning, University of Birmingham.

22. ZANGWILL, W.I.

'Minimizing a function without calculating derivatives' Computer Journal, vol. 10, no. 3, November 1967.

BIBLIOGRAPHY

ABADIE, J.
'Non linear Programming'
North-Holland, Amsterdam, 1967.

FIACCO, A.V. and McCORMICK, G.P.
'Sequential Unconstrained Minimization Techniques for Non-Linear Programming'
John Wiley and Sons Inc., New York, 1968.

FLETCHER, R.
'Optimization'
Academic Press, London and New York, 1969.

HADLEY, G.
'Non linear and Dynamic Programming'
Addison-Wesley, Reading, Massachussetts, 1969.

KOWALIK, J. and OSBORNE, M.R.
'Methods for Unconstrained Optimization Problems'
Elsevier, New York-London-Amsterdam, 1968.

KUNZI, H.P., KRELLE, W. and OETTI, W.
'Non linear programming'
Blaisdell Publishing Company, 1966.

ZANGWILL, W.I.
'Non Linear Programming'
Prentice Hall, Englewood Cliffs, New Jersey, 1969.

APPENDIX I

LISTING OF PROGRAMS FOR UNCONSTRAINED OPTIMIZATION PROBLEMS

SIMPLEX PROGRAM

```
'BEGIN'
      'INTEGER' COUNT,I,J,N,H,L,COLIMIT;
      'REAL' A,B,C,CONVER,CRITER,Y1,Y2,Y3,SUMY,STORE,BARY,VAR,X1,X2;
      N:=READ;
      A:=READ;
      B:=READ;
      C:=READ;
      CRITER:=READ;
      COLIMIT:=READ;
      COUNT:=0;
      'BEGIN'
            'REAL' 'ARRAY' P[0:N,1:N],ORP,STL,CENT,SUM,P1,P2,P3,PH[1:N]
                           Y[0:N];
            ***********  INSERT PROCEDURE F1(N,X,Z) HERE
            ***********  INSERT PROCEDURE F2(PP,F) HERE
            'PROCEDURE' MAX(A,N,M,ROW);
            'VALUE' N,M,A;
            'INTEGER' N,M,ROW;
            'ARRAY' A;
            'BEGIN'
                  'INTEGER' I;
                  'REAL' Q;
                  Q:=A[M];
                  ROW:=0;
                  'FOR' I:=M+1 'STEP' 1 'UNTIL' N 'DO'
                  'BEGIN'
                        'IF' A[I]>Q 'THEN'
                        'BEGIN'
                              Q:=A[I];
                              ROW:=I;
                        'END' 'ELSE'
                        Q:=Q;
                  'END';
            'END';
            'PROCEDURE' MIN(A,N,M,ROW);
            'VALUE' N,M,A;
            'INTEGER' N,M,ROW;
```

```
'ARRAY' A;
'BEGIN'
      'INTEGER' I;
      'REAL' Q;
      Q:=A[M];
      ROW:=0;
      'FOR' I:=M+1 'STEP' 1 'UNTIL' N 'DO'
      'BEGIN'
            'IF' A[I]<Q 'THEN'
            'BEGIN'
                  Q:=A[I];
                  ROW:=I;
            'END' 'ELSE'
            Q:=Q;
      'END';
'END';
'PROCEDURE' REFLECT(A,CEN,NO,N,P,P1);
'VALUE' A,CEN,NO,N;
'REAL' A;
'INTEGER' NO,N;
'ARRAY' P,P1,CEN;
'BEGIN'
      'INTEGER' I;
      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
      P1[I]:=(1+A)*CEN[I]-A*P[NO,I];
'END';
'PROCEDURE' EXPAND(C,CEN,P1,P2,N);
'VALUE' C,P1,CEN,N;
'REAL' C;
'INTEGER' N;
'ARRAY' CEN,P1,P2;
'BEGIN'
      'INTEGER' I;
      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
      P2[I]:=C*P1[I]+(1-C)*CEN[I];
'END';
'PROCEDURE' CONTRACT(B,CEN,P,NO,N,P3);
'VALUE' CEN,B,NO,N;
'REAL' B;
'INTEGER' N,NO;
```

```
            'ARRAY' CEN,P,p3;
            'BEGIN'
                  'INTEGER' I;
                  'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
                  p3[I]:=B*P[NO,I]+(1-B)*CEN[I];
            'END';
            'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
            ORP[J]:=READ;
'COMMENT'   ORIGINAL POINTS;
            'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
            STL[J]:=READ;
'COMMENT'   STEP LENGTH;
            'FOR' I:=0 'STEP' 1 'UNTIL' N 'DO'
            'BEGIN'
                  'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
                  'IF' I=J 'THEN' P[I,J]:=ORP[J]+STL[J]
                  'ELSE'
                  P[I,J]:=ORP[J];
            'END';
            F1(N,P,Y);
            COUNT:=COUNT+1;
AGAIN;      VAR:=9999;
            MAX(Y,N,0,H);
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            PH[I]:=P[H,I];
            MIN(Y,N,0,L);
            'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
            'BEGIN'
                  SUM[J]:=0.0;
                  'FOR' I:=0 'STEP' 1 'UNTIL' N 'DO'
                  SUM[J]:=SUM[J]+P[I,J];
                  CENT[J]:=(SUM[J]-P[H,J])/N;
            'END';
            REFLECT(A,CENT,H,N,P,P1);
            F2(P1,Y1);
            COUNT:=COUNT+1;
            VAR:=0;
            'IF' Y1<Y[L] 'THEN'
            'BEGIN'
                  EXPAND(C,CENT,P1,P2,N);
```

```
            F2(p2,Y2);
            COUNT:=COUNT+1;
            VAR:=-1;
      'END' 'ELSE' 'GOTO' L1;
      'IF' Y2<Y[L] 'THEN'
      'BEGIN'
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            'BEGIN'
                  P[H,I]:=P2[I];
                  PH[I]:=P[H,I];
            'END';
            Y[H]:=Y2;
            'GOTO' CHECK;
      'END' 'ELSE'
      'BEGIN'
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            'BEGIN'
                  P[H,I]:=P1[I];
                  PH[I]:=P[H,I];
            'END';
            Y[H]:=Y1;
            'GOTO' CHECK;
      'END';
L1;   'IF' Y[1]>Y[0] 'THEN'
      'BEGIN'
            X1:=Y[1];
            X2:=Y[0];
      'END' 'ELSE'
      'BEGIN'
            X1:=Y[0];
            X2:=Y[1];
      'END';
      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
      'BEGIN'
            'IF' Y[I]>X1 'THEN'
            'BEGIN'
                  X2:=X1;
                  X1:=Y[I];
            'END';
            'IF' Y[I]>X2 'AND' Y[I]<X1 'THEN' X2:=Y[I];
```

```
                'END';
                'IF' Y1>X2 'THEN' 'GOTO' L2
                'ELSE'
                'BEGIN'
                      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
                      'BEGIN'
                            P[H,I]:=P1[I];
                            PH[I]:=P[H,I];
                      'END';
                      Y[H]:=Y1;
                      'GOTO' CHECK;
                'END';
L2:             'IF' Y1>Y[H] 'THEN' 'GOTO' L3
                'ELSE'
                'BEGIN'
                      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
                      'BEGIN'
                            P[H,I]:=P1[I];
                            PH[I]:=P[H,I];
                      'END';
                'END';
L3:             CONTRACT(B,CENT,P,H,N,P3);
                F2(P3,Y3);
                COUNT:=COUNT+1;
                VAR:=1;
                'IF' Y3>Y[H] 'THEN'
                'BEGIN'
                      'FOR' I:=0 'STEP' 1 'UNTIL' N 'DO'
                      'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
                      P[I,J]:=(P[I,J]+P[L,J])/2;
                      F1(N,P,Y);
                      'GOTO' CHECK;
                'END''ELSE'
                'BEGIN'
                      'FOR'I:=1 'STEP' 1 'UNTIL' N 'DO'
                      'BEGIN'
                            P[H,I]:=P3[I];
                            PH[I]:=P[H,I];
                      'END';
                      Y[H]:=Y3;
```

```
                'END';
CHECK:          SUMY:=STORE:=0.0;
                'FOR' I:=0 'STEP' 1 'UNTIL' N 'DO'
                'BEGIN'
                      SUMY:=SUMY+Y[I];
                'END';
                BARY:=SUMY/N;
                'FOR' I:=0 'STEP' 1 'UNTIL' N 'DO'
                STORE:=STORE+(Y[I]-BARY)*(Y[I]-BARY);
                CONVER:=SQRT(STORE/N);
                'IF'  COUNT>COLIMIT 'THEN' 'GOTO' FINISH;
                'IF'  CONVER<CRITER 'THEN' 'GOTO' FINISH
                'ELSE'
                'GOTO' AGAIN;
FINISH:         'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
                'BEGIN'
                      PRINT(P[H,I],0,10);
                'END';
                'IF' VAR=0 'THEN' PRINT(Y1,0,10);
                'IF' VAR=-1 'THEN' PRINT(Y2,0,10);
                'IF' VAR=1 'THEN' PRINT(Y3,0,10);
      'END';
'END';
```

```
'PROCEDURE' POW64(X,E,N,ESCALE,IPRINT,ICON,MAXIT,F);
'REAL' ESCALE;
'INTEGER' N,IPRINT,ICON,MAXIT;
'ARRAY' X,E,F;
'BEGIN'
      'ARRAY' W[1:N*(N+3)];
      'REAL' DDMAG,FKEEP,SCER,SUM,FP,DMAX,DACC,DDMX,
             D,DL,FPREV,FA,DA,DD,FB,DB,FHOLD,DMAG,FC
             ,DC,A,B,DI,FI,AAA;
      'INTEGER' JJ,JJJ,K,NFCC,IND,INN,I,J,ITERC,ISGRAD,
                ITONE,IXP,IDIRN,ILINE,IS,JIL;
      'SWITCH' SW1:=L10,L11,L12,L13,L14,L96,L5,L7,L8,L58,
                    L18,L15,L24,L21,L23,L83,L25,L26,L28,
                    L29,L71,L30,L34,L44,L86,L45,L47,L41,
                    L50,L51,L94,L92,L87,L61,L37,L38,L72,
                    L53,L76,L78,L88,L35,L108,L101,L105,
                    L115,L113,L107,L106,L20;
      ***********  INSERT PROCEDURE FUNCT HERE
      DDMAG:=0.1*ESCALE;
      SCER:=0.05/ESCALE;
      JJ:=N*N+N;
      JJJ:=JJ+N;
      K:=N+1;
      NFCC:=IND:=INN:=ITERC:=1;
      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
      'BEGIN'
            'FOR' J:=1 'STEP' 1 'UNTIL' N 'DO'
            'BEGIN'
                  'IF' I=J  'THEN'
                  'BEGIN'
                        W[K]:=ABS(E[I]);
                        W[I]:=ESCALE;
                  'END' 'ELSE'
                  W[K]:=0;
                  K:=K+1;
             'END';
      'END';
```

```
        ISGRAD:=2;
        FUNCT(1,X,F);
        FKEEP:=2*ABS(F[1]);
L5:     ITONE:=1;
        FP:=F[1];
        SUM:=0;
        IXP:=JJ;
        'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
        'BEGIN'
              IXP:=IXP+1;
              W[IXP]:=X[I,1]
        'END';
        IDIRN:=N+1;
        ILINE:=1;
L7:     DMAX:=W[ILINE];
        DACC:=DMAX*SCER;
        DMAG:='IF' DDMAG<0.1*DMAX 'THEN' DDMAG 'ELSE' 0.1*DMAX;
        DMAG:='IF' DMAG>20*DACC 'THEN' DMAG 'ELSE' 20*DACC;
        DDMX:=10*DMAG;
        'IF' ITONE=3 'THEN' 'GOTO' L71;
        DL:=0;
        D:=DMAG;
        FPREV:=F[1];
        IS:=5;
        FA:=F[1];
        DA:=DL;
L8:     DD:=D-DL;
        DL:=D;
L58:    K:=IDIRN;
        'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
        'BEGIN'
              X[I,1]:=X[I,1]+DD*W[K];
              K:=K+1;
        'END';
        FUNCT(1,X,F);
        NFCC:=NFCC+1;
        'GOTO' SW1[IS];
L14:    'IF' F[1]<FA 'THEN' 'GOTO' L15;
        'IF' F[1]>FA 'THEN' 'GOTO' L24;
        'IF' ABS(D)>DMAX 'THEN' 'GOTO' L18;
```

```
        D:=2*D;
        'GOTO' L8;
L18:    WRITETEXT('('MAXIMUM%CHANGE%DOES%NOT%ALTER%FUNCTION')');
        'GOTO' L20;
L15:    FB:=F[1];
        DB:=D;
        'GOTO' L21;
L24:    FB:=FA;
        DB:=DA;
        FA:=F[1];
        DA:=D;
L21:    'IF' ISGRAD=1 'THEN' 'GOTO' L83;
L23:    D:=2*DB-DA;
        IS:=1;
        'GOTO' L8;
L83:    D:=0.5*(DA+DB-(FA-FB)/(DA-DB));
        IS:=4;
        'IF'(DA-D)*(D-DB) 'GE' 0 'THEN' 'GOTO' L8;
L25:    IS:=1;
        'IF' ABS(D-DB) 'LE' DDMX 'THEN' 'GOTO' L8;
L26:    D:=DB+('IF' DB 'GE' DA 'THEN' ABS(DDMX)
                'ELSE' (-ABS(DDMX)));
        IS:=1;
        DDMX:=2*DDMX;
        DDMAG:=2*DDMAG;
        'IF' DDMX 'LE' DMAX 'THEN' 'GOTO' L8;
        DDMX:=DMAX;
        'GOTO' L8;
L13:    'IF' F[1] 'GE' FA 'THEN' 'GOTO' L23;
L28:    FC:=FB;
        DC:=DB;
L29:    FB:=F[1];
        DB:=D;
        'GOTO' L30;
L12:    'IF' F[1] 'LE' FB 'THEN' 'GOTO' L28;
        FA:=F[1];
        DA:=D;
        'GOTO' L30;
L11:    'IF' F[1] 'GE' FB 'THEN' 'GOTO' L10;
        FA:=FB;
```

```
        DA:=DB;
        'GOTO' L29;
L71:    DL:=D:=1;
        DDMX:=5;
        FA:=FP;
        DA:=-1;
        FB:=FHOLD;
        DB:=0;
L10:    FC:=F[1];
        DC:=D;
L30:    A:=(DB-DC)*(FA-FC);
        B:=(DC-DA)*(FB-FC);
        'IF' (A+B)*(DA-DC)>0 'THEN' 'GOTO' L34;
        FA:=FB;
        DA:=DB;
        FB:=FC;
        DB:=DC;
        'GOTO' L26;
L34:    D:=0.5*(A*(DB+DC)+B*(DA+DC))/(A+B);
        DI:=DB;
        FI:=FB;
        'IF' FB 'LE' FC 'THEN' 'GOTO' L44;
        DI:=DC;
        FI:=FC;
L44:    'IF' ITONE 'NE' 3 'THEN' 'GOTO' L86;
        ITONE:=2;
        'GOTO' L45;
L86:    'IF' ABS(D-DI)'LE' DACC 'THEN' 'GOTO' L41;
        'IF' ABS(D-DI)'LE' 0.03*ABS(D) 'THEN' 'GOTO' L41;
L45:    'IF' (DA-DC)*(DC-D)<0 'THEN' 'GOTO' L47;
        FA:=FB;
        DA:=DB;
        FB:=FC;
        DB:=DC;
        'GOTO' L25;
L47:    IS:=2;
        'IF' (DB-D)*(D-DC) 'GE' 0 'THEN' 'GOTO' L8;
        IS:=3;
        'GOTO' L8;
L41:    F[1]:=FI;
```

```
        D:=DI-DL;
        DD:=SQRT((DC-DB)*(DC-DA)*(DA-DB)/(A+B));
        'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
        'BEGIN'
              X[I,1]:=X[I,1]+D*W[IDIRN];
              W[IDIRN]:=DD*W[IDIRN];
              IDIRN:=IDIRN+1
        'END';
        W[ILINE]:=W[ILINE]/DD;
        ILINE:=ILINE+1;
        'IF'  IPRINT 'NE' 1 'THEN' 'GOTO' L51;
L50:    PRINT(ITERC,0,10);
        PRINT(NFCC,0,10);
        PRINT(F[1],0,10);
        'IF' IPRINT=2  'THEN' 'GOTO' L53;
L51:    'IF' ITONE=2 'THEN' 'GOTO' L38;
        'IF' FPREV<(F[1]+SUM) 'THEN' 'GOTO' L94;
        SUM:=FPREV-F[1];
        JIL:=ILINE;
L94:    'IF' IDIRN 'LE' JJ 'THEN' 'GOTO' L7;
        'IF' IND=2  'THEN' 'GOTO' L72;
L92:    FHOLD:=F[1];
        IS:=6;
        IXP:=JJ;
        'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
        'BEGIN'
              IXP:=IXP+1;
              W[IXP]:=X[I,1]-W[IXP]
        'END';
        DD:=1
        'GOTO' L58;
L96:    'IF'IND=2 'THEN' 'GOTO' L87;
        'IF' FP<F[1] 'THEN' 'GOTO' L37;
        'IF' ABS(FP-F[1]) 'LE' &-15 'THEN'
        'BEGIN'
              PRINT(F[1],0,10);
              'GOTO' L20;
        'END';
        D:=2*(FP+F[1]-2*FHOLD)/(FP-F[1])↑2;
        'IF' D*(FP-FHOLD-SUM)↑2 'GE' SUM 'THEN' 'GOTO' L37;
```

```
L87:  J:=JIL+N+1;
      'IF' J>JJ 'THEN' 'GOTO' L61;
      'FOR' I:=J 'STEP' 1 'UNTIL' JJ 'DO'
      'BEGIN'
            K:=I-N;
            W[K]:=W[I];
      'END';
      'FOR' I:=JIL 'STEP' 1 'UNTIL' N 'DO'
      W[I-1]:=W[I];
L61:  IDIRN:=K:=IDIRN-N;
      ITONE:=3;
      IXP:=JJ;
      AAA:=0;
      'FOR'I:=1 'STEP' 1 'UNTIL' N 'DO'
      'BEGIN'
            IXP:=IXP+1;
            W[K]:=W[IXP];
            'IF' AAA<ABS(W[K]/E[I]) 'THEN' AAA:=ABS(W[K]/E[I]);
            K:=K+1;
      'END';
      DDMAG:=1;
      W[N]:=ESCALE/AAA;
      ILINE:=N;
      'GOTO'L7;
L37:  IXP:=JJ
      AAA:=0;
      E[1]:=EHOLD;
      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
      'BEGIN'
            IXP:=IXP+1;
            X[I,1]:=X[I,1]-W[IXP];
            'IF' AAA*ABS(E[I])<ABS(W[IXP]) 'THEN'
            AAA:=ABS(W[IXP/E[I]);
      'END';
      'GOTO' L72;
L38:  AAA:=AAA*(1+DI);
      'IF' IND =2  'THEN' 'GOTO' L106;
L72:  'IF' IPRINT 'GE' 2 'THEN' 'GOTO' L50;
L53:  'IF' IND=2 'THEN' 'GOTO' L88;
      'IF' AAA>0.1 'THEN' 'GOTO' L76;
```

```
      'IF' ICON=1 'THEN' 'GOTO' L20;
      IND:=2;
      'IF' INN=2 'THEN' 'GOTO' L101;
      INN:=2;
      K:=JJJ;
      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
      'BEGIN'
            K:=K+1;
            W[K]:=X[I,1];
            X[I,1]:=X[I,1]+10*E[I];
      'END';
      FKEEP:=F[1];
      FUNCT(1,X,F);
      NFCC:=NFCC+1;
      DDMAG:=0;
      'GOTO' L108;
L76:  'IF' F[1]<FP  'THEN' 'GOTO' L35;
L78:  WRITETEXT('('ACCURACY%LIMITED%BY%ERROR%IN%F')');
      'GOTO'L20;
L88:  IND:=1;
L35:  DDMAG:=0.4*SQRT(FP-F[1]);
      ISGRAD:=1;
L108:  ITERC:=ITERC+1;
      'IF' NFCC 'LE' MAXIT 'THEN' 'GOTO' L5;
      WRITETEXT('('FUNCTION%EVALUATIONS%COMPLETED')');
      'IF'F[1] 'LE' FKEEP  'THEN' 'GOTO' L20;
      F[1]:=FKEEP;
      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
      'BEGIN'
            JJJ:=JJJ+1;
            X[I,1]:=W[JJJ];
      'END';
      'GOTO' L20;
L101: JIL:=1;
      FP:=FKEEP;
      'IF' F[1]<FKEEP 'THEN' 'GOTO' L105  'ELSE'
      'IF' F[1]=FKEEP 'THEN'  'GOTO' L78;
      JIL:=2;
      FP:=F[1];
      F[1]:=FKEEP;
```

```
L105: IXP:=JJ;
      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            K:=IXP+N;
      'BEGIN'
            IXP:=IXP+1;
            'IF' JIL=2 'THEN' 'GOTO' L115;
            W[IXP]:=W[K];
            'GOTO' L113;
L115:       W[IXP]:=X[I,1];
            X[I,1]:=W[K];
L113: 'END';
      JIL:=2;
      'GOTO' L92;
L107: INN:=1;
      'GOTO' L35;
L106: 'IF' AAA>0.1 'THEN' 'GOTO' L107;
L20:  OUTPUT(NFCC);
      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
      PRINT(X[I,1],0,10);
      PRINT(F[1],0,10);
'END'  POWELL64;
```

FLEPOMIN

```
'PROCEDURE' FLEPOMIN(N,X,F,EPS,FUNCT,CONV,LIMIT,H,LOADH);
'VALUE' N,EPS,LOADH,LIMIT;
'INTEGER' N,LIMIT;
'REAL' F,EPS;
'BOOLEAN' CONV,LOADH;
'ARRAY' X,H;
'PROCEDURE' FUNCT;
'BEGIN'
      'REAL' OLDF,SG,GHG,STEP,ITA,FA,FB,GA,GB,W,Z,LAMBDA;
      'INTEGER' I,J,K,COUNT;
      'ARRAY' G,S,GAMMA,SIGMA[1:N];
      ********** INSERT PROCEDURE FUNCT HERE
      'REAL' 'PROCEDURE' DOT(A,B);
      'ARRAY' A,B;
      'BEGIN'
            'INTEGER' I;
            'REAL' S;
            S:=0;
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            S:=S+A[I]*B[I];
            DOT:=S;
      'END';
      'REAL' 'PROCEDURE' UPDOT(A,B,I);
      'VALUE' I;
      'ARRAY' A,B;
      'INTEGER' I;
      'BEGIN'
            'INTEGER' J,K;
            'REAL' S;
            K:=I;
            S:=0;
            'FOR' J:=1 'STEP' 1 'UNTIL' I-1 'DO'
            'BEGIN'
                  S:=S+A[K]*B[J];
                  K:=K+N-J;
            'END';
            'FOR' J:=I 'STEP' 1 'UNTIL' N 'DO'
```

```
            S:=S+A[K+J-I]*B[J];
            UPDOT:=S;
      'END';
SET INITIAL H;
      'IF' LOADH 'THEN'
            'BEGIN'
                  K:=1;
                  'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
                  'BEGIN'
                        H[K]:=1;
                        'FOR' J:=1 'STEP' 1 'UNTIL' N-I 'DO'
                        H[K+J]:=0;
                        K:=K+N-I+1;
                  'END';
            'END';
START OF MINIMIZATION;
      CONV:='TRUE';
      STEP:=1;
      FUNCT(N,X,F,G);
      'FOR' COUNT:=1,COUNT+1 'WHILE' OLDF>F 'DO'
      'BEGIN'
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            'BEGIN'
                  SIGMA[I]:=X[I];
                  GAMMA[I]:=G[I];
                  S[I]:=-UPDOT(H,G,I);
            'END' PRESERVATION OF X,G,AND FORMATION OF S;
            FB:=F;
            GB:=DOT(G,S);
            'IF' GB 'GE' 0 'THEN' 'GOTO' EXIT;
            OLDF:=F;   ITA:=STEP;
EXTRAPOLATE:
            FA:=FB;  GA:=GB;
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            X[I]:=X[I]+ITA*S[I];
            FUNCT(N,X,F,G);
            FB:=F;  GB:=DOT(G,S);
            'IF' GB<0 'AND' FB<FA 'THEN'
            'BEGIN'
                  ITA:=4*ITA; STEP:=4*STEP;
```

```
                'GOTO' EXTRAPOLATE
            'END';
INTERPOLATE:
            'IF' ITA<0.00005 'THEN' 'GOTO' SKIP;
            Z:=3*(FA-FB)/ITA+GA+GB;
            W:='IF' Z*Z-GA*GB<0 'THEN' 0 'ELSE' SQRT(Z*Z-GA*GB);
            LAMBDA:=ITA*(1-('IF' GA+Z 'GE' 0 'THEN'
                    (GA+Z+W)/(GA+GB+2*Z)
                    'ELSE' GA/(GA+Z-W)));
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            X[I]:=X[I]-LAMBDA*S[I];
            FUNCT(N,X,F,G);
            'IF' F>FA 'OR' F>FB 'THEN'
            'BEGIN'
                  STEP:=STEP/4;
                  'IF' FB<FA 'THEN'
                  'BEGIN'
                        'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
                        X[I]:=X[I]+LAMBDA*S[I];
                        FUNCT(N,X,F,G)
                  'END'  'ELSE'
                  'BEGIN'
                        GB:=DOT(G,S);
                        'IF' GB<0 'AND' COUNT>N 'AND' STEP<8-6
                        'THEN' 'GOTO' EXIT;
                        FB:=F;  ITA:=ITA-LAMBDA;
                        'GOTO' INTERPOLATE
                  'END';
SKIP:       'END' OF SEARCH ALONG S;
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            'BEGIN'
                  SIGMA[I]:=X[I]-SIGMA[I];
                  GAMMA[I]:=G[I]-GAMMA[I];
            'END';
            SG:=DOT(SIGMA,GAMMA);
            'IF' COUNT 'GE' N 'THEN'
            'BEGIN'
                  'IF' SQRT(DOT(S,S))<EPS 'AND'
                  SQRT(DOT(SIGMA,SIGMA))<EPS 'THEN' 'GOTO' FINISH;
```

```
            'END';
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            S[I]:=UPDOT(H,GAMMA,I);
            GHG:=DOT(S,GAMMA);
            K:=1;
            'IF' SG=0 'OR' GHG=0 'THEN' 'GOTO' TEST;
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            'FOR' J:=I 'STEP' 1 'UNTIL' N 'DO'
            'BEGIN'
                  H[K]:=H[K]+SIGMA[I]*SIGMA[J]/SG-S[I]*S[J]/GHG;
                  K:=K+1
            'END' UPDATING OF H;
TEST:       'IF' COUNT>LIMIT 'THEN' 'GOTO' EXIT;
      'END' OF LOOP CONTROLLED BY COUNT;
      'GOTO' FINISH;
EXIT: CONV:='FALSE';
FINISH:
'END' OF FLEPOMIN;
```

FLEPOMIN MODIFIED

```
'PROCEDURE' FLEPOMIN(N,X,F,EPS,FUNCT,CONV,LIMIT,H,LOADH);
'VALUE' N,EPS,LOADH,LIMIT;
'INTEGER' N,LIMIT;
'REAL' F,EPS;
'BOOLEAN' CONV,LOADH;
'ARRAY' X,H;
'PROCEDURE' FUNCT;
'BEGIN'
      'REAL' OLDF,SG,GHG,STEP,ITA,FA,FB,GA,GB,W,Z,LAMBDA;
      'INTEGER' I,J,K,COUNT;
      'ARRAY' G,S,GAMMA,SIGMA[1:N];
      ********** INSERT PROCEDURE FUNCT HERE
      'REAL' 'PROCEDURE' DOT(A,B);
      'ARRAY' A,B;
      'BEGIN'
            'INTEGER' I;
            'REAL' S;
            S:=0;
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            S:=S+A[I]*B[I];
            DOT:=S;
      'END';
      'REAL' 'PROCEDURE' UPDOT(A,B,I);
      'VALUE' I;
      'ARRAY' A,B;
      'INTEGER' I;
      'BEGIN'
            'INTEGER' J,K;
            'REAL' S;
            K:=I;
            S:=0;
            'FOR' J:=1 'STEP' 1 'UNTIL' I-1 'DO'
            'BEGIN'
                  S:=S+A[K]*B[J];
                  K:=K+N-J;
            'END';
            'FOR' J:=I 'STEP' 1 'UNTIL' N 'DO'
```

```
            S:=S+A[K+J-I]*B[J];
            UPDOT:=S;
      'END';
SET INITIAL H;
      'IF' LOADH 'THEN'
            'BEGIN'
                  K:=1;
                  'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
                  'BEGIN'
                        H[K]:=1;
                        'FOR' J:=1 'STEP' 1 'UNTIL' N-I 'DO'
                        H[K+J]:=0;
                        K:=K+N-I+1;
                  'END';
            'END';
START OF MINIMIZATION;
      CONV:='TRUE';
      STEP:=1;
      FUNCT(N,X,F,G);
      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
      SIGMA[I]:=X[I];
      'FOR' COUNT:=1,COUNT+1 'WHILE' SQRT(DOT(GAMMA,GAMMA))>EPS
      'DO'
      'BEGIN'
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            'BEGIN'
                  SIGMA[I]:=X[I];
                  GAMMA[I]:=G[I];
                  S[I]:=-UPDOT(H,G,I);
            'END';
            'IF' COUNT=1 'THEN' 'GOTO' MIN;
            OLDF:=F;
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            X[I]:=X[I]+1*S[I];
            FUNCT(N,X,F,G);
            'IF' ABS((F-OLDF)/(DOT(S,GAMMA))'GE'0.00001
            'THEN' 'GOTO' FORMH;
MIN:
            FB:=F;
            GB:=DOT(G,S);
```

```
            'IF' GB 'GE' 0 'THEN' 'GOTO' EXIT;
            OLDF:=F;   ITA:=STEP;
EXTRAPOLATE:
            FA:=FB;  GA:=GB;
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            X[I]:=X[I]+ITA*S[I];
            FUNCT(N,X,F,G);
            FB:=F;  GB:=DOT(G,S);
            'IF' GB<0 'AND' FB<FA 'THEN'
            'BEGIN'
                  ITA:=4*ITA; STEP:=4*STEP;
                  'GOTO' EXTRAPOLATE
            'END';
INTERPOLATE:
            'IF' ITA<0.00005 'THEN' 'GOTO' SKIP;
            Z:=3*(FA-FB)/ITA+GA+GB;
            W:='IF' Z**'2-GA*GB<0 'THEN' 0 'ELSE' SQRT(Z**'2-GA*GB);
            LAMBDA:=ITA*(1-('IF' GA+Z 'GE' 0 'THEN'
                    (GA+Z+W)/(GA+GB+2*Z)
                    'ELSE' GA/(GA+Z-W)));
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            X[I]:=X[I]-LAMBDA*S[I];
            FUNCT(N,X,F,G);
            'IF' F>FA 'OR' F>FB 'THEN'
            'BEGIN'
                  STEP:=STEP/4;
                  'IF' FB<FA 'THEN'
                  'BEGIN'
                        'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
                        X[I]:=X[I]+LAMBDA*S[I];
                        FUNCT(N,X,F,G)
                  'END'  'ELSE'
                  'BEGIN'
                        GB:=DOT(G,S);
                        'IF' GB<0 'AND' COUNT>N 'AND' STEP<&-6
                        'THEN' 'GOTO' EXIT;
                        FB:=F;  ITA:=ITA-LAMBDA;
                        'GOTO' INTERPOLATE
                  'END';
```

```
SKIP:       'END' OF SEARCH ALONG S;
FORMH:
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            'BEGIN'
                  SIGMA[I]:=X[I]-SIGMA[I];
                  GAMMA[I]:=G[I]-GAMMA[I];
            'END';
            SG:=DOT(SIGMA,GAMMA);
            'IF' COUNT 'GE' N 'THEN'
            'BEGIN'
                  'IF' SQRT(DOT(S,S))<EPS 'AND'
                  SQRT(DOT(SIGMA,SIGMA))<EPS 'THEN' 'GOTO' FINISH;
            'END';
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            S[I]:=UPDOT(H,GAMMA,I);
            GHG:=DOT(S,GAMMA);
            K:=1;
            'IF' SG=0 'OR' GHG=0 'THEN' 'GOTO' TEST;
            'IF' SG<GHG 'THEN'
            'BEGIN'
                  'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
                  'FOR' J:=I 'STEP' 1 'UNTIL' N 'DO'
                  'BEGIN'
                        H[K]:=H[K]+SIGMA[I]*SIGMA[J]/
                              SG-S[I]*S[J]/GHG;
                        K:=K+1;
                  'END';
            'END'
            'ELSE'
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            'FOR' J:=I 'STEP' 1 'UNTIL' N 'DO'
            'BEGIN'
                  H[K]:=H[K]-(SIGMA[I]*S[J]+SIGMA[J]*S[I])
                        /SG+(1+GHG/SG)*SIGMA[I]*SIGMA[J]/SG;
                  K:=K+1;
            'END';
TEST:       'IF' COUNT>LIMIT 'THEN' 'GOTO' EXIT;
      'END' OF LOOP CONTROLLED BY COUNT;
      'GOTO' FINISH;
EXIT: CONV:='FALSE';
FINISH:
'END' OF FLEPOMIN;
```

APPENDIX II

LISTING OF PROGRAMS FOR CONSTRAINED OPTIMIZATION PROBLEMS

POWCON

```
'PROCEDURE'POWCON(X,EPS,RATIO,M);
'ARRAY' X;
'REAL' EPS,RATIO;
'INTEGER' M;
'BEGIN'
      'INTEGER' I;
      'REAL' MAXPSI,MAXL;
      'BOOLEAN' SWUP;
      'ARRAY' THETABAR[1:M];
      MAXPSI:=&10;
      SWUP:=FALSE;
AGAIN:MAXL:=MAXPSI;
      MAXPSI:=-&-8;
      **********CALL MINIMIZATION PROCEDURE HERE
      PSICAL(PSI,X);
      SECOND:='TRUE';
      'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
      'IF' MAXPSI<ABS(PSI[I]) 'THEN' MAXPSI:=ABS(PSI[I]);
      'IF' MAXPSI<EPS 'THEN' 'GOTO' EXIT;
      'IF' MAXPSI 'GE' MAXL 'THEN' MAXPSI:=MAXL 'ELSE'
      'GOTO' THINC1;
      'IF' SWUP 'THEN'
      'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
      THETA[I]:=THETABAR[I];
SIGINC;'FOR' I:=1 'STEP' 1 'UNTIL'M 'DO'
      'IF' ABS(PSI[I]) 'GE' MAXL/RATIO 'THEN'
      'BEGIN'
            SIGMA[I]:=10*SIGMA[I];
            THETA[I]:=0.1*THETA[I];
      'END';
      SWUP:='FALSE';
      'GOTO' AGAIN;
THINC1:
      'IF' 'NOT' SWUP 'THEN' 'GOTO' THINC2;
      'IF' MAXPSI<MAXL/RATIO 'THEN' 'GOTO' SIGINC;
THINC2:
```

```
        'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
        'BEGIN'
              THETABAR[I]:=THETA[I];
              THETA[I]:=THETA[I]+PSI[I];
        'END';
        SWUP:='TRUE';
        'GOTO' AGAIN;
EXIT:   'END' POWCON;
```

OPTKOV

```
'PROCEDURE' OPTKOV(X,XK,XKT,XKM,STEP,F,EPS);
'REAL' XK,XKT,XKM,STEP,EPS,F;
'ARRAY' X;
'BEGIN'
COMMENT  PHASE 1;
      'REAL' BL,BU;
      XK:= ****OBJECTIVE FUNCTION;
RIP:  XK:=XK-STEP;
      ************CALL THE MINIMIZATION PROCEDURE HERE
      'IF' SQRT(F)<EPS 'THEN'
      'BEGIN'
            STEP:=2*STEP;
            'GOTO' RIP
      'END' 'ELSE'
      BU:=XK+STEP;
      BL:=XK;
COMMENT PHASE 2;
PARAMETER;
      XKM:=XK+SQRT(F);
      XKT:=XK+F/SQRT(F-T);
      BL:=XKM;
      'IF' XKT<BU 'THEN'
      XK:=XKT 'ELSE'
      XK:=XKM;
MIN:  ***********CALL THE MINIMIZATION PROCEDURE HERE
      'IF' SQRT(F) 'GE' EPS 'THEN''GOTO' PARAMETER
      'ELSE' BU:=XK;
      'IF' ABS(BU-BL)<EPS    'THEN' 'GOTO' EXIT
      'ELSE'  XK:=BL;
      'GOTO' MIN;
EXIT:
'END' OF OPTKOV;
```

SUMT

```
'PROCEDURE' SUMT(X,EPS,RATIO),
'ARRAY'X;
'REAL' EPS;
'INTEGER' RATIO;
'BEGIN'
START;
      ***********CALL MINIMIZATION PROCEDURE HERE
      'IF' ABS(F-D)>EPS 'THEN'
      'BEGIN'
            R:=R/RATIO;
             'GOTO' START;
      'END';
'END' SUMT;
```

AS IT HAS BEEN MENTIONED EARLIER ON IN THE DESCRIPTION OF THE ALGORITHM, THE PTOCEDURE FLEPOMIN HAD TO BE MODIFIED. THIS MODIFICATION IS AS FOLLOWS:

```
EXTRAPOLATE:
      FA:=FB;
      GA:=GB;
REPEAT;
      'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
      X[I]:=X[I]+ITA*S[I];
      FUNCT(N,X,F,G);
      'FOR' I:=1 'STEP' 1 'UNTIL' M 'DO'
      'IF' C[I] 'LT' 0.0 'THEN'
      'BEGIN'
            'FOR' I:=1 'STEP' 1 'UNTIL' N 'DO'
            X[I]:=X[I]-ITA*S[I];
            ITA:=ITA/DD;
            'GOTO' REPEAT;
      'END';
```