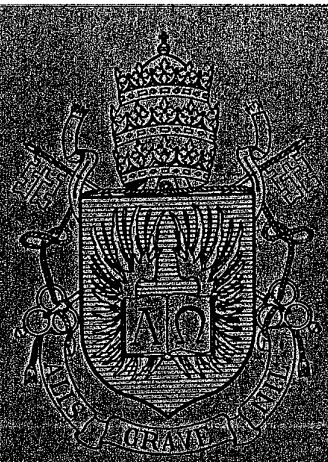


PUC

Series: Monographs in Computer Science
and Computer Applications

Nº 3/69

FORMULA MANIPULATION IN THE FLETCHER AND POWELL OPTIMIZATION METHOD

by

ANTÔNIO LUZ FURTADO - FIRMO FREIRE

Computer Science Department - Rio Datacenter

CENTRO TECNICO CIENTIFICO
Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 209 — ZC-20
Rio de Janeiro — Brasil

FORMULA MANIPULATION IN THE FLETCHER AND POWELL OPTIMIZATION METHOD

ANTÔNIO LUZ FURTADO - FIRMO FREIRE
COMPUTER SCIENCE DEPARTMENT
PUC - RIO DE JANEIRO

FORMULA MANIPULATION IN THE FLETCHER AND POWELL OPTIMIZATION METHOD.

1. Introduction

The method developed by Fletcher and Powell [1] finds the optimum of a function f(x).

A particular problem involving function minimization is curve-fitting one of the most widely known is least-squares, consisting of the minimization of the function $\Sigma\left[y - f(x)\right]^2$.

The present work is concerned with general least - squares curve -fitting using the Fletcher and Powell method.

The method requires an evaluation of partial derivatives at each step.Two possibilities are usually considered:

a. to find the expression of the derivatives by hand and use then in function subprograms;

b. to employ finite differences to evaluate the derivatives numerically.

The first solution imposes upon the user the necessity of doing work that is both time - consuming and subject to manipulative errors.

The second solution involves approximating a function by a polynomial thus introducing truncation errors.

A third solution is proposed in this paper. Methods of formula manipulation on a computer including formal differentiation have been used to automate the first method.

The resulting derivatives have been either evaluated interpretively or compiled into machine language and then executed under control of the optimization program.

Since formal differentiation and compilation are performed only once there is very little increase in the lenght of time required to execution.

An implementation of these techniques was written in FORTRAN IV,MAP, and the list - processing system SLIP [2].

An IBM 7044 (32 k) of the " Pontificia Universidade Católica " (Rio de Janeiro - Brasil) was used to run a number of examples.

2. The Formula Manipulation Techniques

2.1 The Differentiation and Simplification Algorithms

Consider any arithmetic expression such as a + b + c . d. Our first step would be to rewrite it in prefixed Polish notation [3] as (+ a (+ b (* c d))).

Notice that each sub - expression contains only one operator (we shall call it p) and two operands (q and r; in case of just one operand, the second will be ϕ). Rules [4] for differentiating a sub - expression, giving the results in the same notation, and representing as q' and r' the derivative with respect to a general variable x, are:

p	derivative
+,-	($\pm$ q' r')
*	(+ (* q r') (* r q'))
/	(- (/ q' r) (/ (* q r') (** r 2)))
**	(+ (* (* r (** q (- r 1))) q')
	(* (* (log q ϕ) (** q r)) r')
sin	(* (cos q ϕ) , q')
cos	(* (- 0 (sin q ϕ)) , q')
atan	(/ q' (+ 1 (** q 2)))
log	(/ q' q)
exp	(* (exp q ϕ) q')
sqrt	(/ q' (* 2 (** q 0.5)))

Unary minus may be either assimilated to 0 - q or separately specified. The meaning of q' (and r'), is

- if q is an element then if q = x then 1, else 0;
- if q is a sub-expression the rules above must be applied

recursively.

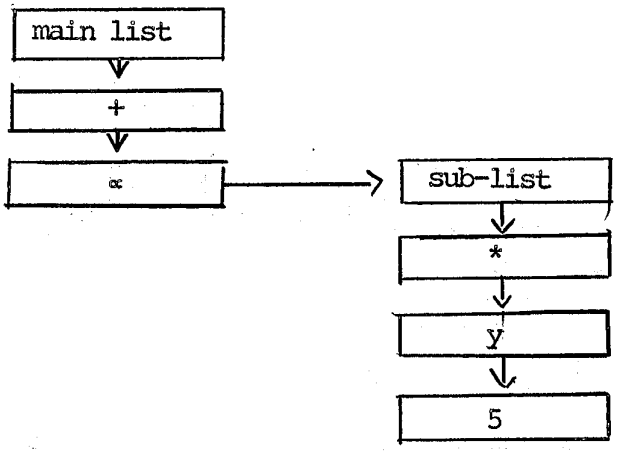
Indeed the reasons to use this notation are: to organize the whole expression and its successive sub-expressions in the same three-fold way thus allwing recursion, and to organize the results alike thus allowing n^{th} order differentiation.

The simplification process may tale place as each resulting sub-expression is built. Some of the possibilities are:

sub-expression	simplification rule
(+ q r)	q = 0 →r
	r = 0 →q
	q = r = 1 →2
	q = r ≠ 1 →(* 2 q)
(- q r)	r = 0 →q
	q = r →0
	r = 1 →q - 1
(* q r)	q = 0 →0
	r = 0 →0
	q = 1 →r
	r = 1 →q
	q = r →(** q 2)
(/ q r)	q = 0 →0
	r = 1 →q
(** q r)	q = 1 →1
	r = 0 →1
	r = 1 →q

2.2 List Structures

An expression like (+ x (* y 5)) would be represented as:



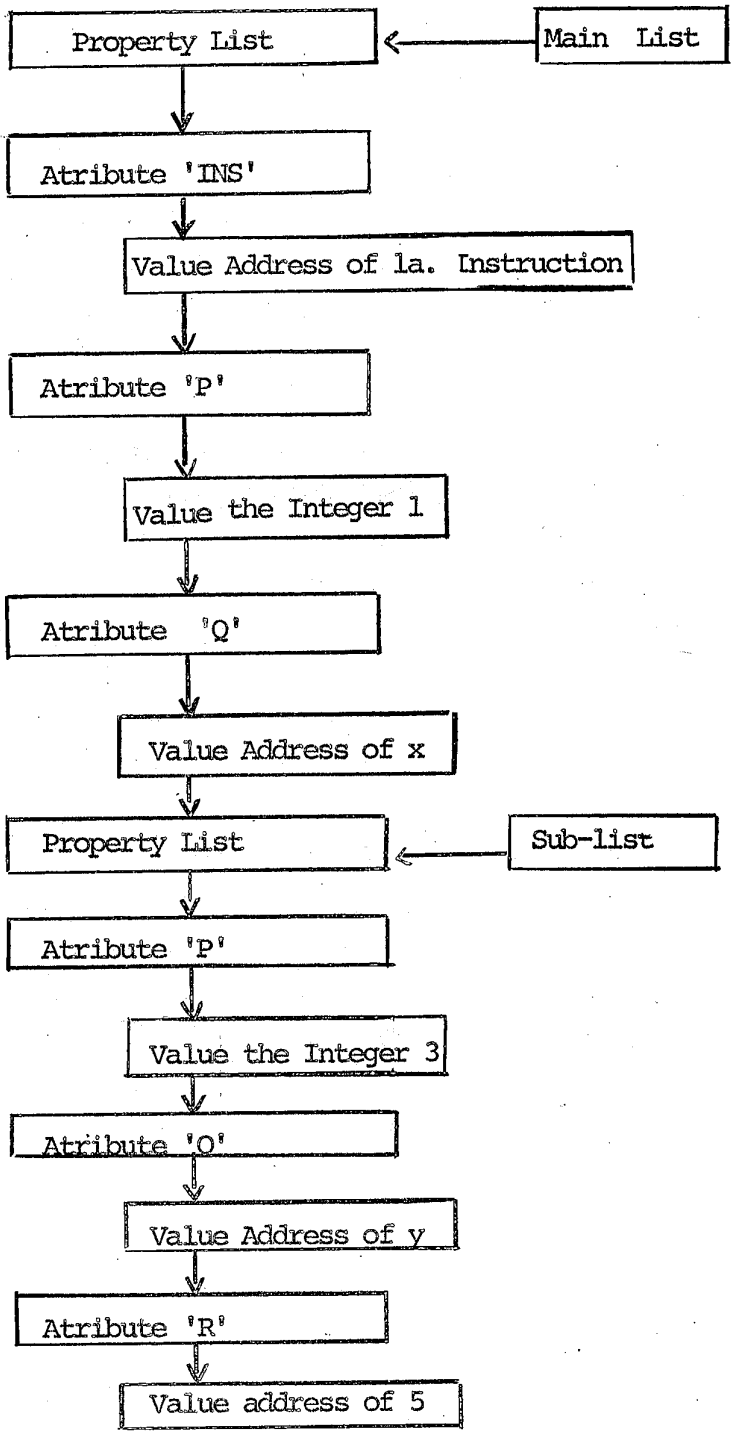
To provide more information about the expression we associate to it a property (or attribute - value)list . This device is available in most list-processing languages, and we use it

a. to prepare for directed evaluation, supposing the expression has been compiled;

b. to prepare for simple interpretative evaluation;

c. when the expression is an $n^{\underline{th}}$ derivative, to provide access to the $1^{\underline{st}}$ to $n - 1^{\underline{th}}$ derivative (this is intended for other applications for which a Taylor series is to be evaluated).

We give the property lists for the expression above and its sub-expression:


Property List
Main List
Atribute 'INS'
Value Address of 1a. Instruction
Atribute 'P'
Value the Integer 1
Atribute 'Q'
Value Address of x
Property List
Sub-list
Atribute 'P'
Value the Integer 3
Atribute 'O'
Value Address of y
Atribute 'R'
Value address of 5

2.3 Remarks on Programming Aspects

The expressions are not given as function sub-programs but rather as data, though the standard FORTRAN notation was employed.
Any fucntion, besides the six indicated above, could be added to the differentiation and simplification algorithms. Our solution was to call a routine (that is otherwise a dummy) containing the appropriate rules whenever none of those six functions is recognized. Another solution was to give the model of the derivative as data.
Interpretive evaluation uses the machine address of defined constants and variables to fetch their current numerical values. The integers, corresponding to operators, given in the property lists, make it easy to branch to the operation to be performed. Interpretive evaluation is recursive.
Compilation is done by OR ing the instruction codes with machine addresses and storing in an array; entry points to SIN, COS, etc. are known through non-executable calls to these routines. Direct evaluations is accomplished through a transfer and store location (TSL) instruction, thus providing the return address.
An interesting problem is to make any variable v in a program correspond to a variable with the same name appearing in an expression (that is given as data), at execution time.
A routine to put resulting expressions in the infixed form is necessary for readable output.
As to the Fletcher and Powell application, we have chosen to differentiate only $f(x, c_1,\ldots,c_n)$ with respect to $c_1,\ldots,c_n$; the derivatives $f_k\ (x, c_1,\ldots,c_n)$ were evaluated, and then, for each c_k

$$-2 \sum_i \left[y_i - f\ (x_i, c_1,\ldots,c_n) \right] \cdot f_k (x_i, c_1,\ldots,c_n)$$

(that is the full expression for the derivative of the function $\Sigma \left[y - f(x,c_1,\ldots,c_n) \right]^2$ to be minimized) is computed by standard FORTRAN statements.

FLETCHER AND POWELL OPTIMIZATION METHOD

The optimization method described by Fletcher and Powell in 1963 finds the local minimum of a function $F(x)$, on an unrestricted space R^n. The process as it was originaly described, requires that the gradient of the function being minimized,

$$Fx_i = \partial F(\underline{x}) / \partial Xi ,$$

can be obtained for every $\underline{x} \in R^n$.

This algorithm has quadratic convergence taking advantage of the fact that near the optimum the second order terms of the Taylor Series expansion of $F(\underline{x})$ predominate. For this reason, in this region, $F(\underline{x})$ has quadratic behavior.

4.1 Description of the Method

Taking a quadratic form

$$F(\underline{x}) = F(\underline{xo}) + \underline{g}^t \underline{x} + 1/2\, \underline{x}^t F\, xx\underline{x} \qquad (1)$$

where

$$(2) \quad \underline{x} = \begin{Bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{Bmatrix} , \qquad g_i = \frac{\partial F(\underline{x})}{\partial x_i} \quad (i = 1, 2, \ldots, n) , \qquad (3)$$

$$Fx_i\, xj = Fxj\, x_i = \frac{\partial^2 F}{\partial x_i \partial xj} \quad (i = 1, 2, \ldots, n;\ j = 1, 2, \ldots, n). \qquad (4)$$

Diferentiating (1)

$$\underline{Fx} = \underline{g}\ Fxx\ \underline{x} \quad (5)$$

Suposing that Fxx is known and Fx has been calculated, the solution to (5) is

$$\underline{x} = Fxx^{-1} - \underline{g}\ ) \quad (6)$$

In the minimum, x*, we have that $Fx = \underline{0}$

so

$$\underline{x}^* = - Fxx^{-1}\ \underline{g} \quad (7)$$

Taking (7) from (6)

$$\underline{x}^* - \underline{x} = \Delta\underline{x} = - Fxx^{-1}\ \underline{Fx} \quad (8),$$

from which the displacement, Δx, leading to the minimum of $F(\underline{x})$ can be obtained.

In this method Fxx^{-1} is not calculated directily. On the begining it is chosen to be any simetric, positive definite matrix, H. During the process, this initial matrix is modified in such way that it conver ges to Fxx^{-1}.

The search for the minimum is then made in the direction

$$\underline{\sigma} = -H\underline{Fx} \quad (9)$$

Initially, if a good aproximation for Fxx^{-1} is not known, H is taken to be the identity matrix I. The first direction of search is then the gradient direction.

Suposing that $\underline{x}^i$ is the starting point with gradient $\underline{Fx}^i$ and matrix H^i, an iteration consists on:

I) taking $\sigma^i = H^i\ Fx^i$ (10)

II) obtaining α^i that minimises $F(\underline{x} + \lambda\underline{\sigma}^i)$ (11)

III) Taking $\delta^i = \alpha^i\ \underline{\sigma}^i$ (12)

IV) Taking $\underline{x}^{i+1} = x^i + \delta^i$ (13)

V) Obtaining $F(x^{I+1})$ and $F^{i+1}x$ (14)

VI) obtaining $y^i = Fx^{i+1} - Fx^i$ (15)

VII) obtaining $H^{i+1} = H^i + A^i + B^i$ (16)

where $$A^i = \frac{\delta^i \delta^{Ti}}{\delta^{Ti} y^i} \qquad (17)$$

$$B^i = - \frac{H^i y^{Ti} H^i}{y^{Ti} H^i y^i} \qquad (18)$$

VII) taking i = i + 1 and reapeating the cycle until the minimum achieved. During step (V) it is important to check if the sign of $F\underline{x}^{i+1}$ is negative If it is positive, indicating a maximum in this direction, the method should be reestarted by taking H · I.

The process of fiding the minimum along a line is not important, and any method (quadratic or cubic interpolation, Fibonacci) can be used.

Fo a complete discussion of the method the reader is referred to [1] and [5].

4. Some examples

4.1 A third - dregree polynomial

$$f(x) = c_1 x^3 + c_2 x^2 + c_3 x + c_4$$

and 50 points are given, being taken from a sine in the interval [0.1256, 6.28]; error is simply indicated by $\Sigma[y - f(x)]^2$.

iteration	error	gradient
0	0.71057871E06	-0.13793703E13
1	0.93553701E03	-0.20710005E07
2	0.26828310E01	-0.18342468E02
3	0.28229170E01	-0.11064342E02
4	0.22096718E00	-0.86280064E-12

execution (IBM - 7044) - 11 seconds.

4.2 A Fourier Series

$$f(x) = c_1 + c_2 \cos(x) + c_3 \sin(x) + c_4 \cos(2x) + c_5 \sin(2x) + c_6 \cos(3x) + c_7 \sin(3x)$$

and 13 points are arbitrarily given:
(0.,0.), (0.5236,1.) , (1.0472,4.) , (1.5708,5.),
(2.0944,4.), (2.618,3.),(3.1416,3.5),
(3.6652,3.5),(4.1888,3.),(4.7124,1),
(5.236,0.5),(5.7596,0.2),(6.2832,0.)

iteration	error	gradient
0	0.12259356E03	-0.38771982E04
1	0.22949982E01	-0.56263757E02
2	0.55373018E00	-0.76470583E01
3	0.34009569E00	-0.44165643E-11

parameters

	initial guess	final values
c1	0.10000000	0.23950854E01
c2	0.10000000	-0.15498134E01
c3	0.10000000	0.13411897E01
c4	0.10000000	-0.72648882E00
c5	0.10000000	0.54848364E00
c6	0.10000000	-0.15982872E00
c7	0.10000000	-0.61666735E00

execution - 17 seconds

4.3 A logistic Curve (Verhulst's curve)

$f(x) = c_1 / (1 + c_2 . e^{c3x})$ and 16 points are given, being taken from a logistic curve; this was an attempt to an "exact" fitting well confirmed by the final error.

iteration	error	gradient
0	0.21018392E07	-0.66830117E14
1	0.23161255E06	-0.96784435E12
2	0.13359862E06	-0.16247006E07
3	0.88821628E05	-0.30176431E05
4	0.22509569E05	-0.13335278E05
5	0.29605248E04	-0.57720680E04
6	0.35425489E02	-0.44027244E02
7	0.53447918E00	-0.78347632E00
8	0.16417862E-02	-0.31300855E-02
9	0.16298145E-07	-0.11570152E-07

parameters		
	initial guess	final results
c1	0.30000000E04	0.28976811E04
c2	0.10000000E01	0.16533642E01
c3	0.50000000E00	0.37292729E00

execution - 14 seconds

4.4 A curve with increasing amplitude and period

$$f(x) = \sin\left(c_1 \cdot e^{c2x} \cdot x\right) \cdot e^{c3x}$$

and 26 points are given:
(0 ,0),(0.1,0.2),(0.2,0.),(0.3,-0.4),(0.4,-1.),
(0.5,-0.6),(0.6,0.),(0.7,0.8),(0.8,1.8),
(0.9,3.),(1,2.2),(1.1,1.2),(1.2,0.),
(1.3,-1.4),(1.4,-3.),(1.5,-4.8),(1.6,-6.8),
(1.7,-5.4),(1.8,-3.8),(1.9,-2.),(2.,0.),
(2.1,2.2),(2.2,4.6),(2.3,7.2),(2.4,10.),
(2.5,13.)

iteration	error	gradient
0	0.76787446E03	-0.43668890E07
1	0.36010441E03	-0.85459424E04
2	0.35875788E03	-0.73709711E02
3	0.19162920E03	-0.64111556E02
4	0.71163198E02	-0.59923968E02
5	0.46289303E02	-0.30989299E02
6	0.22226243E02	-0.20746101E01
7	0.17774374E02	-0.93962388E00
8	0.17417844E02	-0.53808595E00
9	0.16913818E02	-0.70761245E-01
10	0.16872360E02	-0.15951866E-03
11	0.16872282E02	-0.78429112E-07

parameters	initial guess	final results
c1	0.20000000E02	0.28529033E02
c2	-0.10000000E01	-0.14383141E01
c3	0.50000000E00	0.10474954E01

execution - 39 seconds

REFERENCES

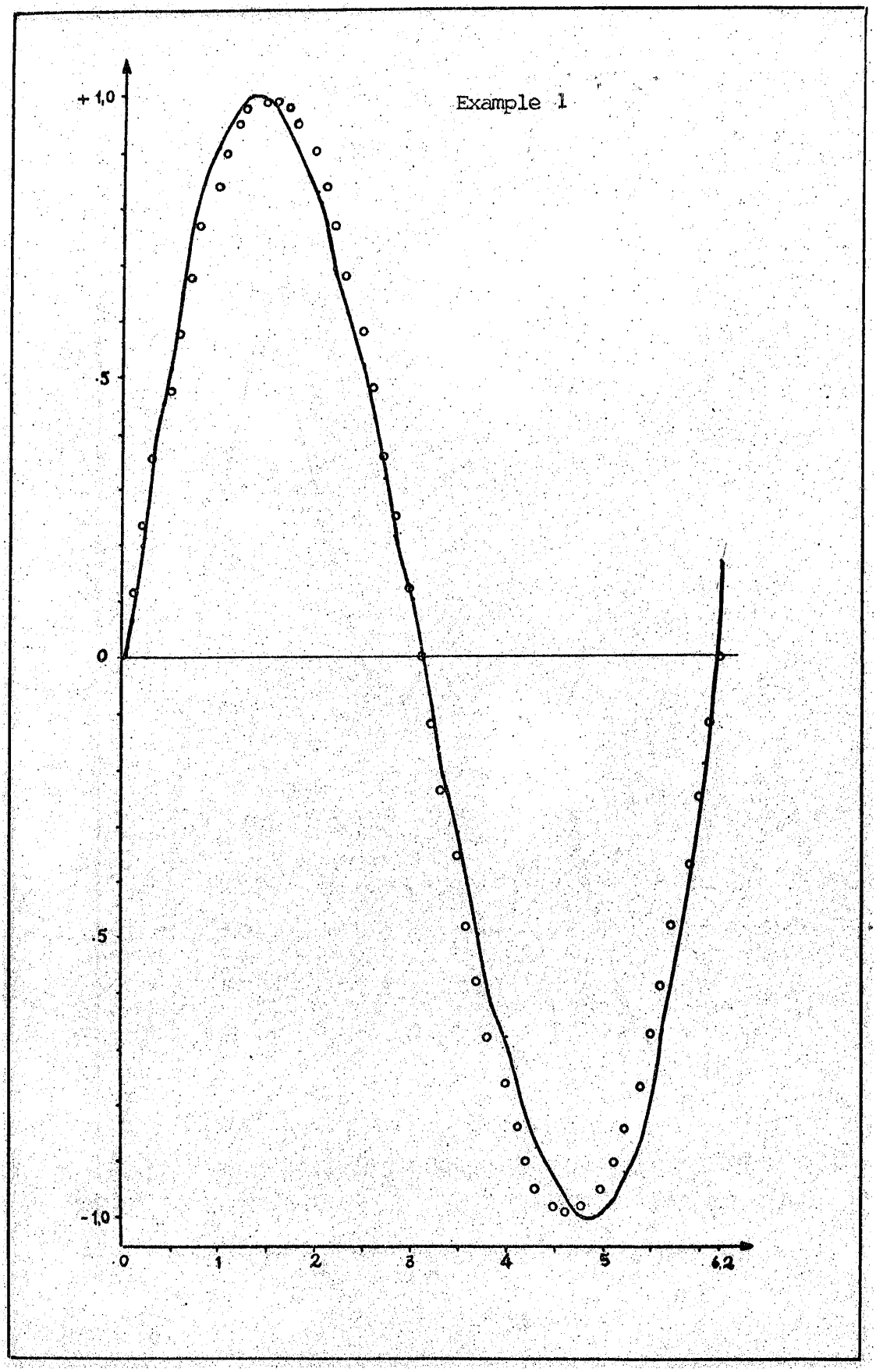
1. Fletcher,R. and Powell,M.J.D.
"A Rapifly Convergent Descent Method for Minimization"
The Computer Journal, vol. 6 - p. 163-168

2. Weizenbaum,J.
"Symmetric List Processor"
Comm. ACM, p. 534 - September, 1963

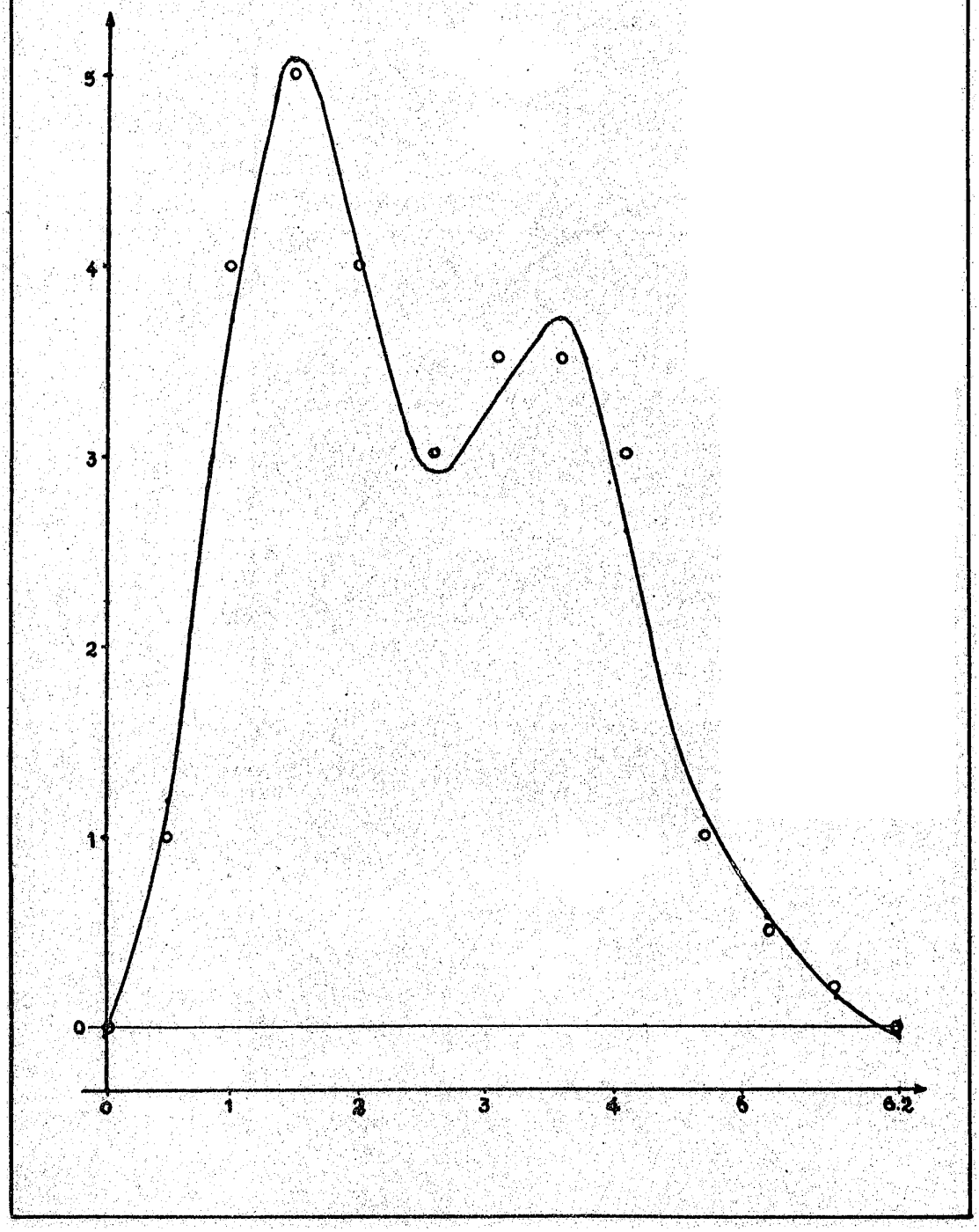
3. Graham, R.M.
"Bounded Context Translation"
Programming Systems and Languages
edited by Rosen,S. - Mc Graw Hill - 1967

4. Barron,D.W. and Strachey,C.
"Programming"
Advances in Programming & Nor. Numerical Computation, edited by Fox,L.
Pergamon - 1966

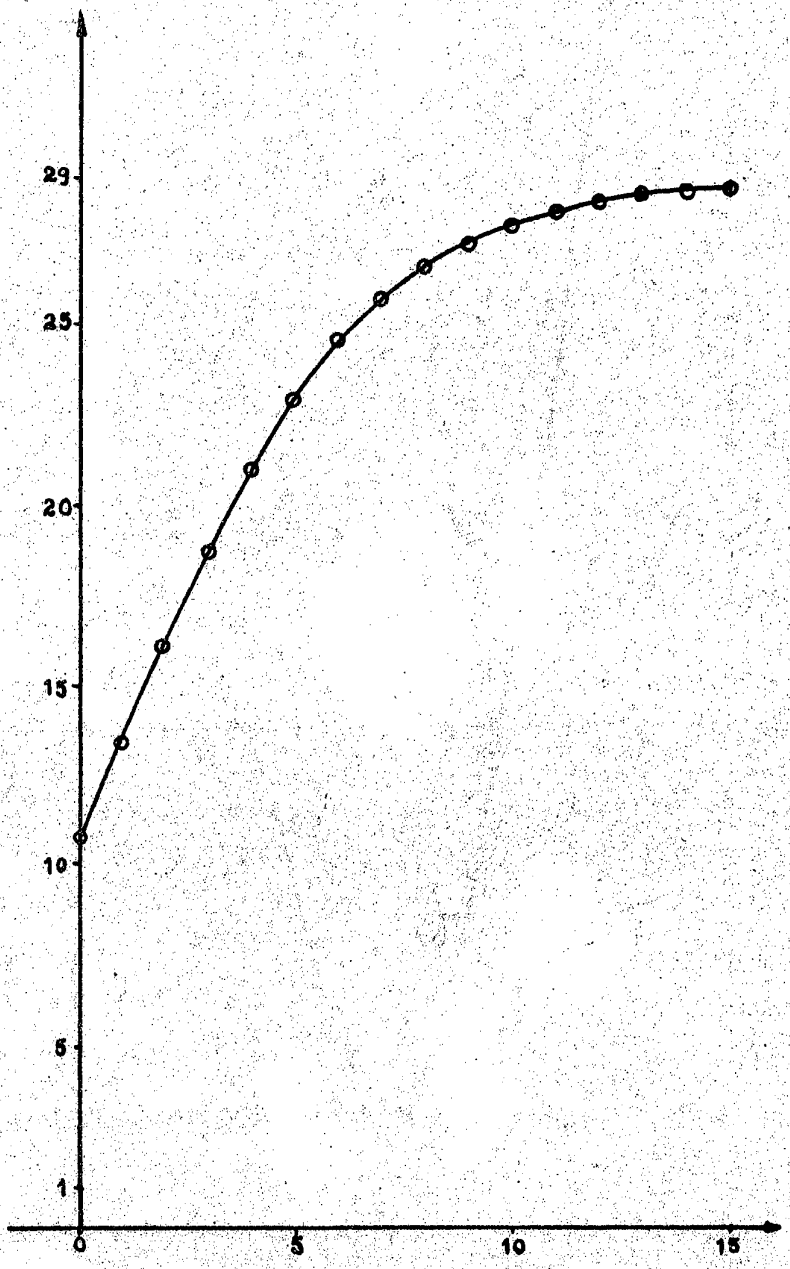
5. Wilde,D.J. and Beightler,C.S.
"Foundations of Optimization" - Prentice Hall 1967.

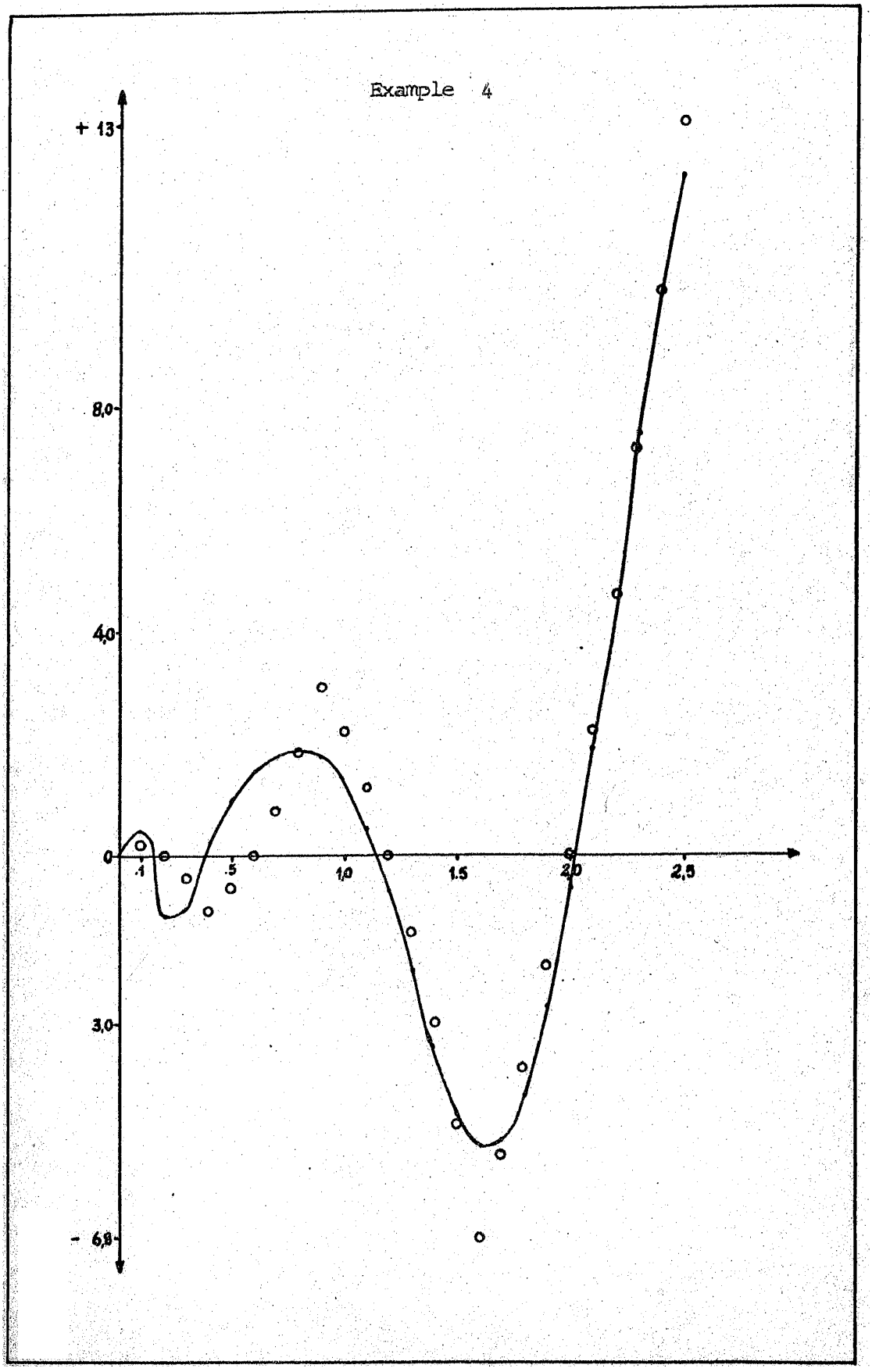

Example 1
+1,0
.5
0
.5
-1,0
0
1
2
3
4
5
6,2

Example 2



Example 3




Example 4
+ 13
8,0
4,0
0
3,0
- 6,8
.1
.5
1,0
1,5
2,0
2,5