stichting
mathematisch
centrum

MC

AFDELING NUMERIEKE WISKUNDE NW 25/75 DECEMBER

J.C.P. BUS

A COMPARATIVE STUDY OF PROGRAMS FOR SOLVING NONLINEAR EQUATIONS

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

AMS(MOS) subject classification scheme (1970): 65H10

A comparative study of programs for solving nonlinear equations

by

J.C.P. Bus

ABSTRACT

In this report we propose a method for comparing the efficiency and reliability of programs for solving systems of nonlinear equations. We use this method for comparing a great number of existing programs. The results of these comparisons are given in such a way that it is easy for the user to decide which program he should choose for solving a given system of nonlinear equations.

KEY WORDS AND PHRASES: *Systems of nonlinear equations, comparison of efficiency and reliability of programs.*

CONTENTS

INTRODUCTION

In recent years, the testing of numerical software becomes more and more important. There are several reasons for this development. One is the creation of large user libraries of numerical programs (IMSL [32], NAG [38], NUMAL [39] etc.), where the need for choosing the programs to be included, makes testing very urgent. Another reason is the confusing variety of programs in some fields of numerical mathematics, which makes it impossible for the unsophisticated user of numerical software to choose the right program for solving his problem. A lot of papers are devoted to the testing of software (HAGUE et al.[29], HILLSTRÖM [30], LOOTSMA [34], EINARSSON [23], HULL [31] etc.). However, many of the ideas suggested in the various papers are controversial or contradict each other. Therefore, we want to point out clearly the purposes of this report. In our opinion, the process of selecting useful numerical software consists of three stages:

- analysis of the theoretical properties of the underlying algorithms;
- analysis of the practical performance of the algorithms;
- analysis of programs.

We will elucidate these three stages.

1. *Analysis of theoretical properties*

The algorithms should have a sound mathematical basis. It should be clear on what conditions convergence is guaranteed.

2. *Analysis of practical performance*

We are interested in two desirable properties.

a. The work that has to be done to solve a problem. We say that an algorithm is more *efficient* than another for solving a problem, when the work that has to be done for solving this problem with this algorithm is less than for solving with the other algorithm.

b. The capability of an algorithm to compute accurate answers to severe problems or to compute answers at all to such problems. This is called *reliability* or *robustness*.

One should realize that the most efficient algorithm for solving relatively easy problems may frequently fail in solving severe problems. Moreover, an algorithm that is capable of solving severe problems will usually

not be efficient for solving easy problems. For instance, evaluating a function for all representable numbers on a computer is clearly a robust method for finding a solution of an equation in one variable, however, using interpolation will be far more efficient in most cases but may fail sometimes.

In most practical cases, the user does not know in advance whether his problem is relatively easy to solve. Hence, he wants to choose the algorithm that has both the highest probability that it solves his problem and is the most efficient algorithm for solving it. However, the arguments above indicate that these wishes are rather contradictory in most cases. Hence, the user has to choose the appropriate algorithm by a method of trial and error. The goal of this report is to tell him which algorithm is the best to try first and which one when the first is failing and so on.

In performing an analysis of the relative efficiency and reliability of some algorithm one should have some measure for these properties. For many non-iterative algorithms it is easy to count the number of basic arithmetical operations (+,-,×,/) and the number of evaluations of the functions involved, if there are any. This gives a very practical measure of the efficiency of such algorithms. Furthermore, a theoretical analysis of non-iterative algorithms will usually give enough information about the reliability. However, for iterative algorithms these problems are far more complicated. Although it is possible to count the number of arithmetical operations as well as the number of function evaluations performed at each iteration step, provided that there are no iterative subprocesses, we do not know the number of iteration steps needed to obtain a certain result. Therefore, we have to make programs which implement the iterative algorithms in order to be able to get this number for a representative set of testproblems. Clearly, the reliability of the algorithm is measured by just counting the number of failures while solving the problems of the given set. By measuring the efficiency, however, we feel that we should not take into account the failures of an algorithm, since we know that it may fail in solving relatively difficult problems. Therefore, it is necessary to create a set of relatively easy testproblems in a sense that should be specified clearly. This set should be used for comparing the efficiency of all algorithms. Obviously, the notions efficiency and reliability as used in this report are dependent on the sets of testproblems chosen. Selection of these sets should

be based on thorough theoretical and practical arguments. We tried to do so, but we do realize that it is still far from being ideal.

Finally, we want to emphasize that a measure for the efficiency of an algorithm should be as independent as possible of the environment in which the algorithm is used. Therefore, computation time is a very bad measure, since it depends on the running system of the computer (usually swapping time is added to normal computation time), on the hardware (the ratio of the time needed for addition and for multiplication varies from one computer to another), on the compiler used (see PARLETT & WANG [42]) and on many other things which are difficult to define precisely.

3. *Analysis of programs*

Examples of properties that the program should satisfy are:

a. the program should be well-structured (built up from independent modules), so that error detection becomes easy;
b. stopping criteria should be such that the required results can be guaranteed (if at all possible); some kind of error messages should be given when the algorithm breaks down;
c. machine-dependent quantities should be avoided if at all possible, otherwise they should be defined explicitly and the computation should be such that under- and overflow is avoided.

In this report we will be concerned with the first two stages with respect to the problem of solving systems of nonlinear equations, although the first stage is mainly restricted to giving relevant literature.

In section 1 the problem is defined. In section 2 some theoretical background is given. Particularly, Newton-like algorithms are briefly discussed. In section 3 we describe the methods known and mention relevant literature about convergence and stability. In section 4 we list the programs which are chosen for testing. We did only choose those programs of which an implementation in ALGOL 60 or FORTRAN is readily available from the literature. We did not implement algorithms by ourselves since one of our purposes is to present the unsophisticated user a guide for choosing an existing program for solving his problems.

In section 5 we define the testproblems and we propose a classification of these problems. In section 6 we summarize the results of section 1 to 5

in some rules of thumb for the user. We give him the tools which should enable him to classify his problem. The main part of this report, at least quantitatively, consists of section 7 where the numerical experiments are described and where the results are given in tables and diagrams. In section 8 conclusions about the efficiency and reliability of the various programs are given. Here we give the user the information that is necessary to make a reasonable decision about which program he should choose for his problem.

Finally, the unsophisticated user is advised to examine his problem in the way that is advised in section 6, subsequently, to choose the program with the help of the conclusions given in section 8, and finally, to read the description of the program given in section 4 and to perform the modifications proposed there. Doing so, he will not be involved with theoretical considerations and yet he will take advantage of the results of this report as much as possible.

1. STATEMENT OF THE PROBLEM

We consider the problem of solving a system of nonlinear equations. Let F denote an n-dimensional continuous (nonlinear) function of n variables, defined on some region $D \in \mathbb{R}^n$:

$$F: D \subset \mathbb{R}^n \to \mathbb{R}^n. \tag{1.1}$$

Then we want to compute some vector $z \in D$, such that

$$F(z) = 0 \in \mathbb{R}^n. \tag{1.2}$$

In numerical analysis, a wide variety of problems may be formulated in such a way that the solution of a system of nonlinear equations is required for solving these problems. For instance, solving a two point boundary value problem

$$u'' = f(t,u), \qquad 0 \le t \le 1,$$
$$u(0) = \alpha, \quad u(1) = \beta,$$

with a finite difference or finite element method gives rise to a system of

nonlinear equations if f(t,u) is nonlinear in u. Other problems, for which solving may require the solution of a system of nonlinear equations are elliptic boundary value problems, integral equations or two-dimensional variational problems (see ORTEGA & RHEINBOLDT [40]).

Algorithms for solving nonlinear problems are usually iterative. I.e., given any initial approximation x_0 to z, the algorithm generates a series of approximations $\{x_i\}_{i=1}^{\infty}$ to z, such that

$$\lim_{i\to\infty} x_i = z.$$

It is very obvious that the choice of the initial guess may highly affect the convergence of the sequence $\{x_i\}$ to the solution vector z. Therefore, we give a more precise definition of the problem considered

(1.3) given $F: D \subset \mathbb{R}^n \to \mathbb{R}^n$ and $x_0 \in D$;
calculate $z \in D$, such that $F(z) = 0$.

We denote this problem by

(1.4) $[F(x) = 0; x_0; D]$.

In this report we compare programs for solving problem (1.3).

2. THEORETICAL BACKGROUND

2.1. *General theoretical considerations*

An iterative m-step method which uses the function and its first derivative for solving problem (1.3) may generally be defined by:

given $x_0,\ldots,x_{m-1}$,
calculate for $k = m-1,m,m+1,\ldots$

$$x_{k+1} = \Phi_m(x_k,\ldots,x_{k-m+1},F(x_k),\ldots,F(x_{k-m+1}),J(x_k),\ldots,J(x_{k-m+1})), \tag{2.1.1}$$

where J(x) is the so-called Jacobian matrix of partial derivatives. Speci-

fic examples are

$$x_{k+1} = \phi(x_k) = x_k - [J(x_k)]^{-1}F(x_k) \qquad \text{(Newton's method)}, \tag{2.1.2}$$

$$x_{k+1} = \psi(x_k) = x_k - [M(x_k)]^{-1}F(x_k), \tag{2.1.3}$$

where $M(x_k)$ will usually be some approximation to $J(x_k)$. Most methods considered in this report, including Newton's method, can be given in the form (2.1.3). Therefore, we will pay some special attention to these so-called Newton-like methods.

A theoretical analysis of Newton's method, which is based on the Newton-Kantorovich theorem can be found in the literature (e.g. ORTEGA & RHEINBOLDT [40], COLLATZ [16], RALL [45]). For this method, one can prove that the error in $\phi(x)$ as an approximation to the solution z satisfies:

$$\|\phi(x)-z\| \leq S(x,z)\|x-z\|^2, \tag{2.1.4}$$

where $S(x,z)$ depends on $\|[J(x)]^{-1}\|$ and the norm of the second derivative of the function in some region containing x and z (COLLATZ [16], BUS [14]). Hence, provided $S(x,z)$ is bounded (i.e. $J(x)$ is nonsingular and the second derivative is bounded) the asymptotic order of convergence of Newton's method is quadratic.

However, the use of iteration formula (2.1.3) leads to the more complicated bound for the error in $\psi(x)$:

$$\begin{aligned}\|\psi(x)-z\| \leq\ & c_1(x)\|[J(x)]^{-1}\|\|x-z\| + \\ & + (c_1(x)\|[J(x)]^{-1}\| + 1)S(x,z)\|x-z\|^2,\end{aligned} \tag{2.1.5}$$

where $c_1(x)$ is a measure for the error in $M(x)$ as an approximation to $J(x)$ (BUS [14]). It is obvious from (2.1.5) that superlinear convergence of the method given by (2.1.3) can only be guaranteed if

$$c_1(x) = O(\|x-z\|), \qquad \text{for } x \to z. \tag{2.1.6}$$

For somewhat different treatments of the convergence analysis of methods as given by (2.1.3) we refer to ORTEGA & RHEINBOLDT [40] or BOGGS & DENNIS [1].

2.2. *Numerical aspects*

Using a method as given by (2.1.3) on a computer, we are confronted with two kinds of problems due to the finite word length of a computer. The first one is that in computing $M(x_k)$ as an approximation to $J(x_k)$, the best we can obtain anyhow is a relative error which is about the same as the precision of arithmetic. Hence (2.1.6) cannot be satisfied. The second problem is the stability of the method for solving the linear system in each iteration step. Using gaussian elimination for solving a linear system

$$Ax = b$$

we obtain an upper bound for the relative error in the solution

$$(2.2.1) \qquad \frac{\|\delta x\|}{\|x\|} \leq \kappa(A) R\varepsilon = \alpha$$

where ε is the precision of arithmetic, $\kappa(A) = \|A\|\|A^{-1}\|$ is the condition number of the matrix A and R is some constant, mainly depending on the order of the system and specific details of the method used (WILKINSON [49]). It is assumed that $\kappa(A) << 1/\varepsilon$.

Let $\bar{\psi}(x)$ be the value obtained by evaluating the right hand side of (2.1.3) with precision of arithmetic ε. Then, we obtain for the error in $\bar{\psi}(x)$ as an approximation to z (BUS [14]):

$$(2.2.2) \qquad \|\bar{\psi}(x)-z\| \leq \varepsilon\|x\| + L(x)\|x-z\| + Q(x)\|x-z\|^2,$$

where

$$(2.2.3) \qquad L(x) = \beta(x) + (1 + \beta(x))c(x)\|[J(x)]^{-1}\|,$$

$$(2.2.4) \qquad Q(x) = (1 + L(x))S(x,z),$$

$$(2.2.5) \qquad \beta(x) = (1+\varepsilon)\alpha(x) + \varepsilon,$$

$\alpha(x)$ and $S(x,z)$ are given by (2.2.1), with A replaced by J(x), and (2.1.4), respectively, and c(x) is a measure for the error in M(x) as a numerical approximation to J(x).

Hence, using a method defined by (2.1.3) for solving problem (1.3),

we can not expect to obtain a numerical solution in a relative precision which is higher than the precision of arithmetic. Furthermore, convergence at all depends on the value of

$S(x,z)$, the convergence factor of the exact Newton method, which depends on the problem,

$c(x)$, a measure of the error in $M(x)$ as a numerical approximation to $J(x)$, which depends on the method as well as on the problem,

$\beta \approx \alpha$, which reflects the condition number of the linear subproblem and depends on the problem as well as on the method used for solving the linear system.

Anyhow,

$$(2.2.6) \qquad \Gamma(x) = L(x) + Q(x)\|x-z\|,$$

for x in some region U, containing the solution, is the critical number which reflects whether a problem is easily solvable by a given method. If

$$\Gamma(x) < 1, \qquad \text{for } x \in U$$

then convergence is assured for each starting point in U. Since almost all methods given in this report may be described by (2.1.3), we will use in section 5.2 the quantity

$$(2.2.7) \qquad \Gamma = \sup_{x \in U} \Gamma(x),$$

where

$$(2.2.8) \qquad U = \{x_0\} \cup \{x \in \mathbb{R}^n \mid \|x-z\| \leq r(x_0)\},$$

$$r(x) = \varepsilon\|x\| + \|\phi(x)-z\| + L(x)\|\phi(x)-x\|,$$

for selecting problems which are easy or difficult to solve (see also BUS [14]).

2.3. *The influence of scaling*

It is well known that scaling of the variables may influence the behaviour of a method for solving problem (1.3).

Suppose problem (1.3) is given and we introduce new variables $\bar{x}$ defined by

$$\bar{x} = Dx, \tag{2.3.1}$$

where D is some diagonal matrix with positive nonzero diagonal elements d_i (i=1,...,n). Then we obtain for the Jacobian matrix

$$J(\bar{x}) = \frac{d}{d\bar{x}} F(\bar{x}) = J(x)D^{-1}$$

and for the tensor of partial second derivatives

$$H(\bar{x}) = (H_{ijk}(\bar{x})) = \left(\frac{\partial F_i(\bar{x})}{\partial x_j \partial x_k}\right);$$

$$H_{ijk}(\bar{x}) = \frac{1}{d_j d_k} H_{ijk}(x).$$

Hence, $\beta(x)$, $c(x)$, $\|[J(x)]^{-1}\|$ and $S(x,z)$ are all changed by scaling and therefore the number Γ may well be reduced. However, it is hard to prove such a statement for a specific problem. In practice, it seems best to scale the variables such that they all have about the same order of magnitude. Another reason for scaling in such a way may be that one wants to have all variables in about the same relative precision (see section 2.4).

2.4. *The choice of stopping criteria*

Since the methods used for finding the solution of a system of nonlinear equations are iterative, we have to find some stopping criteria. For these methods, the most commonly used criteria are

$$\|x_k - x_{k-1}\| \leq t_0\|x_k\| + t_1, \tag{2.4.1}$$

$$\|F(x_k)\| \leq t_2, \tag{2.4.2}$$

where t_0, t_1, t_2 are tolerance values which should be given by the user. These criteria can be applied since these quantities are known in each iteration step. In all methods discussed in this report, the calculation of a new iterate is done by some kind of linear approximation of the function and the error in such an approximation is highly dependent on the second

derivative of the function. When we take a Newton-like method as an example, we see from (2.2.2) that

$$(2.4.3) \qquad \frac{\|\bar{\psi}(x_k)-z\|}{\|x_k-z\|} \leq \frac{\varepsilon}{\|x_k-z\|}\|x\| + \Gamma(x).$$

Hence, if the right hand side is nearly equal to 1 then the step length may satisfy (2.4.1) while the error in $\bar{\psi}(x_k)$ as an approximation to z may be almost arbitrarily large. We see from (2.2.3) and (2.2.4) that $\Gamma(x)$ may be nearly equal to 1, without $\|x_k-z\|$ being small, when $\|[J(x)]^{-1}\|$ and/or the norm of the second derivative is large relative to 1. Therefore, it is desirable to use both criteria (2.4.1) and (2.4.2) in an algorithm for solving nonlinear systems, although one should realize that this is also not enough to guarantee the required precision. In order to be sure, it is necessary to know more about the behaviour of the function considered.

Finally, we should point out that scaling of the variables in such a way that each variable has about the same order of magnitude, is desirable when the usual norms are used (e.g. the euclidean or maximum-norm) in (2.4.1) and (2.4.2) and when one wants to obtain the variables in about the same precision.

3. DESCRIPTION OF METHODS KNOWN

3.1. *Newton's method and some of its modifications*

The most commonly known method for solving nonlinear equations using analytical derivatives of the function is Newton's method (also called the method of Newton-Raphson). This method is defined by (2.1.2). However, in this form, it has the disadvantage that the user has to supply analytical expressions for the elements of the Jacobian matrix. This may be very difficult or even impossible. To remove this difficulty one can approximate the elements of the Jacobian matrix with difference formulas. Methods obtained in this way are sometimes called discretized Newton methods and they are included in the class of Newton-like methods which are defined by (2.1.3). However, the approximation of the Jacobian matrix with difference formulas requires, even in its simplest form, at least n extra function evaluations (n denotes the number of variables). This appears to be inefficient, as

will be shown from the experimental results. A second disadvantage of using discretized Newton methods is that they are sometimes very sensitive to the step size used. In fact, this step size should be balanced in such a way that the truncation error and the error due to cancellation of significant digits by subtracting two almost equal function values have the same order of magnitude. However, the truncation error depends on the norm of the second derivative tensor which is usually not available.

A second disadvantage of Newton's method, which is in fact shared with all Newton-like methods, is the possibility of divergence in cases that the Jacobian matrix is (nearly) singular for some x_k. There is a simple strategy for avoiding an unstable behaviour when the Jacobian is only nearly singular. This is by using step size control. Instead of formula (2.1.2) the iteration is then defined by

$$\bar{\phi}(x) = x - \omega(x)[J(x)]^{-1}F(x), \tag{3.1.1}$$

where the scalar $\omega(x)$ determines the step length and is chosen, for instance, such that the method is norm-reducing in the sense that:

$$\|F(\bar{\phi}(x))\| \leq \|F(x)\|. \tag{3.1.2}$$

A strategy which can also deal with singular Jacobian matrices was originally given by LEVENBERG [33] and MARQUARDT [35]. It can be defined by:

$$\bar{\phi}(x) = x - [J(x) + \lambda(x)J^T(x)]^{-1}F(x), \tag{3.1.3}$$

where $\lambda(x) \geq 0$ is chosen such that $J(x) + \lambda J^T(x)$ is nonsingular and mostly such that (3.1.2) is satisfied.

A very elegant method for avoiding the problems of a singular Jacobian matrix is the use of the Moore-Penrose pseudo-inverse. Here the iteration is defined by

$$\bar{\phi}(x) = x - [J(x)]^{+}F(x), \tag{3.1.4}$$

where A^{+} denotes the pseudo-inverse of the matrix A.

One should note that for all these methods, the solution of a linear system is needed or even the calculation of the pseudo-inverse. Since the

number of arithmetical operations needed for such calculations is of order n cubed, it may be inefficient for large n.

In order to give a theoretical analysis of the given methods, one should realize that they are all Newton-like methods, even the one given by (3.1.4) if the Jacobian matrix is assumed to be nonsingular. Therefore, the theory in section 2 can be applied. For a detailed analysis see BOGGS & DENNIS [1], BUS [14] or ORTEGA & RHEINBOLDT [40].

3.2. *Generalized secant and related methods*

The secant method for solving the equation

$$f(t) = 0 \in \mathbb{R}, \quad t \in \mathbb{R},$$

which can be defined by

$$t_{k+1} = t_k - \frac{f(t_k)(t_k - t_{k-1})}{f(t_k) - f(t_{k-1})},$$

can be extended to n dimensions. Then we calculate the next iterate as the intersection of n hyperplanes which interpolate F(x) at given points in a neighbourhood of x (see ORTEGA & RHEINBOLDT [40]).

This method can be formulated as a Newton-like method in the following way:

let $x, x^1, \ldots, x^n$ be given;

(3.2.1) $H = [x-x^1, \ldots, x-x^n]$

be the matrix with columns $x - x^i$, $i = 1, \ldots, n$; then

(3.2.2) $\psi(x) = x - [M(x,H)]^{-1}F(x)$,

where

(3.2.3) $M(x,H) = [F(x+He_1) - F(x), \ldots, F(x+He_n) - F(x)]H^{-1}$

is an approximation to J(x). (Here e_i denotes the i-th unit-vector.)

Obviously, this method requires the solution of a linear system in every iteration step. In order to avoid this we can use $[M(x,H)]^{-1}$, or rather the triangularized form of M(x,H), in a certain number of subsequent iteration steps. Such a modified generalized secant method can be defined by the super-iteration (SCHWETLICK [47]):

let $x,x^1,\ldots,x^n$ be given,
let moreover H and M(x,H) be defined by (3.2.1) and (3.2.3) respectively
then
set $\nu^{(0)}(x) = x$;
for $k = 0,1,\ldots,u$ compute

$$\nu^{(k+1)}(x) = \nu^{(k)}(x) - [M(x,H)]^{-1}F(\nu^{(k)}(x)) \tag{3.2.4}$$

$$\phi(x) = \nu^{(u+1)}(x), \tag{3.2.5}$$

where u is some fixed value which should depend on the order of convergence of the iteration.

We call this a super-iteration, since u + 1 modified iteration steps are taken together as one step. Another useful modification of the generalized secant method is proposed by GRAGG & STEWART [28]. In their method the orthogonal decomposition of the subsequent matrices M(x,H) is used. The advantage is that, once this orthogonal decomposition is calculated, which requires $0(n^3)$ arithmetical operations, only $0(n^2)$ arithmetical operations are required to obtain the orthogonal decomposition of the matrix used in the next step.

With the formulations (3.2.2) and (3.2.5) we may again use the analysis of Newton-like method to obtain results about the convergence behaviour. However, the error in M(x,H) as an approximation to J(x) (in (3.2.2)) or $J(\nu^{(k)}(x))$ (in (3.2.5)) is the most important problem here (note that H is singular when $x_1,\ldots,x_n$ are linearly dependent). For further results about the stability and convergence of these methods, see GRAGG & STEWART [28], ORTEGA & RHEINBOLDT [40] and ROBINSON [46].

3.3. *Quasi-Newton methods*

One of the most remarkable Newton-like algorithms is the so-called quasi-Newton algorithm (DAVIDON [18], BROYDEN [7], [8], POWELL [44]). In this algorithm the Jacobian matrix or its inverse is approximated by a matrix which is updated in each iteration step with the information gained so far about the function. The algorithm can be defined by:

$$\psi(x) = x - Q(x)F(x),$$
$$Q(\psi(x)) = Q(x) + U(x,\psi(x),F(x),F(\psi(x)),Q(x)) \tag{3.3.1}$$

or

$$\psi(x) = x - [P(x)]^{-1}F(x),$$
$$P(\psi(x)) = P(x) + U(x,\psi(x),F(x),F(\psi(x)),P(x)). \tag{3.3.2}$$

The updating of the matrices Q and P requires no additional function evaluations. Clearly, the formulation given by (3.3.2) requires the solution of a linear system. So in this formulation the number of arithmetical operations needed per iteration step is proportional to n cubed. Therefore, at least from a theoretical point of view, formulation (3.3.1) is preferable since the number of arithmetical operations needed per iteration step is only proportional to n squared. We can use the same analysis as for Newton-like algorithms (see BUS [14]) to obtain results about the convergence behaviour of formulation (3.3.2) and in a slightly modified way also of formulation (3.3.1). However, proving reasonable bounds for the errors in Q(x) and P(x) as approximations to $[J(x)]^{-1}$ and J(x) respectively, appears to be a hard problem. An analysis of quasi-Newton methods is given by BROYDEN [9] and DENNIS & MORÉ [20],[21].

3.4. *Methods of component-wise approximation*

These methods can be defined by the following formula (see also ORTEGA & RHEINBOLDT [40]):

$$x_{k+1}^{(i)} = g^{(i)}(x_{k+1}^{(1)},\ldots,x_{k+1}^{(i-1)},x_k^{(i)},\ldots,x_k^{(n)}), \quad i= 1,\ldots,n, \quad k = 0,1,\ldots, \tag{3.4.1}$$

where $x_k = (x_k^{(1)},\ldots,x_k^{(n)})^T \in \mathbb{R}^n$ and $g^{(i)}: \mathbb{R}^n \to \mathbb{R}$ for $i = 1,\ldots,n$. Hence, a new approximation $x_{k+1}^{(j)}$ to the j-th component of the solution vector is used as soon as it is available. The choice of g_i is usually based on expanding the function into a Taylor series at the point $(x_{k+1}^{(1)},\ldots,x_{k+1}^{(i-1)},x_k^{(i)},\ldots,x_k^{(n)})^T$ and neglecting second and higher order terms. Examples of such methods are the Gauss-Seidel algorithm and successive over-relaxation methods (see ORTEGA & RHEINBOLDT [40], section 7.4).

A remarkable algorithm which we will also incorporate in this class is given by BROWN [2]. This method is based on expanding a component, say $F^{(i)}(x)$, of the function $F(x) = (F^{(1)}(x),\ldots,F^{(n)}(x))^T$ into a Taylor series. Neglecting second and higher order terms, we obtain a linear approximation which is equated to zero and solved for one of the variables, $x^{(j)}$ say. Subsequently, another function component is expanded into a Taylor series as a function of the remaining n - 1 variables ($x^{(j)}$ is substituted) and equated to zero again. After n such steps we obtain a new approximation to the solution vector. For a detailed description see BROWN [2]. He also gives theoretical justifications for his method and some results about the convergence behaviour. For further theoretical results about methods of component-wise approximation see ORTEGA & RHEINBOLDT [40].

3.5. *Continuation methods*

The continuation methods (DAVIDENKO [17], BROYDEN [8], MEYER [36] and ORTEGA & RHEINBOLDT [40]) have a rather special place among the methods for solving systems of nonlinear equations, because, in fact, the problem is transformed into a sequence of problems of the form (1.3) which might be easier to solve than the original problem. Let the problem $[F(x) = 0;x_0;D]$ (cf. (1.4)) be given. Then this problem is replaced by the sequence of problems

$$(3.5.1)\qquad P_k: [G(x,\theta_k) = 0;z_{k-1};D], \qquad k = 1,\ldots,m,$$

where $z_0 = x_0$ and z_k is a solution of P_k, $k = 1,\ldots,m$. Furthermore,

$$G(x,\theta_m) \equiv F(x)$$

and a solution of $G(x,\theta_0)$ should be easy to calculate. Examples of G are (BROYDEN [8], MEYER [36])

$$(3.5.2) \qquad G(x,\theta_k) \equiv F(x) - (1-\theta_k)F(x_0)$$

or

$$(3.5.3) \qquad G(x,\theta_k) \equiv (x-x_0)(1-\theta_k) + \theta_k F(x),$$

where $0 = \theta_0 < \theta_1 < \ldots < \theta_m = 1$. These methods are primarily designed to remove the difficulty of choosing a good initial guess. Hence, these methods are designed to be robust rather than efficient.

Obviously, we can use any method of the preceding sections for solving the subproblems P_k, $k = 1,\ldots,m$.

3.6. *Additional remarks*

In practice, it appears to be almost impossible to separate the algorithms according to the theoretical framework given in this section. Several procedures known use mixtures of the methods described and quite often, the first step is entirely different from all others. For instance, the initial approximation to the inverse Jacobian used in quasi-Newton methods is usually obtained with forward difference formulas and inversion. However, summing up the basic tools used in the various algorithms is sufficient to make this report comprehensible.

Considering the data that are required by the various algorithms, we can distinguish two classes:

1. algorithms that use a Jacobian matrix whose elements are obtained by the evaluation of analytical expressions supplied by the user;
2. algorithms that only require the programming of the function.

In the first case, the efficiency is dependent on the ratio between the time needed to evaluate the function and the time needed to evaluate the Jacobian matrix. As will appear from our comparisons, the use of an algorithm that requires analytical derivatives will not necessarily be more efficient than using an algorithm that requires only function evaluations.

4. SELECTED PROGRAMS

4.1. *Introductory remarks*

The goal of this report is to test those programs for solving systems of nonlinear equations that are available as computer programs in ALGOL 60 or FORTRAN from the literature or well-known software libraries. The sources from which the programs are selected are:

1. Collected Algorithms from CACM,
2. Computer Journal,
3. Computing,
4. Mathematical Science Library [37],
5. NUMAL, [39],
6. Some specific papers as are given by BROWN [2], GRAGG & STEWART [29] and POWELL [44].

It should be pointed out here that we did not test programs for minimizing sums of squares of nonlinear functions which can also be used for solving systems of nonlinear equations. In our opinion this should be the scope of a separate test report. Furthermore, we will not consider programs that implement continuation methods. In fact, the program used for solving the subproblems that arise in these methods should be selected on the basis of this test report, while the choice of $G(x,\theta)$ and the stepsize (cf. section 3.5) is outside the scope of this report. Therefore, one of the programs (nonlinb) given by BROYDEN [8] is omitted.

For two reasons we distinguish between programs written in ALGOL 60 and in FORTRAN. Firstly, since arithmetical operations and elementary functions deliver different results in different languages, a problem defined in ALGOL 60 differs from the mathematical analogue in FORTRAN. Therefore the tests are not quite comparable. Secondly, we like to give the user a possibility to overview the field of programs which are available in the programming language he uses.

The source texts of the programs are given in the appendix. In this report we will denote the programs to be tested by capitals.

4.2. *Programs written in* ALGOL 60

PROGRAM A

This program, written by Kok (see also NUMAL [39], section 5.1) is based on Newton's algorithm (see section 3.1). We supplied analytical derivatives. No step size control is performed. There are no method dependent control parameters in this program. In each iteration step one evaluation of the function, one evaluation of the Jacobian matrix and the solution of a linear system have to be performed.

PROGRAM B

This program, written by Kok, is based on Newton's algorithm with step size control (cf. (3.1.1)). We supplied analytical derivatives. In this algorithm, $\omega(x)$ is chosen by successively trying the values 2^{-k} for $k = 0,1,2,\ldots,u-1$, where the upper bound u should be supplied by the user. In fact, $\omega(x) = 2^{-r}$, where $r = 0$ if

$$\|F(x-s)\| \leq \|F(x)\|$$

otherwise, r is the minimum of u and the smallest value of k such that

$$\|F(x-2^{-k}s)\| < \|F(x)\|$$

and

$$\|F(x-2^{-(k+1)}s)\| \geq \|F(x-2^{-k}s)\|,$$

where $s = [J(x)]^{-1}F(x)$. In this program an error exit is incorporated when in t subsequent iteration steps the value of $\omega(x)$ is chosen to be 2^{-u}. The value of the integer t should also be given by the user. We chose u and t (*in* [6] and *in* [7] in the program given in the appendix) as follows:

$$u = 15,\ t = 1.$$

No other method dependent control parameters have to be set by the user.

In each iteration step one evaluation of the Jacobian matrix and the solution of a linear system have to be performed. The number of function evaluations in an iteration step depends on the value of r in that step. If $r = 0$, then one evaluation of the function is performed, otherwise

r + 2 evaluations of the function are performed.

PROGRAM C

This program is the same as program A, except for the evaluation of the Jacobian matrix, which is done by approximating it with forward difference formulas with step size equal to $10^{-4}\|x\| + 10^{-4}$, where x denotes the argument vector. There is no difference in the source texts of the programs A and C since the user has to program the evaluation of the Jacobian matrix. In each iteration step n + 1 evaluations of the function and the solution of a linear system have to be performed.

PROGRAM D

This program is the same as program B, except for the evaluation of the Jacobian matrix, which is done by approximating it with forward difference formulas with step size equal to $10^{-4}\|x\| + 10^{-4}$. As for the programs A and C there is no difference between the source texts of the programs B and D. In each iteration step a linear system has to be solved. The number of function evaluations in a certain iteration step depends on the value of r (see program B). If r = 0 then n + 1 otherwise r + n + 2 evaluations of the function have to be performed.

PROGRAM E

The Newton-like algorithm as given by PANKIEWICZ [41]. This algorithm is the same as algorithm C except for the choice of the step size, used to approximate the Jacobian matrix with forward differences. In fact, this step size should be given initially by the user and it is multiplied by 0.1 in every step. Since choosing the step size too small may cause singularity of the approximation to the Jacobian matrix, we followed the advise of PANKIEWICZ [41] to use the given procedure repeatedly.
As is required, we incorporated a procedure for solving linear systems. Furthermore we changed some minor details concerning error exits such that it became more convenient for our test programs.

PROGRAM F

This is a program given by SCHWETLICK [47], which is based on the modified generalized secant algorithm given by (3.2.5). In order to be able to deal with zero vectors, we incorporated in our program stopping criterion (2.4.2) and replaced the statements:

$$g := y[k] \times eps$$

and

$$\underline{\text{if}}\ \text{abs}(h) > \text{abs}(eps) \times \text{abs}(g)$$

by

$$g := y[k] \times eps + eps \times eps$$

and

$$\underline{\text{if}}\ \text{abs}(h) > \text{abs}(eps \times g) + \text{abs}(eps)$$

We chose $eps = 0.0001$. This value is used as a step length to obtain the matrices H and M(x,H), (in fact x^i is chosen to be $x + eps \times e^i$, where e^i denotes the i-th unit vector). Furthermore the value of *pivot* is chosen equal to the precision of computation ($\approx 10^{-14}$). This value is used to check whether or not the matrix H (cf.(3.2.1)) is singular. In each (super-) iteration step of this algorithm the solution of a linear system is required and at least n + 1 (cf.(3.2.5)) evaluations of the function have to be performed.

PROGRAM G

This program, given by DULLEY & PITTEWAY [22], is based on the generalized secant algorithm (formula (3.2.2)). As is required, we incorporate a procedure for solving linear equations (Bus, NUMAL [37], section 3.1.1.1.1.1.3). The value of the control parameter *initstep*, which is used as a step length in the same way as *eps* is used in program F, is chosen equal to 0.0001.

In each iteration step of this algorithm the solution of a linear system is required and one evaluation of the function is required.
We tested two versions:

program Ga: the program given by DULLEY & PITTEWAY [22] with the minor changes described above;

program Gb: the same program but with the change incorporated, which is proposed by VANDERGRAFT & MESTENYI [48].

PROGRAM H

This program is given by BROYDEN [8] (procedure *nonlina*) and is based on the quasi-Newton algorithm defined by (3.3.1). Initially, an approximation to the inverse Jacobian matrix is obtained by using the updating formula with fixed steps along the coordinate axis. We like to point out here that this requires $3n^3$ multiplications, while normal inversion of a forward difference approximation to the Jacobian matrix would only require n^3 multiplications (neglecting lower order terms). So it seems to be rather inefficient to use the method in the form proposed by Broyden.
In the source text that we used, we chose the step size in the initializing phase relative to the value of the arguments:

$$| x_i | \times 10^{-6} + 10^{-10}$$

We used a version which is converted for use with the software library NUMAL [39].
After the rather expensive initializing phase the number of arithmetical operations per iteration step is only proportional to n squared.
Furthermore n + 1 evaluations of the function have to be performed in the initializing phase and one in each iteration step.

PROGRAM I

This program is based on the method of component-wise approximation given by BROWN [2] (see also section 3.4). The source text that we use is already adapted to our software library NUMAL [39]. Apart from some details, such as adding absolute tolerances where only relative tolerances were used, it is equivalent to the source text given by BROWN [2].

In this program, difference approximations to the elements of the Jacobian matrix are made with a step size equal to 0.001. Furthermore, instead of supplying some procedure for calculating the vector function F(x), one should supply a procedure that calculates the i-th component of this vector F(x), for given i ($1 \leq i \leq n$).
Furthermore it is advised to define the function in such a way that its linear components come first.
The number of multiplications needed per iterative step is $0.25\ n^4$, where lower order terms are neglected, and the number of function-component evaluations equals $(n^2+3n)/2$ in each step. For a more up to date implementation of this method see program 0.

4.3. *Programs written in* FORTRAN

PROGRAM J

This is the program, based on Newton's method, which is available in the MSL [37] software library as routine NEWT. The Jacobian matrix is approximated with forward difference formulas and there is a possibility of incorporating step size control. The step size control is done in terms of a fraction of the norm of the current solution vector. In fact, the step vector is multiplied repeatedly with the factor

$$\min(E/(S2/(S1+0.001))^{\frac{1}{2}},\ 1)$$

until (3.1.2) is satisfied. Here S2 denotes the norm of the current solution vector squared, S1 is the norm of the step vector squared and E is the so-called maximum fractional change allowed. When E is chosen large enough, no step size control is done.
As was suggested in the manual, we changed the statement

```
RATIO = SQRT(S2/S1)
```

in

```
RATIO = SQRT(S2/S1+0.001),
```

in order to be able to deal with the zero vector as initial guess.

We tested this program for two values of E:

Program Ja: $E = 10^{100}$, so that no step size control can occur;

Program Jb: E = 0.18, a value suggested in the manual, such that step size control should work as well as possible.

For both programs the solution of a linear system is required in each iteration step. Furthermore, without step size control, n + 1 evaluations of the function have to be performed in each iteration step, with step size control this number may be more.

The source text of this program is not given in the appendix since it is not free for publication.

PROGRAM K

This program is given by GRAGG & STEWART [28], and is based on the generalized secant algorithm. The matrices appearing are kept as products of orthogonal matrices (see section 3.2). We made two changes to the source text as given by Gragg and Stewart. The first one is on line 3000 of subroutine SSM which reads in our program : MCEPS = 1.E-14 since the precision of computation on the computer used is about that value. The second one is the correction of a small programming error on line 3200 of subroutine SSM which should read : OUTBND = NN + 3.

The program has the feature to deal directly with linear function components. We did not use this feature for the general tests.

The user has to provide n + 1 starting guesses of the solution vector. Since in our problems only one initial guess is given we generate them automatically as follows:

$$x_0^{(0)} = x_0$$

$$x_0^{(k)} = x_0 + se^{(k)} \qquad , k = 1,\ldots, n. \tag{4.3.1}$$

Where x_0 denotes the given initial guess, $x_0^{(k)}$ the k-th starting guess for the program, $e^{(k)}$ the k-th unit-vector and s some fixed value. We do, in fact, consider two programs:

Program Ka: s = 0.5 ;

Program Kb: s = 0.001.

For both programs, Householder orthogonalisation of two n-th order matrices (see WILKINSON [50]) is necessary initially, which requires $8n^3/3$ arithmetical operations (neglecting lower order terms).The iteration steps require only $0(n^2)$ arithmetical operations. The number of function evaluations needed per step may vary from 1 up to n. We do not give the source text that we used, since it is fully given by Gragg and Stewart, apart from the two small corrections mentioned above.

PROGRAM L

This program, which is available in the MSL [37] software library is based on the generalized secant algorithm given in section 3.2 (formula (3.2.2)). As in program K, the user has to provide n + 1 starting guesses, which are chosen according to formula (4.3.1) with s = 0.5.
In each iteration step the solution of one or two linear systems is required.(There is a recovery scheme in cases the matrix appears to be singular.) In each iteration only one evaluation of the function is performed. The source text of this program is not free for publication.

PROGRAM M

This program, which is available in the MSL [37] software library is based on the quasi-Newton algorithm defined by (3.3.2). In each iteration step the solution of a linear system and one evaluation of the function has to be performed. The source text is not available for publication.

PROGRAM N

This program is given by POWELL [44] and is basically an implementation of a quasi-Newton method as defined by (3.3.1). A version of this program is also available in the NAG [38] software library. Here, this method is combined with the steepest descent method for minimizing $\|F(x)\|$ and with Newton's method with forward difference approximations to the Jacobian matrix. Initially, and in some iteration steps, the approximation to the inverse Jacobian matrix is (re-)set by inverting the forward difference approximation to the Jacobian matrix. Hence, initially and in some iteration steps, the number of arithmetical operations is $0(n^3)$ (neglecting lower order terms). In all other steps it is proportional to n^2.

The number of function evaluations needed in a particular step depends on what kind of step it is.
The value of the control parameter DMAX is chosen to be equal to 10.
DMAX controls the changes in the variables and is used in an error condition. For the control parameter DSTEP we tested two values:
Program Na: DSTEP = 10^{-4} ;
Program Nb: DSTEP = 10^{-7} .
DSTEP is used as a step size for the forward difference approximation, but also for controlling the updating of the approximation to the Jacobian matrix. We used the source text given by Powell except for the change of line 0092 where the call of subroutine MB01B for solving a linear system is replaced by a call of the subroutine INVERS from the MSL [37] software library.

PROGRAM O

This program is obtained from the University Computer Center of the University of Minnesota and is based on the method of component-wise approximation of BROWN [4]. A FORTRAN-version of this agorithm which is the same as we used is available in the IMSL [32] software library.
The program has the same properties as program I. In the program the value for the step length to calculate the forward difference approximations to the elements of the Jacobian matrix has been given the value 10^{-8}.
We tested the program for two different values for the step length:
Program Oa: steplength 10^{-4}
Program Ob: steplength 10^{-8}.
Furthermore we used an absolute tolerance value in the stopping criterion in order to be able to solve problems with the zero-vector as a solution.

4.4. *General remarks about the programs selected.*

Although it is desirable that both stopping criteria (2.4.1) and (2.4.2) are used in a program for solving systems of nonlinear equations, almost none of the programs given in this section meets these requirements In our opinion it is not too hard to incorporate these criteria. However, we did not do so for testing, partly to reduce the possibility of making errors, partly because different norms are induced by the

various programs, so that they would not be equivalent after all.

Finally we give in table 4.1 the work that has to be done by the various programs in the initializing phase, Λ_i say, as well as per iteration step, Λ_s say. In fact, we give the number of multiplications needed in the initializing phase and per iteration step. Since, these numbers will usually depend on the number of variables n, we denote Λ_i and Λ_s as functions of n and only give the highest order term.
However, if the highest order term is of order n^2, we neglect all.
For program F, Λ_s denotes the work per super-iteration and for program L, we assume that one linear system is solved per iteration step.

TABLE 4.1

Size of magnitude of Λ_i and Λ_s relative to n

PROGRAM	Λ_i	Λ_s
A	-	$\frac{1}{3}n^3$
B	-	$\frac{1}{3}n^3$
C	-	$\frac{1}{3}n^3$
D	-	$\frac{1}{3}n^3$
E	-	$\frac{1}{3}n^3$
F	-	$\frac{1}{3}n^3$
G	-	$\frac{1}{3}n^3$
H	$3n^3$	-
I	-	$\frac{1}{4}n^4$
J	-	$\frac{1}{3}n^3$
K	$\frac{8}{3}n^3$	-
L	-	$\frac{1}{3}n^3$
M	-	$\frac{1}{3}n^3$
N	n^3	sometimes n^3
O	-	$\frac{1}{4}n^4$

5. CLASSIFICATION OF PROBLEMS AND SELECTION OF TESTPROBLEMS

5.1. *Classification of problems*

We consider the class Ψ of all problems of the form (1.4). When we measure the efficiency of a program for solving some problem from class Ψ we may distinguish the following three characteristics which influence this efficiency.

a. The degree of difficulty for solving.
b. The number of variables of the problem.
c. The computational effort of an evaluation of the function, i.e. the number of basic arithmetical operations needed to evaluate the function.

Before defining precisely these characteristics we like to point out why the first characteristic is important. It is obvious that it is desirable to know in advance whether a problem is easily solvable or not. However, the degree of difficulty also depends on the method used and, in practice, it is very hard, or even impossible, to measure it before solving the problem. Hence, a classification according to this characteristic will, in general, not be very helpful to the user. However, it is extremely important for comparing the efficiency of the various programs. Since none of the programs for solving nonlinear systems is such that it solves all problems from class Ψ we have to take into account that the programs tested fail sometimes. Therefore we have to decide whether a failure is due to a difficultly solvable problem or to bad programming of the method. If the problem appears to be difficultly solvable then it makes no sense to draw from this failure the conclusion that the program is inefficient, for then all programs will appear to be inefficient. Clearly, we need a precise definition of the notions easily solvable and difficultly solvable. Although several definitions are possible we choose one which is based on the fact that all methods considered in this report can be defined as Newton-like methods in some way or another, and which appears to be convenient. In fact, we use as a model-method the Newton-like method defined by (2.1.3), where $M(x_k)$ is calculated with forward difference approximations. For this method we can compute an upper bound for the error in $M(x)$ as an approximation to $J(x)$, by calculating the second derivative and using the mean value theorem. Hence, for this method we can calculate

an upper bound for the number Γ (cf. (2.2.7)) of a certain problem. (For more details and an example see BUS [14].) When this upper bound appears to be less than 1, then, obviously, the problem is easy to solve by this Newton-like method. However, we do not use this number 1 so rigorously, because we made a lot of choices and sometimes, crude estimates. Therefore, we end up with the definition:

DEFINITION 5.1.1

A problem is *easily solvable* when the number Γ, given by (2.2.7), for this problem and for Newton's method with forward difference approximations to the Jacobian matrix, has an order of magnitude about 1 or less.
Otherwise the problem is *difficultly solvable*.

We will denote the class of easily solvable and difficultly solvable problems with superscripts e and d respectively. So Ψ^e denotes the class of easily solvable problems, Ψ^d the class of difficultly solvable problems.

As far as classification according to the number of variables is concerned, we distinguish between small and large problems, where the choice of the bound, n = 15, is a matter of practical experience.

The last classification quantity is induced by the fact that for most programs tested the number of basic arithmetical operations needed to perform one iteration step is not neglectable relative to the number of arithmetical operations needed to evaluate the function when the function is not too complicated. Therefore, neither the number of function-evaluations, nor the number of iteration steps is a good measure for the efficiency of the programs. We should use a combination of these two quantities, which depends on the expensiveness of the function. For small problems we use only a distinction between cheap and expensive functions where it is mainly a matter of feeling how to classify a certain problem. For large problems, however, we can relate this quantity to the number of variables n. When we express the number of arithmetical operations needed to evaluate a function as a polynomial in n and we assume that αn^β is its leading term, where β is some integer, usually equal to 1,2,3 or 4 and α is some real, then we distinguish between:

very cheap problems : $\beta = 1$,
cheap problems : $\beta = 2$,
expensive problems : $\beta = 3$,
very expensive problems : $\beta \geq 4$.

Combining this with classification according to the size we obtain the following classification of problems in the class Ψ^e of easily solvable problems:

Ψ^e_{s1} : small ($n \leq 15$), cheap and easily solvable problems;
Ψ^e_{s2} : small ($n \leq 15$), expensive and easily solvable problems;
$\Psi^e_{\ell 1}$: large ($n > 15$), very cheap ($\beta = 1$) and easily solvable problems;
$\Psi^e_{\ell 2}$: large ($n > 15$), cheap ($\beta = 2$) and easily solvable problems;
$\Psi^e_{\ell 3}$: large ($n > 15$), expensive ($\beta = 3$) and easily solvable problems;
$\Psi^e_{\ell 4}$: large ($n > 15$), very expensive ($\beta \geq 4$) and easily solvable problems.

We obtain analogously for the subclass Ψ^d of difficultly solvable problems the classes:

Ψ^d_{s1}, Ψ^d_{s2}, $\Psi^d_{\ell 1}$, $\Psi^d_{\ell 2}$, $\Psi^d_{\ell 3}$ and $\Psi^d_{\ell 4}$.

5.2 *Definition of testproblems*

We have chosen a number of testfunctions known from literature. Most of them are used with several initial guesses, since it depends highly on the choice of the initial guess, whether a problem is easily solvable or not.

5.2.1 (BROWN [3]).

$$F_i(x) = -(n+1) + x_i + \sum_{j=1}^{n} x_j, \quad i = 2,\ldots,n;$$

$$F_1(x) = -1 + \prod_{j=1}^{n} x_j.$$

Initial guess: $x_i = 0.5$, $i = 1,\ldots,n$.

Order : $n = 2, 3, 5, 10, 15$ and 25.

Solutions : $x_i = 1$ $(i = 1,\ldots,n)$;
for instance for $n = 5$, approximately:
$x = (-0.579, -0.579, -0.579, -0.579, 8.90)^T$.

Remarks: All function components are linear except for the first one.

5.2.2 (BROWN [3]).

$$F_1(x) = x_1^2 - x_2 - 1,$$
$$F_2(x) = (x_1 - 2)^2 + (x_2 - 0.5)^2 - 1.$$

Initial guess: 0. $(0.1, 2)^T$,
1. $(2, 0.5)^T$,
2. $(-1, 1.5)^T$,
3. $(1, 0.99)^T$.

Solutions : $(1.54634288, 1.39117631)^T$,
$(1.06734609, 0.139227667)^T$,
approximately.

5.2.3 (FREUDENSTEIN & ROTH [26], BROWN [3])

$$F_1(x) = -13 + x_1 + ((-x_2 + 5)x_2 - 2)x_2,$$
$$F_2(x) = -29 + x_1 + ((x_2 + 1)x_2 - 14)x_2.$$

Initial guess: 0. $(15, -2)^T$,
1. $(-5, 0)^T$,
2. $(-5, 3)^T$,
3. $(0, 2.24)^T$,

Solution : $(5, 4)^T$.

5.2.4 (CARNAHAN, LUTHER & WILKES [15], BROWN & CONTE [5])

$$F_1(x) = 0.5 \sin(x_1 x_2) - x_2/(4\pi) - x_1/2,$$
$$F_2(x) = (1 - 1/(4\pi))(\exp(2x_1) - e) + ex_2/\pi - 2ex_1.$$

Initial guess: 0. $(0.6, 3)^T$,
1. $(0.4, 3)^T$.

Solutions : $(0.5, \pi)^T$,
$(0.2994487, 2.836928)^T$, approximately,
$(1.604571, -13.36290)^T$, approximately.

5.2.5 (BROWN & CONTE [5])

$F_1(x) = 3x_1 + x_2 + 2x_3^2 - 3,$
$F_2(x) = -3x_1 + 5x_2^2 + 2x_1x_3 - 1,$
$F_3(x) = 25x_1x_2 + 20x_3 + 12.$

Initial guess: $(0, 0, 0)^T$.

Solutions : $(0.2900523, 0.6874306, -0.8492385)^T$,
$(1.1, -0.8, 0.5)^T$, approximately.

5.2.6 (BROWN [3])

$F_1(x) = x_1^2 - 2x_2 + 1,$

$F_2(x) = x_1 + 2x_2^2 - 3.$

Initial guess: 0. $(0, 1)^T$,
1. $(-0.5, 1)^T$,
2. $(1, -0.5)^T$,
3. $(1, -0.24)^T$.

Solutions : $(1, 1)^T$,
$(-1.402680, 1.483683)^T$, approximately.

5.2.7 (POWELL [44])

$F_1(x) = 10000x_1x_2 - 1,$

$F_2(x) = \exp(-x_1) + \exp(-x_2) - 1.0001.$

Initial guess: 0. $(0, 1)^T$,
1. $(0, -1)^T$.

Solution : $(1.098_{10}-5, 9.106)^T$, approximately.

Remark: This problem is badly scaled.

5.2.8 (POWELL [44])

$F_1(x) = x_1 - 1,$

$F_2(x) = x_1x_2 - 1.$

Initial guess: 0. $(-1, 2)^T$,
1. $(-1, -2)^T$,
2. $(0.01, 0)^T$.

Solution : $(1, 1)^T$.

5.2.9 (BROYDEN [8])

$F_1(x) = 10(x_2 - x_1^2)$

$F_2(x) = 1 - x_1.$

Initial guess: 0. $(-1.2, 1.0)^T$.

Solution : $(1, 1)^T$.

5.2.10 (BROYDEN [8])

$F_1(x) = 2(x_1 - 1) - 400x_1(x_2 - x_1^2),$

$F_2(x) = 200(x_2 - x_1^2).$

Initial guess: 0. $(-1.2, 1.0)^T$,
1. $(-1, 1)^T$.

Solution : $(1, 1)^T$.

5.2.11 (POWELL [44])

$F_1(x) = x_1,$

$F_2(x) = 10x_1/(x_1 + 0.1) + 2x_2^2.$

Initial guess: 0. $(3, 1)^T$,
1. $(0, 1)^T$,
2. $(-1, 1)^T$,
3. $(-0.9, 0.24)^T$.

Solution : $(0, 0)^T$.

5.2.12 (POWELL [43])

$$F_1(x) = 2(x_1 + 10x_2) + 40(x_1 - x_4)^3,$$

$$F_2(x) = 20(x_1 + 10x_2) + 4(x_2 - 2x_3)^3,$$

$$F_3(x) = 10(x_3 - x_4) - 8(x_2 - 2x_3)^3,$$

$$F_4(x) = -10(x_3 - x_4) - 40(x_1 - x_4)^3.$$

Initial guess: $(3, -1, 0, 1)^T$.

Solution : $(0, 0, 0, 0)^T$.

Remark: The Jacobian matrix has only rank two at the solution.

5.2.13 (DEIST & SEFOR [19])

$$F_i = \sum_{\substack{j=1 \\ j \neq i}}^{6} \cot(\beta_i x_j), \quad i = 1,\ldots,6,$$

where $\beta_1 = 0.02249$, $\beta_2 = 0.02166$, $\beta_3 = 0.02083$,
$\beta_4 = 0.02000$, $\beta_5 = 0.01918$, $\beta_6 = 0.01835$.

Initial guess: $x_i = 75.0$, $i = 1,\ldots,6$.

Solution : $(121.850, 114.161, 93.6488, 62.3186,$
$41.3219, 30.5027)^T$, approximately.

5.2.14 (FLETCHER [24])

Chebyquad, a function defined by the ALGOL 60 program given by FLETCHER [24]:

Order: n = 2, 3, 4, 5, 6, 7 and 9.

Initial guess: $x_i = i/(n + 1)$, $i = 1,\ldots,n$.

For reasons of brevity we omit the solution vectors.

5.2.15 (GHERI & MANCINO [27])

$$F_i = \beta n x_i + (i - \frac{n}{2})^\gamma + \sum_{\substack{j=1 \\ j \neq i}}^{n} [z_{ij}(\sin^\alpha(\ln(z_{ij})) + \cos^\alpha(\ln(z_{ij})))], \quad i = 1,\ldots n,$$

where $z_{ij} = \sqrt{x_j^2 + i/j}$, $i,j = 1,\ldots,n$.

Order : $n = 10, 20, 30, 50.$

Initial guess: $x = -F(0)\ (\frac{c+K}{2cK})$

where

$K = \beta n + (\alpha + 1)(n - 1)$ and

$c = \beta n - (\alpha + 1)(n - 1).$

We distinguished between the following cases:

0. $\alpha = 5$, $\beta = 14$, $\gamma = 3$;
1. $\alpha = 4$, $\beta = 7$, $\gamma = 1$;
2. $\alpha = 7$, $\beta = 17$, $\gamma = 4$.

A solution for n = 50 of case a is given by GHERI & MANCINO [27].

5.2.16 (FLETCHER & POWELL [25])

$$F(x) = e - (As(x) + Bc(x)),$$

where A and B are $n \times n$ matrices, whose elements are generated as random integers between -100 and +100, s(x) and c(x) are n-vectors such that:

$$s(x) = (\sin(x_1), \sin(x_2), \ldots, \sin(x_n))^T \text{ and}$$

$$c(x) = (\cos(x_1), \cos(x_2), \ldots, \cos(x_n))^T.$$

e is a vector, calculated as follows. Let x^* be a vector, whose elements are generated as random numbers between $-\pi$ and $+\pi$, then

$$e = As(x^*) + Bc(x^*).$$

Order : $n = 10, 20, 30, 40.$

Initial guess : $x^* + 0.01\ \delta$, where the elements of δ are random numbers between $-\pi$ and $+\pi$ and x^* as used for calculating e.

Solution : x^*, as used for calculating e.

5.2.17 (BROYDEN [11])

$$F_i(x) = (k_1 + k_2 x_i^2)x_i + 1 - k_3 \sum_{j \in I_i} (x_j + x_j^2),$$

where $I_i = \{k \mid k \neq i,\ \max(1, i-r_1) \leq k \leq \min(n, i+r_2)\}$

and r1, r2, k1, k2 and k3 are given integers.

Order: n = 20, 30.

Initial guess: $x_i = -1$, $i = 1,2,\ldots,n$.

We distinguish between the following cases:

0. r1 = 3 , r2 = 3 , k1 = 1 , k2 = 1 , k3 = 1 ;
1. r1 = 5 , r2 = 1 , k1 = 1 , k2 = 1 , k3 = 1 ;
2. r1 = 5 , r2 = 5 , k1 = 2 , k2 = 1 , k3 = 1 ;
3. r1 = 3 , r2 = 2 , k1 = 3 , k2 = 2 , k3 = 1 ;
4. r1 = 4 , r2 = 4 k1 = 2 , k2 = 5 , k3 = 1 .

For reasons of brevity we do not give the solution vectors.

Remark: The Jacobian matrix of this function is a band matrix with lower band width r1 and upper band width r2.

5.2.18 (BROYDEN [11])

$$F_i(x) = (3 - kx_i)x_i + 1 - x_{i-1} - 2x_{i+1} ,$$

$$i = 2,\ldots,n-1,$$

$$F_1(x) = (3 - kx_1)x_1 + 1 - 2x_2 ,$$

$$F_n(x) = (3 - kx_n)x_n + 1 - x_{n-1},$$

where k is a given integer.

Order: $n = 5, 10, 20, 30, 40.$

Initial guess: $x_i = -1$, $i = 1, 2,\ldots,n$.

We distinguish between the following cases:

0. k = 0.1 ,
1. k = 0.5 ,
2. k = 2.0 .

For reasons of brevity we do not give the solution vectors.

Remark: The Jacobian matrix of this function is a tridiagonal matrix.

In the sequel we will denote a given testfunction by a triple (p,n,c), where p denotes the last number of the subsection in which it is defined (1 up to 18), n the number of variables (i.e. the order of the problem) and c the starting point or case. For instance, testfunction (18,20,1) denotes the testfunction given in 5.2.18, with order 20 and for k = 0.5 .

5.3. *Classification of testproblems*

We classify our testproblems according to the same rules as given in section 5.1. However, the data that we derive from our practical experience (the number of function-evaluations and iteration steps) is independent of the expensiveness of the function. Hence, we do not have to distinguish between cheap and expensive problems., this would only be necessary if we use computation time as a measure.
We obtain four sets of testfunctions.

Set T_s^e : (1,2,0) , (2,2,0) , (2,2,2) ,
(4,2,0) , (4,2,1) , (6,2,1) ,
(8,2,0) ,
(15,10,k) , k = 0,1,2 ,
(18,n,k) , n = 5,10 and k = 0,1,2 .

Set T_ℓ^e : (15,n,k) , n = 20,30,50 and k = 0,1,2 ,
(17,n,k) , n = 20,30 and k = 0,1,2,3,4 ,
(18,n,k) , n = 20,30 and k = 0,1,2 ,
(18,40,k), k = 1,2 .

Set T_s^d : (1,n,0) , n = 3,5,10 ,
(2,2,1) , (2,2,3) ,
(3,2,c) , c = 0,1,2,3 ,
(5,3,0) ,
(6,2,c) , c = 0,2 ,
(7,2,c) , c = 0,1 ,
(8,2,c) , c = 1,2 ,
(9,2,0) ,
(10,2,c), c = 0,1 ,
(11,2,c), c = 0,1,2,3 ,
(12,4,0) , (13,6,0) ,
(14,n,0) , n = 2,3,4,5,6,7,9 ,
(16,10,0).

Set T_ℓ^d : (1,n,0) , n = 15,25 ,
(16,n,0) , n = 20,30,40 ,
(18,40,0) .

Although all problems of sets T_s^e and T_ℓ^e are easily solvable in the sense of definition 5.1.1., it is not certain that all problems of sets T_s^d and T_ℓ^d are difficultly solvable since we calculated rather crude upper bounds for the number Γ (cf.(2.2.7)) and it may be possible that Γ is small enough although we could not prove it. However, the given distinction is sufficient for our purpose.
We use T_s^e as a test set for the classes of functions Ψ_{s1}^e and Ψ_{s2}^e, T_ℓ^e for $\Psi_{\ell 1}^e$, $\Psi_{\ell 2}^e$, $\Psi_{\ell 3}^e$ and $\Psi_{\ell 4}^e$, and analogously T_s^d for Ψ_{s1}^d and Ψ_{s2}^d, and T_ℓ^d for $\Psi_{\ell 1}^d$, $\Psi_{\ell 2}^d$, $\Psi_{\ell 3}^d$ and $\Psi_{\ell 4}^d$.
Furthermore, T_s^e and T_ℓ^e are used for testing the efficiency, while T_s^d and T_ℓ^d are used for testing the reliability.

6. RULES FOR USERS

In this section, we give some rules of thumb for the non-specialist user of algorithms for solving systems of nonlinear equations. In fact, we summarize the results of the preceding sections, in such a way that the user is able to classify his problem. After that, it appears from the conclusions of section 8 which algorithm will most likely solve his problem.

6.1. *Information available*

Theoretically, the use of numerical approximations to the Jacobian matrix will always slow down convergence to the solution (see BUS [14]). However, in practice the use of forward difference approximations to the elements of the Jacobian matrix will usually give as good results as evaluation of the analytical expressions. In fact, it depends on the smoothness of the function and the choice of the step length in the forward difference formula (see section 3.1).

If analytical expressions for the Jacobian matrix are available, then the user should compare the number of arithmetical operations required for evaluating the function and the number of arithmetical operations required for evaluating the analytically given Jacobian matrix.

It depends highly on the ratio between these numbers, whether it is efficient to use analytical expressions for the calculation of the

Jacobian matrix. A final ruling on this point, based on experimental results, will be deferred to the conclusions in section 8.

6.2. *The size of the problem*

The number of variables, i.e. the order of the nonlinear system, and the number of arithmetical operations required for evaluating the function, defines the size of the problem. The user should decide in which of the classes defined in section 5.1, his problem has to be placed.

6.3. *Special features of the problem*

It may be possible that the way in which the problem is formulated, will give some preference for one algorithm above another.
Properties that should be noted are:

a. are some or most of the function components linear;
b. is the evaluation of one component of the function independent of evaluation of the other components or has a lot of work to be done for all components together.

Conclusions about the effect of these properties on the efficiency and reliability of the algorithms are given in section 8.

6.4. *Solvability of the problem*

If the user can derive an upper bound for the value of Γ as defined by (2.2.7), it may be considerably simplify the process of choosing the right algorithm. If this number is less than 1 or its order of magnitude is about 1, then he should choose the most efficient algorithm. However, if the order of magnitude of the number Γ is about $1/\varepsilon$ or more, where ε denotes the precision of computation, then he might prefer the most reliable algorithm.
Unfortunately, for most practical problems, it is not possible to give a reasonable estimate of the number Γ. Since all algorithms may fail on severe problems, the best we can advise is, once the problem is classified according to the rules 6.1 up to 6.3, one should choose the most efficient algorithm for this problem. If it fails in solving the problem, then

a more reliable algorithm can be used subsequently.

6.5. *Scaling of the variables*

In practice, it appears to be desirable to scale the variables in such a way, that their order of magnitude is about the same (see section 2.3).

6.6. *The stopping criteria and source text to be used*

When one chooses a program according to the conclusions given in section 8, one should use the source text, which is given in appendix, or one can use the source text to which is referred.
However, in the last case, one should incorporate the changes mentioned in section 4. In either case the conclusions are based on the values for the input parameters as given in section 4.
Concerning the stopping criteria, the user is advised to incorporate both criteria (2.4.1) and (2.4.2) when it is not done already.

6.7. *Interpretation of results*

The user should be cautious in interpreting his results. Nor a small norm of the function, neither a small step length in the last iteration step does necessarily imply a small error in the approximate solution vector. Validation of such statements can be done only if an estimate of the value of Γ (cf.(2.2.7)) is known.

7. EVALUATION OF NUMERICAL EXPERIMENTS

7.1. *The method of evaluation*

7.1.1 Evaluation of the relative efficiency.

As is already pointed out in the introduction and in section 5.1, we use a set of easily solvable testproblems for comparing the efficiency of the various programs. In fact, we use the sets T_s^e and T_ℓ^e. We say that a program is *reasonable* if it solves all easily solvable testproblems.

Let ρ denote some easily solvable problem of the form (1.4). Let R be some program for solving problems of the form (1.4) and let the number

of iteration steps n_s, the number of function evaluations n_F and the number of evaluations of the Jacobian matrix n_J, which are needed for solving problem ρ by program R, be obtained experimentally. Then, the total amount of work which has to be done by program R in order to solve problem ρ can be defined by:

$$(7.1.1.1) \qquad \Lambda(R,\rho) = \Lambda_i + n_s \times \Lambda_s + n_F \times \Lambda_F + n_J \times \Lambda_J,$$

where Λ_i and Λ_s denotes the work done in the initializing phase and per iteration step respectively (cf. section 4.4) and Λ_F and Λ_J denote the work needed to evaluate the function and its Jacobian, respectively. Hence, we say that program R is more efficient than program Q for solving problem ρ if

$$\Lambda(R,\rho) < \Lambda(Q,\rho).$$

Note that, for reasonable programs, we may assume that the numbers n_s, n_F and n_J are finite, since ρ is easily solvable (compare section 5.1).

Clearly, formula (7.1.1.1) is not very useful for our purpose since we should solve the problem before we can compute $\Lambda(R,\rho)$ and we like to know the efficiency of a program for solving some problem before we do actually solve it, in order to be able to choose the most efficient program. Therefore we will define the notion "expected relative efficiency". Let Φ be some class of easily solvable problems and suppose T is a representative set of testproblems from the class Φ. Let, moreover, Δ be a class of reasonable programs for solving problems from class Φ. Then we obtain experimentally the number n_s, n_F and n_J for all programs $R \in \Delta$ and all problems $\rho \in \Phi$. Therefore, we obtain

$$\Lambda(R,\rho) \text{ for all } R \in \Delta \text{ and } \rho \in \Phi,$$

provided Λ_i, Λ_s, Λ_F and Λ_J are known.
Then, the *expected relative efficiency* of program $R \in \Delta$, for solving a problem of class Φ is defined to be:

$$(7.1.1.2) \quad E(R,\Delta,T,\Phi) = \frac{1}{\ell} \sum_{\rho \in T} \frac{\Lambda(R,\rho)}{\max_{Q \in \Delta}(\Lambda(Q,\rho))},$$

where ℓ denotes the number of testproblems in T. Obviously, there remains the problem of measuring Λ_i, Λ_s, Λ_F and Λ_J. As we did before, we will express them in basic arithmetical operations (additions plus multiplications) (see section 4.4).
Since Λ_F and Λ_J are usually related, we express this relation by

$$(7.1.1.3) \quad \Lambda_J = \gamma \Lambda_F.$$

We will use only rough estimates for the quantities Λ_F, Λ_i and Λ_s since precise values are highly dependent on the way of programming. How we estimate these values depends on the kind of problems that are involved. We distinguish between the classes of easily solvable problems defined in section 5.1.

Class Ψ^e_{s1}. The quantities Λ_i, Λ_s and Λ_F are all small for all programs and all problems in this class. Hence, the expected relative efficiency is always acceptable. Therefore, the approximated expected relative efficiencies of all programs for solving a problem of class Ψ^e_{s1} are defined to be equally high:

$$(7.1.1.4) \quad \bar{E}(R,\Delta,\Psi^e_{s1},\Psi^e_{s1}) = c,$$

for all $R \in \Delta$, where Δ is some set of reasonable programs, c is some value between 0 and 1 and the bar above E denotes that it is an approximated value. Note that the quantity in (7.1.1.4) does not depend on a set of testproblems. In fact, only the reliablility of a program is important if one wants to solve a problem of class Ψ^e_{s1},

Class Ψ^e_{s2}. For these problems we may neglect Λ_s and Λ_i relative to Λ_F. Hence, we may approximate $\Lambda(R,\rho)$ by:

$$(7.1.1.5) \quad \Lambda(R,\rho) \simeq \bar{\Lambda}(R,\rho) = (n_F + n_J \times \gamma) \times \Lambda_F, \text{ for } R \in \Delta \text{ and } \rho \in \Psi^e_{s2}.$$

We obtain:

$$(7.1.1.6)\quad \bar{E}(R,\Delta,T_s^e,\psi_{s2}^e) = \frac{1}{\ell} \sum_{\rho\in T_s^e} \frac{\bar{\Lambda}(R,\rho)}{\max_{Q\in\Delta}(\bar{\Lambda}(Q,\rho))}.$$

Classes $\psi_{\ell 1}^e$, $\psi_{\ell 2}^e$, $\psi_{\ell 3}^e$ and $\psi_{\ell 4}^e$. For the problems we express Λ_F as a function of the number of variables n and neglect lower order terms (cf. section 5.1).

$$(7.1.1.7)\quad \bar{\Lambda}_F = \alpha n^{\beta},$$

for some integer β and real α. For the quantities Λ_i and Λ_s we use the approximations given in table 4.1. Doing so we obtain for $\rho \in T_\ell^e$:

$$(7.1.1.8)\quad \bar{\Lambda}(R,\rho) = [n_s/3 + (n_F + \gamma n_J)\alpha n^{\beta-3}]n^3, \text{ for } R \in \{A,B\},$$

$$(7.1.1.9)\quad \bar{\Lambda}(R,\rho) = [n_s/3 + \alpha n_F n^{\beta-3}]n^3, \text{ for } R \in \{C,D,E,F,G,J,L,M\},$$

$$(7.1.1.10)\quad \bar{\Lambda}(H,\rho) = [3 + \alpha n_F n^{\beta-3}]n^3,$$

$$(7.1.1.11)\quad \bar{\Lambda}(R,\rho) = [n_s n/4 + \alpha n_F n^{\beta-3}]n^3, \text{ for } R \in \{I,O\},$$

$$(7.1.1.12)\quad \bar{\Lambda}(K,\rho) = [8/3 + \alpha n_F n^{\beta-3}]n^3,$$

$$(7.1.1.13)\quad \bar{\Lambda}(N,\rho) = [1 + n_s' + \alpha n_F n^{\beta-3}]n^3,$$

where n_s' denotes the number of iteration steps that the Jacobian matrix is reset to the inverse of the forward difference approximation. Using (7.1.1.8) up to (7.1.1.13) we obtain the approximate expected relative efficiency similarly to (7.1.1.6), where the set of testproblems is chosen to be T_ℓ^e.

7.1.2. Evaluation of the reliability

In order to obtain a measure for the reliability we use the set T^d of testfunctions. The reliability of a program is simply obtained by counting the number of failures when solving problems of the testset. Let Φ be some class of difficultly solvable problems and let T be a representable set of testproblems from Φ, then the reliability of a program R

for solving a problem of class Φ is defined to be

(7.1.2.1) $Z(R,T,\Phi) = \frac{1}{\ell} \times$ (number of problems $\rho \in T$ which are solved successfully),

where ℓ is the total number of testproblems in T. We distinguish between:
a. the reliability for small problems: $Z(R,T_s^d,\Psi_s^d)$;
b. the reliability for large problems: $Z(R,T_\ell^d,\Psi_\ell^d)$;
c. the reliability for all problems: $Z(R,T^d,\Psi^d)$.

7.1.3. General remarks

All experiments reported in the next sections are carried out on a CDC Cyber 73 computer, with precision of arithmetic of about 14 digits. The values of the control parameters used are reported in section 4, where the programs are described. In the tables we give the numbers n_s, n_F, n_J (programs A and B) and n_s' (program N). These are the smallest numbers, so that the euclidean norm of the function vector is less than some threshold. We chose this threshold 10^{-7} for the small testproblems ($n \leq 15$) and 10^{-6} for the large testproblems ($n > 15$). The testing on the convergence behaviour of the various programs and on special features of some programs is reported in section 7.3.

7.2. *Efficiency experiments*

As is already mentioned in section 4.1, we distinguish between programs written in ALGOL 60 and those written in FORTRAN. The results of the ALGOL 60 programs for small and large problems are listed in the tables 7.1 and 7.2, respectively, and those of the FORTRAN programs in the tables 7.3 and 7.4. Concerning programs I and O, we assume that we may say that n evaluations of function components are equal to one evaluation of the function vector, so that n_F is equal to the total number of function component evaluations divided by n. For program L, we did in fact give the number of linear systems solved instead of n_s. In many iteration steps of this program two linear systems are solved because of some recovery scheme. However, since the solution of a linear system is the bulk of

TABLE

Experimental results for small problems of set T_s^e

Problem			A			B			C		D	
p	n	c	n_s	n_F	n_J	n_s	n_F	n_J	n_s	n_F	n_s	n_F
1	2	0	1	2	1	1	2	1	1	4	1	4
2	2	0	24	25	24	8	17	8	27	76	8	33
2	2	2	9	10	9	8	13	8	8	25	7	26
4	2	0	4	5	4	4	5	4	4	13	4	13
4	2	1	5	6	5	3	9	3	5	16	3	15
6	2	1	6	7	6	5	10	5	6	19	5	20
8	2	0	2	3	2	2	3	2	2	7	2	7
15	10	0	3	4	3	3	4	3	3	34	3	34
15	10	1	3	4	3	3	4	3	3	34	3	34
15	10	2	3	4	3	3	4	3	3	34	3	34
18	5	0	3	4	3	3	4	3	3	19	3	19
18	5	1	3	4	3	3	4	3	3	19	3	19
18	5	2	4	5	4	4	5	4	4	25	4	25
18	10	0	4	5	4	4	5	4	4	45	4	45
18	10	1	4	5	4	4	5	4	4	45	4	45
18	10	2	4	5	4	4	5	4	4	45	4	45

7.1

and all programs in ALGOL 60

E		F		Ga		Gb		H		I	
n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F
1	4	1	4	9	12	1	4	3	6	1	2.5
10	31	9	28	9	12	13	16	12	15	6	15
8	25	5	17	11	14	12	15	23	26	9	22.5
4	13	3	10	7	10	6	9	7	10	4	10
5	16	3	10	25	28	11	14	10	13	10	25
6	19	7	21	10	13	17	20	10	13	8	20
2	7	2	8	4	7	2	5	3	6	1	2.5
3	34	1	15	5	16	4	15	4	15	3	19.5
3	34	1	15	6	17	4	15	4	15	3	19.5
3	34	1	16	6	17	4	15	4	15	3	19.5
3	19	2	20	9	15	6	12	6	12	3	12
3	19	2	20	9	15	6	12	6	12	4	16
4	25	3	26	9	15	9	15	9	15	4	16
4	45	2	40	15	26	9	20	9	20	4	26
4	45	2	37	17	28	7	18	8	19	4	26
4	45	2	38	12	23	10	21	9	20	4	26

TABLE

results for large problems of

Problem			A			B			C		D	
p	n	c	n_S	n_F	n_J	n_S	n_F	n_J	n_S	n_F	n_S	n_F
15	20	0	3	4	3	3	4	3	3	64	3	64
15	20	1	3	4	3	3	4	3	3	64	3	64
15	20	2	3	4	3	3	4	3	3	64	3	64
15	30	0	3	4	3	3	4	3	3	94	3	94
15	30	1	3	4	3	3	4	3	3	94	3	94
15	30	2	3	4	3	3	4	3	3	94	3	94
15	50	0	3	4	3	3	4	3	3	154	3	154
15	50	1	3	4	3	3	4	3	3	154	3	154
15	50	2	3	4	3	3	4	3	3	154	3	154
17	20	0	3	4	3	3	4	3	4	85	4	85
17	20	1	4	5	4	4	5	4	4	85	4	85
17	20	2	4	5	4	4	5	4	4	85	4	85
17	20	3	5	6	5	5	6	5	5	106	5	106
17	20	4	5	6	5	5	6	5	5	106	5	106
17	30	0	4	5	4	4	5	4	4	125	4	125
17	30	1	4	5	4	4	5	4	4	125	4	125
17	30	2	4	5	4	4	5	4	4	125	4	125
17	30	3	5	6	5	5	6	5	5	156	5	156
17	30	4	5	6	5	5	6	5	5	156	5	156
18	20	0	5	6	5	4	8	4	5	106	4	88
18	20	1	4	5	4	4	5	4	4	85	4	85
18	20	2	4	5	4	4	5	4	4	85	4	85
18	30	0	5	6	5	4	8	4	5	156	4	128
18	30	1	4	5	4	4	5	4	4	125	4	125
18	30	2	4	5	4	4	5	4	4	125	4	125
18	40	1	4	5	4	4	5	4	4	165	4	165
18	40	2	4	5	4	4	5	4	4	165	4	165

1) norm of function only $7_{10}{}^{-5}$; 2) norm of function only $2_{10}{}^{-6}$;

7.2

set T^e_ℓ and all programs in ALGOL 60

E		F		Ga		Gb		H		I	
n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F
3	64	1	25	6	27	4	25	4	25	3	34.5
3	64	1	25	6	27	4	25	4	25	3	34.5
3	64	1	27	6	27	5	26	5	26	3	34.5
3	94	1	36	7	38	5	36	5	36	3	49.5
3	94	1	36	6	37	4	35	4	35	3	49.5
3	94	1	37	7	38	6	37	5	36	3	49.5
3	154	1	56	7	58	4	55	4	55	3	79.5
3	154	1	56	6	57	4	55	4	55	3	79.5
3	154	1	57	7	58	5	56[4)]	5	56	3	79.5
3	64	2	71	8	29	8	29	6	27	4	46
4	85	2	71	19	40	8	29	7	28	4	46
4	85	2	71	8	29	8	29	7	28	4	46
5	106	2	73	15	36	14	35	13	34	5	57.5
5	106	2	73	11	D	11	D	13	34	5	57.5
3	94	1	74	7	38	7	38	6	37	4	66
4	125	1	74	22	53	9	40	7	38	4	66
4	125	2	105	9	40	9	40	8	39	4	66
5	156	2	106	15	46	14	45	13	44	5	82.5
5	156	2	106	16	D	16	D	13	44	5	82.5
5	106	2	79	24	45[1)]	13	34	11	32	5	57.5
4	85	2	71	21	42[2)]	9	30	8	29	4	46
4	85	2	71	11	32	10	31	8	29	4	46
5	156	3	148	12	D	9	D	13	44	5	82.5
4	125	2	105	22	52[3)]	11	42	8	39	4	66
4	125	2	105	11	42	11	42	8	39	4	66
4	165	1	97	23	64	12	53	9	50	4	86
4	165	1	97	11	52	11	52	8	49	4	86

3) norm of function only $5_{10}-6$; 4)norm of function only $2_{10}-6$

TABLE

Experimental results for small problems

Problem			Ja		Jb		Ka		Kb		L	
p	n	c	n_S	n_F	n_S	n_F	n_S	n_F	n_S	n_F	n_S	n_F
1	2	0	1	4	8	25	1	4	1	4	20	24
2	2	0	8	27	24	73	8	13	12	18	2	D
2	2	2	8	27	32	97	18	22	12	16	65	63
4	2	0	4	13	4	13	9	12	7	11	18	16
4	2	1	3	13	4	15	10	13	9	12	4	D
6	2	1	5	18	6	19	7	10	8	12	14	11
8	2	0	2	7	50	T	2	5	3	7	5	8
15	10	0	3	34	3	34	5	16	4	18	8	15
15	10	1	3	34	4	45	6	19	4	23	8	15
15	10	2	3	34	3	34	5	18	4	24	10	16
18	5	0	3	19	5	31	6	14	5	15	13	18
18	5	1	3	19	4	25	8	14	6	15	13	18
18	5	2	4	25	5	31	36	D	8	19	19	21
18	10	0	4	45	8	89	10	25	9	33	21	32
18	10	1	4	45	4	45	13	27	7	28	17	30
18	10	2	4	45	5	56	61	T	8	30	23	33

7.3

of set T_s^e and all programs in FORTRAN

M		Na			Nb			Oa		Ob	
n_s	n_F	n'_s	n_s	n_F	n'_s	n_s	n_F	n_s	n_F	n_s	n_F
9	12	1	7	10	1	7	10	1	2.5	2	5
15	18	1	11	14	1	11	14	6	15	6	15
17	20	1	11	15	1	11	14	9	22.5	9	22.5
7	10	1	8	12	1	11	14	4	10	4	10
6	12	1	11	15	1	11	14	18	45	13	33
10	13	1	9	12	1	9	12	8	20	8	20
15	22	1	11	14	1	11	14	1	2.5	2	5
4	15	1	4	16	1	4	15	2	13	3	19.5
6	17	1	4	16	1	4	15	3	19.5	3	19.5
5	16	1	5	17	1	5	16	3	19.5	3	19.5
8	14	1	8	15	1	8	14	3	12	3	12
6	12	1	6	13	1	6	12	3	12	3	12
10	16	1	10	19	1	9	15	4	16	4	16
13	24	1	12	25	1	11	22	4	26	4	26
9	20	1	9	22	1	9	20	4	26	4	26
11	22	1	12	28	1	11	23	4	26	4	26

TABLE
Experimental results for large problems

Problem			Ja		Jb		Ka		Kb		L	
p	n	c	n_S	n_F	n_S	n_F	n_S	n_F	n_S	n_F	n_S	n_F
15	20	0	3	64	3	64	5	27	4	38	8	25
15	20	1	3	64	5	106	7	31	4	44	8	25
15	20	2	3	64	4	85	6	30	5	44	10	26
15	30	0	3	94	3	94	5	38	5	62	8	35
15	30	1	3	94	3	156	7	42	4	64	8	35
15	30	2	3	94	4	125	6	40	5	62	10	36
15	50	0	3	154	3	154	6	60	5	99	8	55
15	50	1	3	154	5	256	6	59	5	106	8	55
15	50	2	4	205	4	205	7	62	5	105	10	56
17	20	0	4	85	4	85	26	62	7	40	15	69
17	20	1	4	85	4	85	101	T	7	41	15	49
17	20	2	4	85	4	85	23	60	9	47	14	69
17	20	3	5	106	6	127	82	166	14	61	29	56
17	20	4	5	106	5	106	36	93	15	58	27	75
17	30	0	4	125	4	125	28	82	7	58	15	99
17	30	1	4	125	4	125	65	180	7	61	17	70
17	30	2	4	125	4	125	26	79	11	73	16	100
17	30	3	5	156	6	187	151	T	14	93	29	76
17	30	4	5	156	5	156	115	T	17	98	23	103
18	20	0	4	86	9	190	90	T	10	51	45	T
18	20	1	4	85	4	85	37	106	8	49	17	50
18	20	2	4	85	4	85	107	201	9	55	23	53
18	30	0	4	126	9	280	60	187	14	80	50	T
18	30	1	4	125	4	125	101	T	9	70	17	70
18	30	2	4	125	4	125	124	T	11	75	23	73
18	40	1	4	165	4	165	48	D	10	94	19	91
18	40	2	4	165	4	165	208	T	12	101	21	92

7.4

of set T_{ℓ}^{e} and all programs in FORTRAN

M		Na			Nb			Oa		Ob	
n_s	n_F	n_s	n_s	n_F	n_s	n_s	n_F	n_s	n_F	n_s	n_F
5	26	1	4	25	1	4	25	3	34.5	3	34.5
6	27	1	4	26	1	4	25	3	34.5	3	34.5
6	27	1	5	27	1	6	27	3	34.5	3	34.5
5	36	1	5	37	1	5	36	3	49.5	3	49.5
7	38	1	4	35	1	4	35	3	49.5	3	49.5
7	38	1	6	38	1	6	37	3	49.5	3	49.5
5	56	1	5	57	1	5	56	3	79.5	3	79.5
7	58	1	4	55	1	4	55	3	79.5	3	79.5
7	58	1	6	58	1	6	57	3	79.5	3	79.5
6	27	1	8	31	1	7	28	4	46	4	46
7	28	1	8	30	1	8	29	4	46	4	46
8	29	1	8	30	1	8	29	4	46	4	46
14	35	1	17	45	1	14	35	4	46	5	57.5
14	35	1	17	46	1	13	34	5	57.5	5	57.5
6	37	1	7	39	1	7	38	4	66	4	66
8	39	1	8	40	1	8	39	4	66	4	66
8	39	1	9	42	1	8	39	4	66	4	66
14	45	1	17	55	1	14	45	4	66	5	82.5
14	45	1	16	53	1	14	45	5	82.5	5	82.5
11	52	1	15	35	1	15	36	5	57.5	5	57.5
9	30	1	10	32	1	10	31	4	46	4	46
10	31	1	10	34	1	9	30	4	46	4	46
18	49	1	18	51	1	18	49	5	82.5	5	82.5
10	41	1	12	46	1	10	41	4	66	4	66
10	41	1	10	43	1	10	41	4	66	4	66
10	51	1	12	56	1	11	52	4	86	4	86
10	51	1	10	53	1	10	51	4	86	4	86

the work per iteration step, it is convenient to count the solution of a linear system as an iteration step. In the tables the capital D means that the program diverged and is terminated by some error exit, T means that the program is terminated because the number of function evaluations became too high. It is clear from table 7.2 that program G is not a reasonable program for solving large functions, since it failed to solve some easily solvable problems and other problems were not solved in the precision required. For the same reasons we see from tables 7.3 and 7.4 that the programs L and Ka are not reasonable at all for solving nonlinear systems, while program Jb should not be used for small problems. This result for program Ka is rather surprising. We feel that the starting guesses in program Ka should give better results than those given in program Kb. This is affirmed by the fact that for many problems the recovery scheme built in program K is used to obtain a new set of starting guesses when using version Kb. Probably, there are some small programming errors in the code published by GRAGG & STEWART [28]. Another simple conclusion that can be derived from tables 7.3 and 7.4 is that the number of function evaluations as well as the number of iteration steps, needed by program Jb for solving the given problems is always greater or equal to those, needed by program Ja. For this reason and for the one given above, we will also consider program Jb as not reasonable. These conclusions will also be justified by the reliability tests given in section 7.3.

Using the results given in tables 7.1 up to 7.4 we will now calculate the values for the approximate expected relative efficiency of the various procedures for solving the problems from the various classes. For this calculation we use the notions and formulas from section 7.1.

7.2.1. Efficiency for solving small cheap problems

As is already stated in section 7.1, we define the approximated expected relative efficiency of all reasonable programs for solving problems of class Ψ^{e}_{s1} equally high (cf.(7.1.1.4)). Only the reliability of the various programs is important if one wants to solve these problems.

7.2.2. Efficiency for solving small expensive problems

In order to evaluate the right hand side of (7.1.1.6) we should know the value of γ (see (7.1.1.3)). In table 7.5 we give the approximated expected relative efficiency of the various reasonable ALGOL 60 programs for solving problems of class Ψ^e_{s2}, for some typical values of γ. Since for all FORTRAN programs the value of γ is equal to zero, we can give the required results in table 7.6 independent of γ.
In our notation Δ_A means the set of reasonable ALGOL 60 programs:

(7.2.2.1) $\Delta_A = \{A,B,C,D,E,F,Ga,Gb,H,I\}$

and Δ_F denotes the set of reasonable FORTRAN programs:

(7.2.2.2) $\Delta_F = \{Ja,Kb,M,Na,Nb,Oa,Ob\}$.

TABLE 7.5

$\bar{E}(R,\Delta_A,T^e_s,\Psi^e_{s2})$, for $R \in \Delta_A$

γ \ R	A	B	C	D	E	F	Ga	Gb	H	I
1	0.4	0.4	0.9	0.9	0.9	0.7	0.6	0.5	0.5	0.6
2	0.5	0.5	0.9	0.9	0.9	0.7	0.6	0.5	0.5	0.6
5	0.8	0.8	0.8	0.8	0.7	0.6	0.6	0.4	0.4	0.5
n/2	0.6	0.6	0.9	0.9	0.9	0.7	0.6	0.5	0.5	0.6
n	0.9	0.9	0.9	0.9	0.9	0.7	0.6	0.5	0.5	0.6
2n	1.0	0.9	0.6	0.5	0.5	0.4	0.4	0.3	0.3	0.4

TABLE 7.6

$\bar{E}(R,\Delta_F,T^e_s,\Psi^e_{s2})$, for $R \in \Delta_F$

Ja	Kb	M	Na	Nb	Oa	Ob
0.9	0.6	0.6	0.6	0.6	0.6	0.6

As an immediate result of table 7.5 we see that programs A or B (Newton's method with analytic Jacobian) is only preferable above other algorithms if γ is small (about 1).
Furthermore, programs C, D and Ja (Newton's method with forward difference Jacobian) is not efficient.

7.2.3. Efficiency for solving large very cheap problems

To evaluate the approximated expected relative efficiency of the reasonable programs for solving problems of class $\Psi^e_{\ell 1}$ we substitute the values for n_s, n_F and n_J given in tables 7.2 and 7.4 in the expressions (7.1.1.8) up to (7.1.1.13) where $\beta = 1$. However, since the first term within the brackets of these expressions is of order 1 or more and the second term is of order n_F/n^2 we can neglect the second term for large n.
Doing so, we obtain with the use of a formula similar to (7.1.1.6) the results given in tables 7.7 and 7.8. These results are independent of α and γ since they only appear in the terms that we neglected.
Since the programs Ga and Gb are considered to be not reasonable for solving large problems, we will drop them and use the set $\bar{\Delta}_A$ of reasonable programs in ALGOL 60, where

(7.2.3.1) $\quad \bar{\Delta}_A = \{A,B,C,D,E,F,H,I\}.$

TABLE 7.7

$\bar{E}(R,\bar{\Delta}_A,T^e_\ell,\Psi^e_{\ell 1})$, for $R \in \bar{\Delta}_A$

A	B	C	D	E	F	H	I
0.05	0.05	0.05	0.05	0.05	0.02	0.1	1

TABLE 7.8

$\bar{E}(R,\Delta_F,T^e_\ell,\Psi^e_{\ell 1})$, for $R \in \Delta_F$

Ja	Kb	M	Na	Nb	Oa	Ob
0.05	0.1	0.1	0.05	0.05	1	1

Clearly program F is the most efficient program in ALGOL 60 and the programs I in ALGOL 60 and O in FORTRAN are relatively very inefficient for solving large very cheap problems.

7.2.4. Efficiency for solving large cheap problems

As in section 7.2.3 we substitute the values for n_s, n_F and n_J, given in tables 7.2 and 7.4, in the expressions (7.1.1.8) up to (7.1.1.13), where $\beta = 2$. However, we can no longer neglect the second term in these expressions since n_F is usually of order n. Therefore, substituting n and β, there still remain two parameters α and γ (see (7.1.1.7) and (7.1.1.3) respectively). In table 7.9 we list the values for the approximated expected relative efficiencies of the programs in ALGOL 60 for some typical values of α (α=1,20) and γ (γ=1/n, 1,n). Since $\gamma = 0$ for all programs in FORTRAN, the results for these programs, given in table 7.10 depend only on α.

TABLE 7.9

$E(R,\bar{\Delta}_A,T^e_\ell,\Psi^e_{\ell 2})$, for $R \in \bar{\Delta}_A$ and some typical values of γ and α

α	γ \ R	A	B	C	D	E	F	H	I
1	1/n	0.06	0.06	0.2	0.2	0.2	0.1	0.2	1
1	1	0.06	0.06	0.2	0.2	0.2	0.1	0.2	1
1	n	0.2	0.2	0.2	0.2	0.2	0.1	0.2	1
20	1/n	0.1	0.1	1	1	1	0.6	0.4	0.9
20	1	0.1	0.1	1	1	1	0.6	0.4	0.9
20	n	1	1	1	1	1	0.6	0.4	0.9

TABLE 7.10

$E(R,\Delta_F,T^e_\ell,\Psi^e_{\ell 2})$, for $R \in \Delta_F$ and some values of α

α \ R	Ja	Kb	M	Na	Nb	Oa	Ob
1	0.2	0.2	0.2	0.09	0.08	1	1
20	1	0.6	0.4	0.4	0.3	0.8	0.9

It is easily seen from table 7.9 that Newton's method with analytical derivatives (programs A and B) is the most efficient method as long as evaluation of the Jacobian matrix is about as expensive as one evaluation of the function or cheaper. In all other cases, program F (if α=1) or program H (if α=20) is preferable when a program in ALGOL 60 has to be chosen. From table 7.10 we see that the most efficient FORTRAN program is program Nb, for both values of α.

7.2.5. Efficiency for solving large expensive problems

In a similar way as in section 7.2.4 we obtain the results given in table 7.11 and 7.12. For this class of functions $\beta = 3$ (cf.(7.1.1.7)) is substituted.

TABLE 7.11

$E(R,\bar{\Delta}_A,T_\ell^e,\Psi_{\ell 3}^e)$, for $R \in \bar{\Delta}_A$ and some typical values of α and γ.

α	γ \ R	A	B	C	D	E	F	H	I
1	1	0.1	0.1	1	1	1	0.6	0.4	0.8
1	n	1	1	1	1	1	0.6	0.4	0.8
20	1	0.1	0.1	1	1	1	0.6	0.3	0.5
20	n	1	1	1	1	1	0.6	0.3	0.5

TABLE 7.12

$E(R,\Delta_F,T_\ell^e,\Psi_{\ell 3}^e)$, for $R \in \Delta_F$ and some values of α

α \ R	Ja	Kb	M	Na	Nb	Oa	Ob
1	1	0.6	0.4	0.4	0.3	0.8	0.8
20	1	0.6	0.4	0.4	0.4	0.5	0.5

As for large cheap problems (section 7.2.4) we see that programs A and B (Newton's method with analytical derivatives) are superior above the other programs in ALGOL 60 as long as the evaluation of the Jacobian matrix is about as expensive as one evaluation of the function or cheaper. Otherwise program H is preferred. The programs M and N are the most efficient

programs in FORTRAN.

7.2.6. Efficiency for solving large very expensive problems

In calculating the approximated expected relative efficiencies of the various programs for solving problems of class $\Psi^e_{\ell 4}$ we may simplify the expressions (7.1.1.8) up to (7.1.1.13) by neglecting the first term within the brackets relative to the second since $\beta = 4$. Therefore, the results do not depend on α. For the programs in ALGOL 60 we give the results, for $\gamma = 1$ or n, in table 7.13. For the programs in FORTRAN, where $\gamma = 0$ for all programs, the results are given in table 7.14.

TABLE 7.13

$E(R,\bar{\Delta}_A,T^e_\ell,\Psi^e_{\ell 4})$, for $R \in \Delta_A$ and some values of γ

γ \ R	A	B	C	D	E	F	H	I
1	0.1	0.1	1	1	1	0.6	0.3	0.5
n	1	1	1	1	1	0.6	0.3	0.5

TABLE 7.14

$E(R,\Delta_F,T^e_\ell,\Psi_{\ell 4})$, for $R \in \Delta_F$

Ja	Kb	M	Na	Nb	Oa	Ob
1	0.6	0.4	0.4	0.3	0.5	0.5

Again we see that the programs A and B are superior as long as the evaluation of the analytical Jacobian is about as expensive as one evaluation of the function. Otherwise, program H is the most efficient program in ALGOL 60. The programs N and M are the most efficient programs in FORTRAN.

7.3 *Reliability experiments*

Since the reliability of a program, defined by (7.1.2.1), is independent of other programs we do not have to distinguish between programs in ALGOL 60 and FORTRAN when we calculate the reliability. As is mentioned in section 4.1 and 7.1.2 we use the set T^d of testfunctions to measure the

TABLE

Experimental results for testproblems

Problem			A		B		C		D	
p	n	c	n_S	n_F	n_S	n_F	n_S	n_F	n_S	n_F
1	3	0	6	7	5	11	6	25	5	26
1	5	0	17	18	6	17	17	103	6	47
1	10	0	2	D	12	51	2	D	12	171
1	15	0	2	D	2	D	2	D	1	D
1	25	0	2	D	2	D	1	D	1	D
2	2	1	1	D	1	D	18	55	6	34
2	2	3	14	15	6	20	14	43	6	32
3	2	0	42	43	4	D	56	169	4	D
3	2	1	22	23	5	D	65	196	5	D
3	2	2	5	6	4	8	6	19	4	16
3	2	3	16	17	4	14	16	49	4	22
5	3	0	7	8	6	13	7	29	6	31
6	2	0	6	7	5	10	6	19	5	20
6	2	2	100	T	3	D	67	T	3	D
6	2	3	11	12	6	18	11	34	6	30
7	2	0	12	13	70	367	12	37	70	507
7	2	1	16	16	4	D	14	43	4	D
8	2	1	2	3	2	D	2	7	2	D
8	2	2	2	3	9	47	3	10	9	65
9	2	0	2	3	15	71	2	7	15	101

7.15

of set T^d and programs in ALGOL 60

E		F		Ga		Gb		H		I	
n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F
6	25	6	25	4	D	3 [1)]	7	10	14	6	18
20	133	2	D	4	D	1	D	9	15	17	68
1	D	2	D	1	D	1	D	56	D	76	T
1	D	2	D	1	D	1	D	37	D	83	T
1	D	2	D	1	D	1	D	4	D	1	D
8	25	16	48	1	D	1	D	97	T	6	15
7	22	19	58	12	15	11	14	17	20	6	15
25	D	8	25	98	T	98	T	97	T	11	27.5
30	D	25	D	98	T	98	T	97	T	40	T
5	16	5	16	9	12	9	12	11	14	6	15
12	37	2	D	98	T	15	18	97	T	7	17.5
7	29	7	29	27	31	18	22	13	17	7	21
6	19	7	21	12	15	8	11	0	I	6	15
26	D	7	D	16	19	55	D	97	T	12	30
9	28	11	33	62	D	63	D	33	36	12	30
12	D	10	31	23	26	24	27	26	29	12	30
11	D	12	37	26	29	26	29	70	73	13	32.5
2	7	2	8	4	7	2	5	3	6	1	2.5
2	7	2	7	2	5	2	5	4	7	1	2.5
2	7	2	6	5	8	3	6	3	6	2	5

1) norm of function only $8.4_{10}-3$

TABLE

Problem			A		B		C		D	
p	n	c	n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F
10	2	0	6	7	100	T	9	28	44	T
10	2	1	2	3	100	T	8	25	44	T
11	2	0	15	16	15	16	14	43	14	43
11	2	1	13	14	13	14	13	40	13	40
11	2	2	15	16	4	D	15	46	4	D
11	2	3	17	18	3	D	17	52	3	D
12	4	0	19	20	19	20	19	96	19	96
13	6	0	6	7	6	7	6	43	6	43
14	2	0	-	-	-	-	4	13	4	13
14	3	0	-	-	-	-	4	17	4	17
14	4	0	-	-	-	-	6	31	5	31
14	5	0	-	-	-	-	5	31	5	34
14	6	0	-	-	-	-	8	D	5	39
14	7	0	-	-	-	-	6	D	6	59
14	9	0	-	-	-	-	4	D	5	D
16	10	0	8	9	8	9	8	89	8	89
16	20	0	100	T	7	13	96	T	7	153
16	30	0	6	7	6	7	6	187	6	187
16	40	0	7	8	7	8	7	288	7	288
18	40	0	5	6	4	8	4 [3]	165	4	168

2) norm of function only $2.0_{10}{}^{-5}$

3) norm of function only $3.6_{10}{}^{-4}$

4) norm of function only $1.0_{10}{}^{-4}$

7.15 (continued)

E		F		Ga		Gb		H		I	
n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F
9	28	7	21	98	T	98	T	97	T	28	70
9	28	5	17	98	T	98	T	97	T	24	60
14	43	13	44	24	27	22	25	22	25	13	32.5
13	40	11	37	6	D	14[2]	17	18	21	13	32.5
15	46	13	42	21	24	23	26	23	26	13	32.5
21	67	16	51	6	D	20	23	20	23	11	27.5
20	106	16	81	6	D	4	D	27	32	57	T
6	43	4	32	15	22	14	21	0	I	8	36
4	13	4	12	11	14	5	8	5	8	4	10
5	21	3	13	8	D	6	10	7	11	4	12
7	36	5	25	17	22	12	17	9	14	4	14
7	D	4	27	53	D	10	16	9	15	6	24
3	D	2	D	3	D	18	25	31	38	7	31.5
4	D	2	D	44	D	14	22	13	21	5	25
1	D	2	D	3	D	3	D	20	30	5	D
6	67	2	D	1	D	1	D	16	27	7	45.5
26	T	1	D	1	D	1	D	18	39	86	T
6	187	2	76	1	D	1	D	21	52	90	T
25	T	2	D	1	D	1	D	24	65	75	T
5	206	2	D	32[4]	73	6	D	15	56	5	107.5

TABLE

Experimental results for testproblems

Problem			Ja		Jb		Ka		Kb		L	
p	n	c	n_S	n_F	n_S	n_F	n_S	n_F	n_S	n_F	n_S	n_F
1	3	0	6	26	32	134	12	23	10	20	12	D
1	5	0	7	49	50	T	10	23	7	22	45	T
1	10	0	10	121	50	T	16	49	5	26	35	T
1	15	0	50	T	50	T	16	60	6	38	24	T
1	25	0	1	D	50	T	1	D	1	D	1	D
2	2	1	16	58	6	19	8	11	10	16	14	11
2	2	3	6	29	7	22	8	12	8	12	20	14
3	2	0	50	T	50	T	66	T	75	T	12	D
3	2	1	50	T	50	T	29	41	78	T	14	D
3	2	2	4	14	15	46	8	13	13	19	2	4
3	2	3	4	20	18	55	8	12	11	19	20	16
5	3	0	6	27	29	117	23	28	15	22	66	60
6	2	0	5	18	8	25	1	4	10	15	6	D
6	2	2	25	204	50	T	8	11	12	16	22	D
6	2	3	6	26	9	28	7	10	8	12	22	19
7	2	0	33	169	18	55	23	43	20	37	14	D
7	2	1	40	275	50	T	25	46	25	46	7	D
8	2	1	13	102	50	T	7	10	6	10	5	8
8	2	2	5	24	31	94	4	7	4	8	20	20
9	2	0	10	53	35	106	2	5	7	10	6	D

7.16

of set T^d and programs in FORTRAN

M		Na			Nb			Oa		Ob	
n_s	n_F	n'_s	n_s	n_F	n'_s	n_s	n_F	n_s	n_F	n_s	n_F
21	31	1	8	12	1	8	12	6	18	6	18
31	51	1	9	15	1	9	15	17	68	17	68
42	84	1	9	20	1	9	20	50	T	1	D
40	T	1	10	26	1	10	26	50	T	1	D
28	T	1	10	36	1	10	36	1	D	1	D
9	12	1	8	11	1	8	11	6	15	6	15
10	13	1	9	13	1	9	12	6	15	5	12.5
83	T	8	47	D	8	47	D	10	25	10	25
83	T	2	23	D	2	23	D	19	47.5	19	47.5
18	30	1	12	15	1	12	15	6	15	6	15
22	30	1	12	16	1	12	15	7	17.5	7	17.5
76	T	1	11	16	1	12	16	7	21	7	21
10	13	1	10	14	1	10	13	6	15	6	15
83	T	1	13	16	1	13	16	7	17.5	2	5
13	16	1	9	13	1	9	12	11	27.5	2	5
31	46	1	114	117	1	132	135	12	30	7	D
78	151	2	50	D	2	50	D	11	28	8	D
83	T	3	95	D	3	95	D	1	3	2	5
83	T	2	4	8	2	4	8	1	3	2	5
19	32	1	24	27	1	24	27	2	5	3	8

TABLE

Problem			Ja		Jb		Ka		Kb		L	
p	n	c	n_S	n_F	n_S	n_F	n_S	n_F	n_S	n_F	n_S	n_F
10	2	0	50	T	50	T	87	T	87	T	8	D
10	2	1	50	T	50	T	85	T	85	T	9	D
11	2	0	15	46	50	T	19	34	23	41	4	D
11	2	1	13	40	26	79	20	37	19	36	2	D
11	2	2	28	205	50	T	20	36	23	41	17	D
11	2	3	27	174	50	T	22	39	21	38	2	D
12	4	0	19	96	31	156	35	71	29	62	56	76
13	6	0	6	43	7	50	20	32	15	33	68	D
14	2	0	4	13	4	13	6	11	6	12	19	16
14	3	0	4	17	4	17	11	19	6	13	26	24
14	4	0	5	28	5	27	21	33	9	16	47	57
14	5	0	4	26	4	26	27	43	11	19	70	T
14	6	0	5	37	6	43	35	55	15	28	46	51
14	7	0	6	54	5	44	2	D	19	36	4	D
14	9	0	24	355	26	341	1	D	25	48	1	D
16	10	0	3	34	3	34	21	40	6	17	30	26
16	20	0	3	64	3	64	52	93	6	28	100	T
16	30	0	3	94	3	94	83	162	7	40	100	T
16	40	0	4	165	4	165	160	T	35	171	100	T
18	40	0	4	166	9	370	131	T	13	101	100	T

7.16 (continued)

M		Na			Nb			Oa		Ob	
n_s	n_F	n'_s	n_s	n_F	n'_s	n_s	n_F	n_s	n_F	n_s	n_F
83	T	1	97	T	1	97	T	16	40	15	38
83	T	1	97	T	1	97	T	17	42.5	16	40
83	T	4	134	140	4	125	131	13	32.5	10	D
20	23	1	151	154	1	152	155	13	32.5	10	D
83	T	3	11	D	3	11	D	13	32.5	10	D
83	T	1	185	188	1	179	182	11	27.5	8	D
29	34	1	57	66	1	78	83	45	T	50	T
14	35	1	30	38	1	30	37	6	27	6	27
6	9	1	5	8	1	5	8	4	10	4	10
6	10	1	7	13	1	8	14	4	12	4	12
10	18	1	11	19	1	9	14	4	14	12	42
10	16	1	11	20	1	9	15	5	20	5	20
12	28	1	31	41	1	29	36	4	18	4	18
15	31	1	19	30	1	17	25	4	20	4	20
22	44	1	28	41	1	31	41	6	36	5	30
6	17	1	7	21	1	7	18	3	19.5	3	19.5
5	26	1	6	28	1	6	27	3	34.5	3	34.5
6	37	1	8	41	1	8	39	4	66	3	49.5
8	51	1	9	51	1	9	50	7	150.5	7	150.5
19	60	1	21	65	1	20	61	5	107.5	5	107.5

reliability and we distinguish between the reliabiltiy for small problems $(Z(R,T_s^d,\Psi_s^d))$, for large problems $(Z(R,T_\ell^d,\Psi_\ell^d))$ and for all problems $(Z(R,T^d,\Psi^d))$. In order to be able to calculate these values we give in table 7.15 and 7.16 the results of the programs in ALGOL 60 and FORTRAN respectively for the set of testproblems T^d.
Besides the notation that is also used in the tables 7.1 up to 7.4 the capital I in these tables means that the program is already terminated in the initializing phase because of a singular Jacobian matrix.
As is seen in table 7.15 we do not give experimental results of programs A and B for solving the problems (14,n,0), n = 2,3,4,5,6,7,9. For these problems, the analytical Jacobian matrix is not available. Therefore, we give the reliability of the programs C up to O, which is measured with all problems in T^d, in table 7.17. Furthermore, the reliability of all programs measured with the problems in T^d except for the problems (14,n,0), n = 2,3,4,5,6,7,9, are given in table 7.18.
We use the notation:

$$\bar{T}^d = T^d \setminus \{(14,n,0),\ n = 2,3,4,5,6,7,9\}. \tag{7.3.1}$$

Since we do not pretend that T^d or $\bar{T}^d$ is really a representative set of functions for testing the reliability we do only give one significant figure in the tables 7.17 and 7.18. From the results given in these tables, we can draw some simple conclusions.
The statement, given in section 7.2, that the programs Ga, Gb, and L can not be considered as reasonable programs is affirmed by these results. Their reliability is only 0.5 or less, i.e. for at least half of the problems of T^d these programs fail. Furthermore, the programs B, D and Jb (Newton's method with some kind of step size control) are considerably less reliable than its equivalent without step size control (programs A, C and Ja, respectively). Since step size control is incorporated to increase the reliability, we must conclude that this goal is not attained and that these programs are not useful. The conclusion that program Ka is not useful is not affirmed by the figures, but as we mentioned already, the behaviour of program Ka is not clear to us and we feel that there are some small programming errors in the code published by GRAGG & STEWART [28].

TABLE 7.17

reliability of programs

R	$Z(R,T_s^d,\Psi_s^d)$	$Z(R,T_\ell^d,\Psi_\ell^d)$	$Z(R,T^d,\Psi^d)$
C	0.8	0.5	0.8
D	0.7	0.8	0.7
E	0.7	0.5	0.7
F	0.7	0.3	0.7
Ga	0.4	0.2	0.4
Gb	0.6	0.2	0.5
H	0.7	0.8	0.7
I	0.9	0.3	0.8
Ja	0.9	0.8	0.9
Jb	0.6	0.8	0.7
Ka	0.9	0.5	0.8
Kb	0.9	0.8	0.9
L	0.4	0.2	0.4
M	0.7	0.8	0.7
Na	0.8	1.0	0.8
Nb	0.8	1.0	0.8
Oa	0.9	0.8	0.9
Ob	0.7	0.8	0.8

There is another conclusion that can be derived from the tables 7.17 and 7.18. Comparing the figures for the program O we must conclude that program Oa is to be preferred. Hence, the step length used in the forward difference formulas to approximate the elements of the Jacobian matrix should not be chosen as small as 10^{-8} if the machine precision is about 10^{-14}. To summarize we may say that the ALGOL 60 programs A,C,D,E,F,H and I are useful where A,C and I are the most reliable programs; furthermore the FORTRAN programs Ja,Kb,M,N and Oa are useful, where only program M is considerably less reliable than the other programs.

TABLE 7.18

reliability of programs

R	$Z(R,\bar{T}_s^d,\Psi_s^d)$	$Z(R,\bar{T}_\ell^d,\Psi_\ell^d)$	$Z(R,\bar{T}^d,\Psi^d)$
A	0.9	0.7	0.8
B	0.6	0.8	0.6
C	0.9	0.5	0.8
D	0.6	0.8	0.7
E	0.7	0.5	0.7
F	0.7	0.3	0.7
Ga	0.4	0.2	0.4
Gb	0.5	0.2	0.4
H	0.6	0.8	0.6
I	0.9	0.3	0.8
Ja	0.8	0.8	0.8
Jb	0.5	0.8	0.6
Ka	0.9	0.5	0.8
Kb	0.9	0.8	0.9
L	0.3	0.2	0.3
M	0.6	0.8	0.6
Na	0.7	1.0	0.8
Nb	0.7	1.0	0.8
Oa	0.9	0.8	0.9
Ob	0.7	0.8	0.7

7.4 *Experiments about convergence behaviour and special features of the programs*

7.4.1. Convergence behaviour

For the programs C up to I in ALGOL 60 and Ja,Ka,Kb,M,Na,Nb,Oa in FORTRAN we give some diagrams to show the progress of the iteration as a function of the number of function evaluations. These diagrams are only illustrations of the performance of the various programs and, in fact, only for the classes of functions Ψ_{s2}^e and $\Psi_{\ell 4}^e$ where the work done per iteration step can be neglected, these diagrams are illustrations of the

relative efficiency. Nevertheless, the diagrams are typical as illustrations of the behaviour of iterative methods for solving nonlinear systems. The symbols used in these diagrams are explained by the following reference tables.

programs in ALGOL 60:

 : program C ;

 : program D ;

 : program E ;

 : program F ;

 : program Ga;

 : program Gb;

 : program H ;

 : program I ;

programs in FORTRAN:

 : program Ja;

 : program Ka;

 : program Kb;

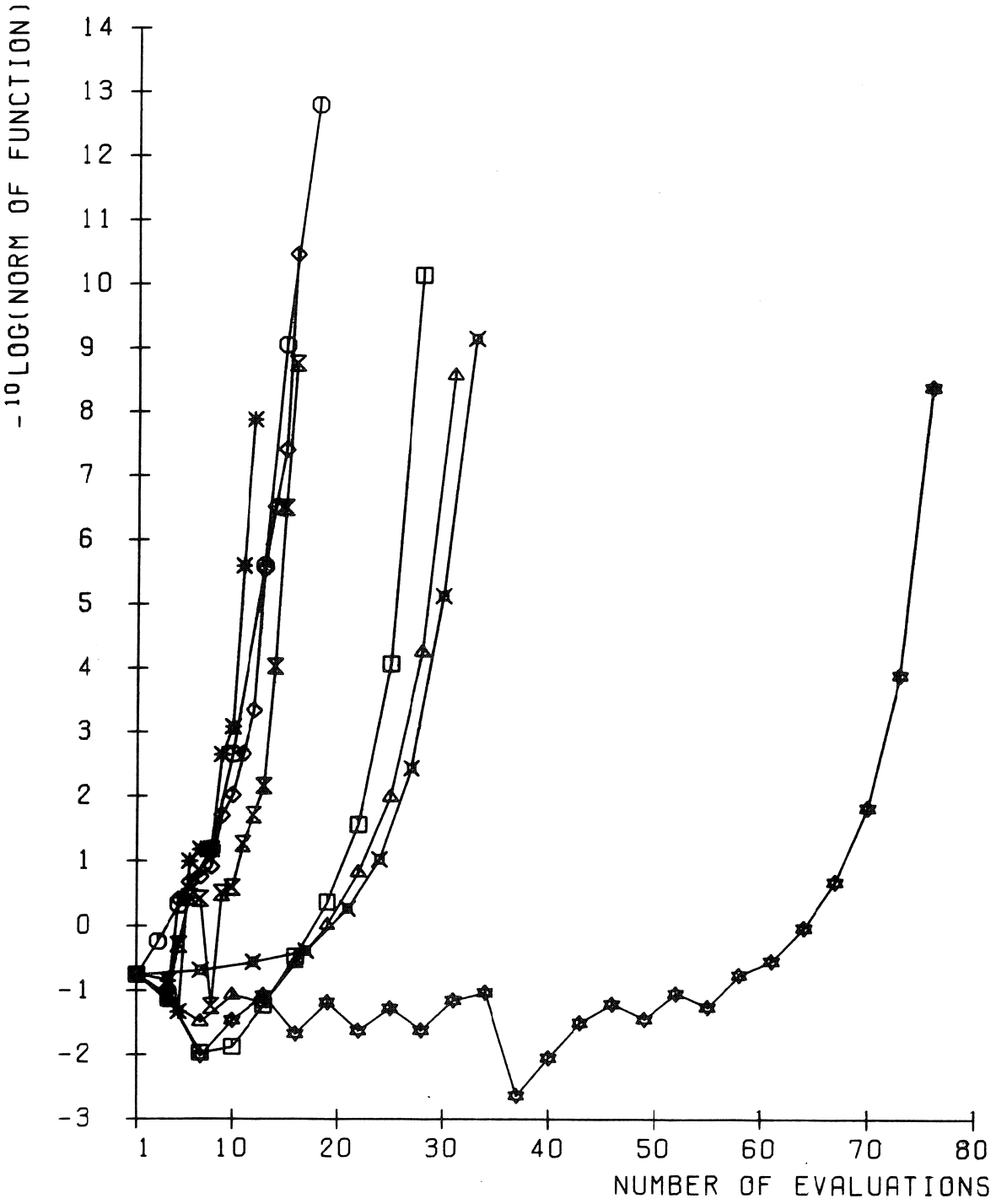
 : program M ;

 : program Na;

 : program Nb;

 : program Ob.

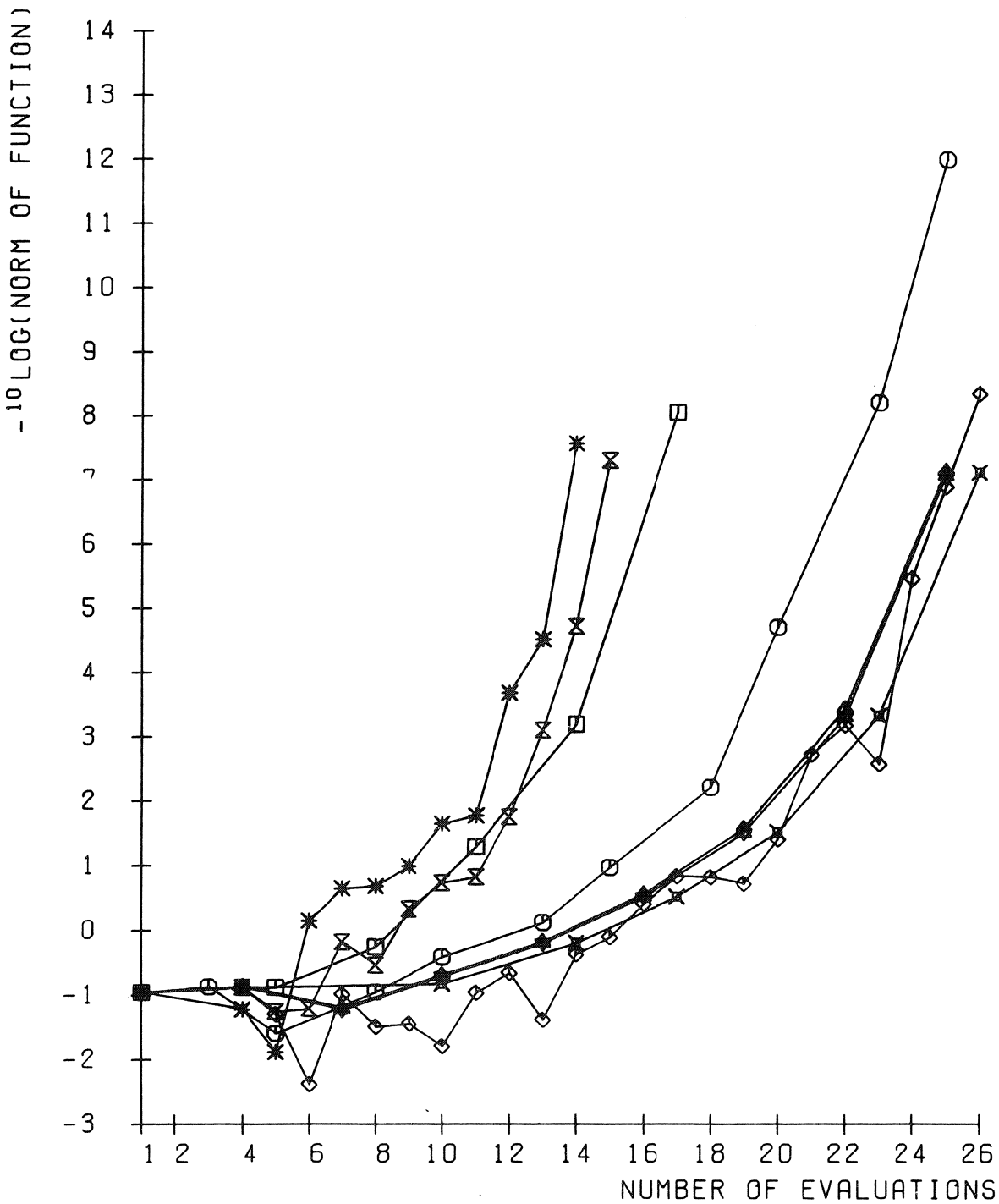
One can see from these diagrams, that, once convergence starts, it is going fast (superlinearly or even quadratically).
Furthermore, there appears to be no reason to expect that one program is is more efficient to obtain the solution in a high precision than another program. The same holds if only low precision is required.



FUNCTION 2 , ORDER 2 , CASE 0

PROGRAMS IN ALGOL 60

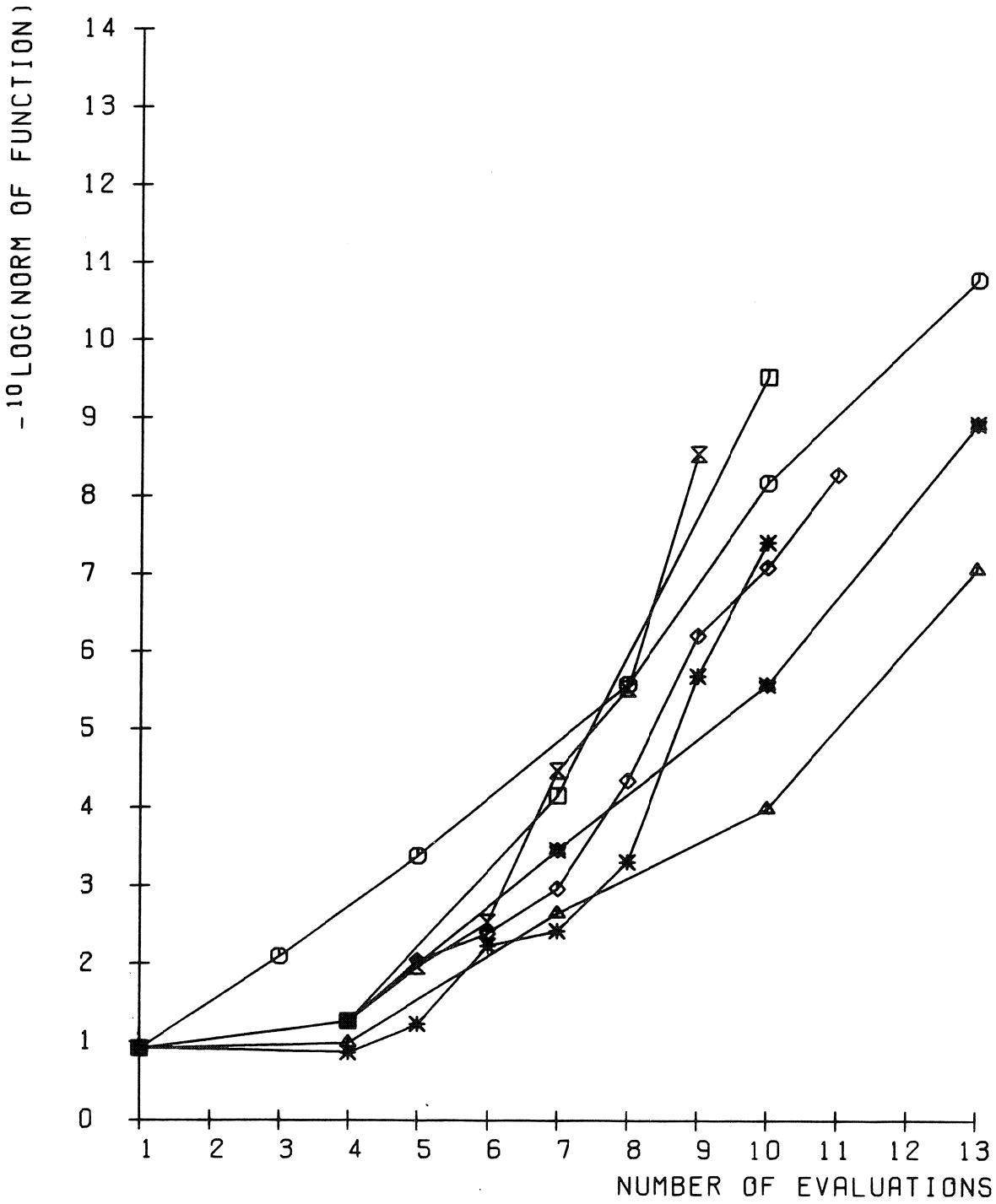
DIAGRAM 7.1



FUNCTION 2 , ORDER 2 , CASE 2

PROGRAMS IN ALGOL 60

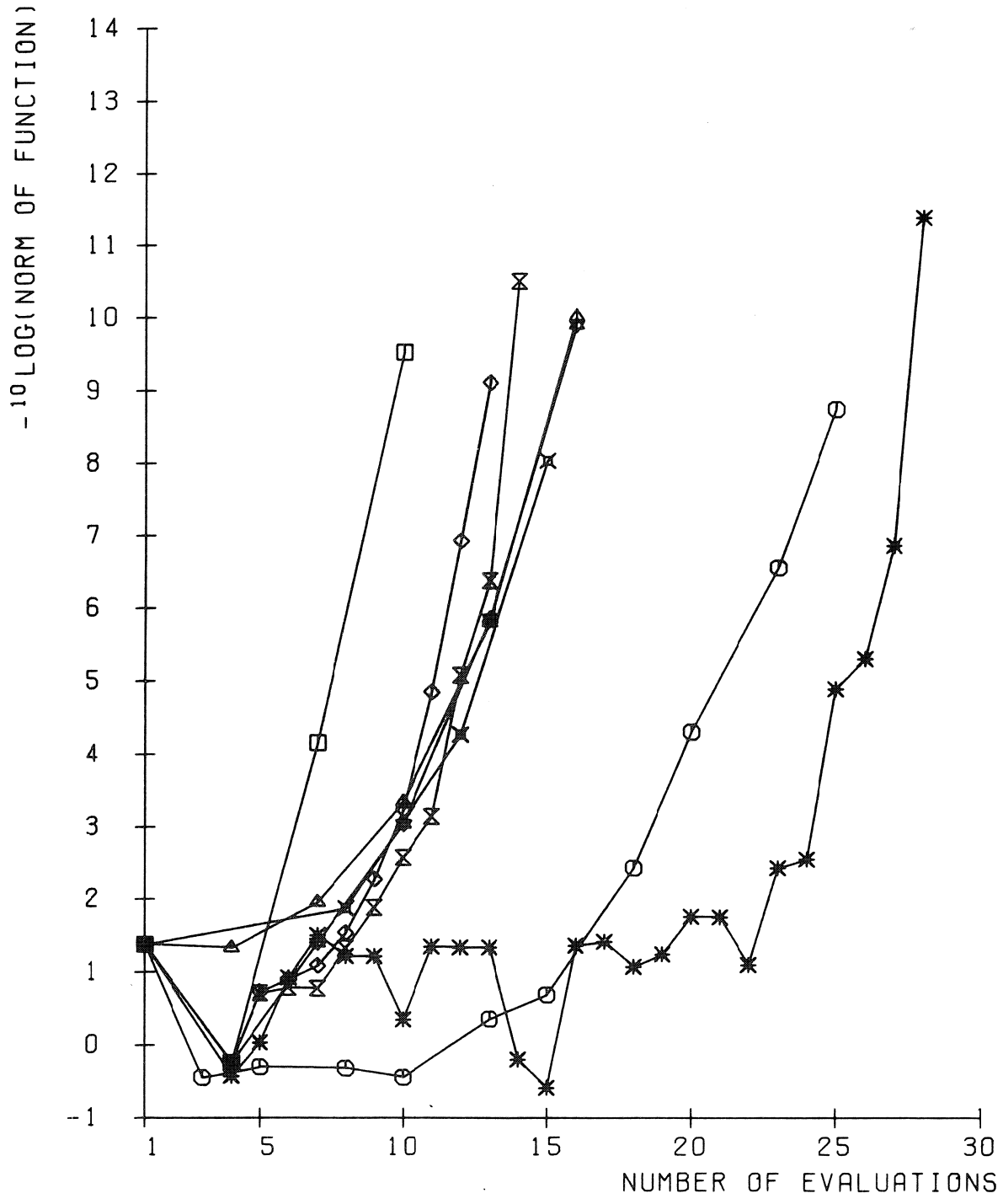
DIAGRAM 7.2



FUNCTION 4 , ORDER 2 , CASE 0

PROAGRAMS IN ALGOL 60

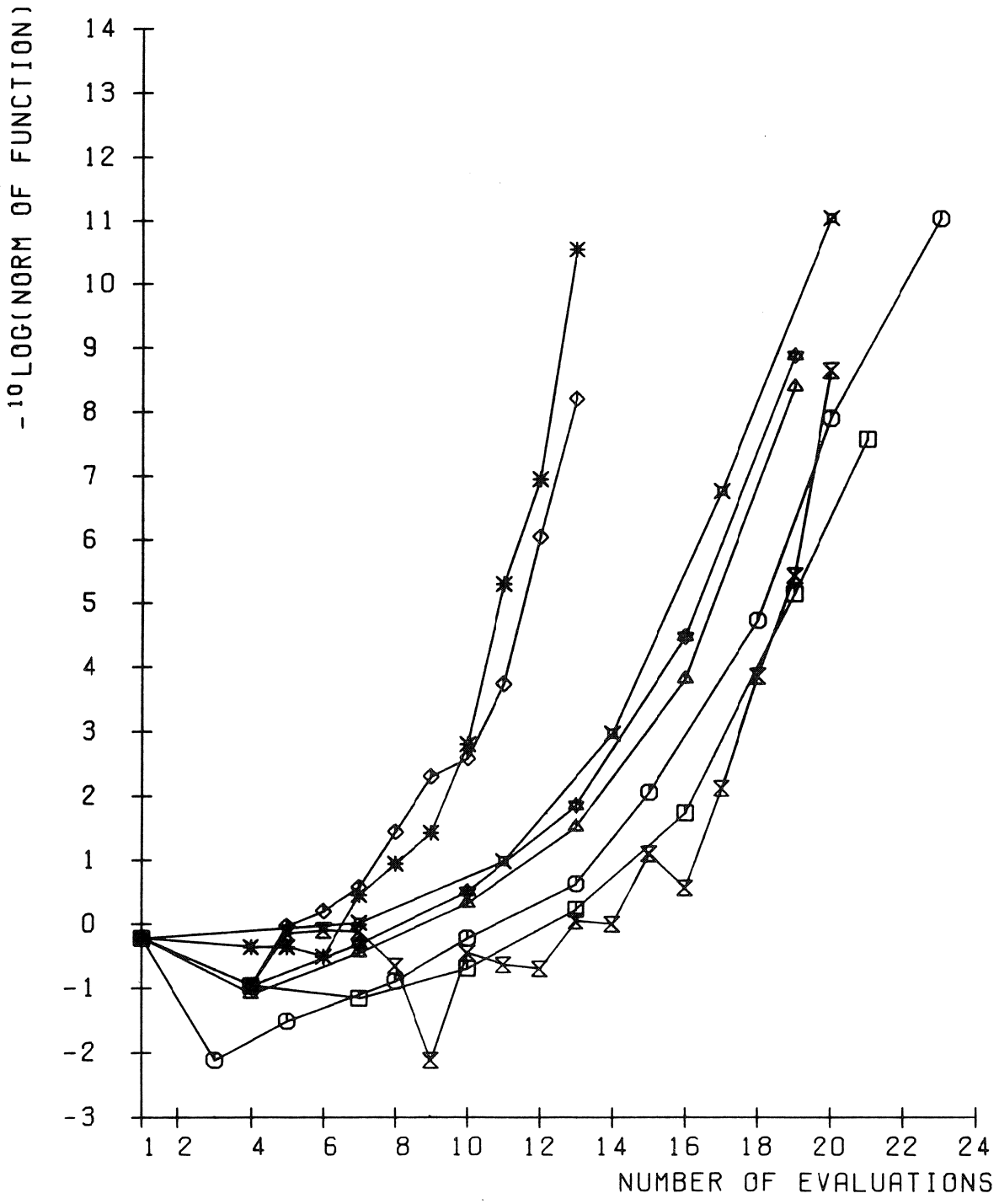
DIAGRAM 7.3



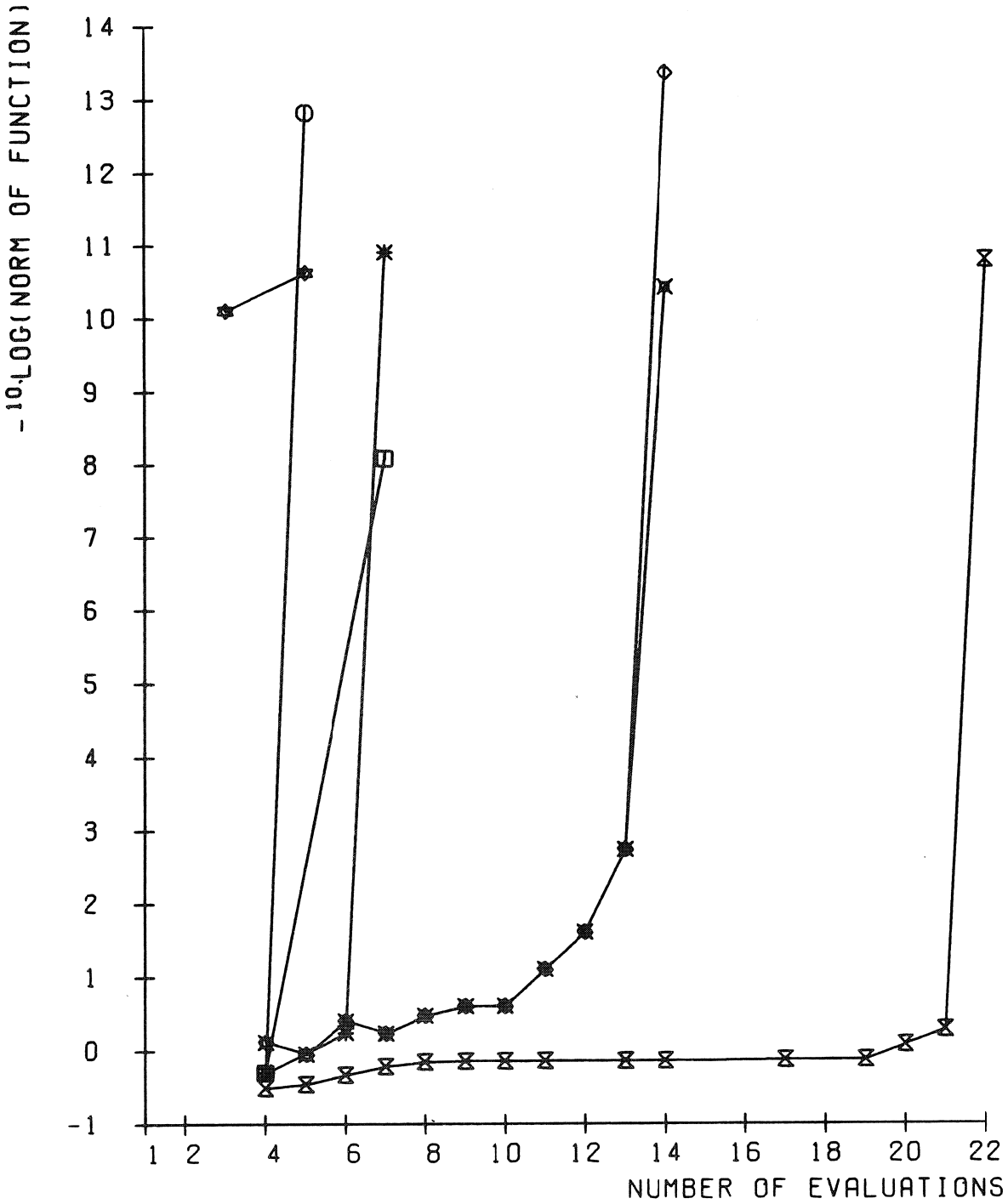
FUNCTION 4 , ORDER 2 , CASE 1

PROGRAMS IN ALGOL 60

DIAGRAM 7.4



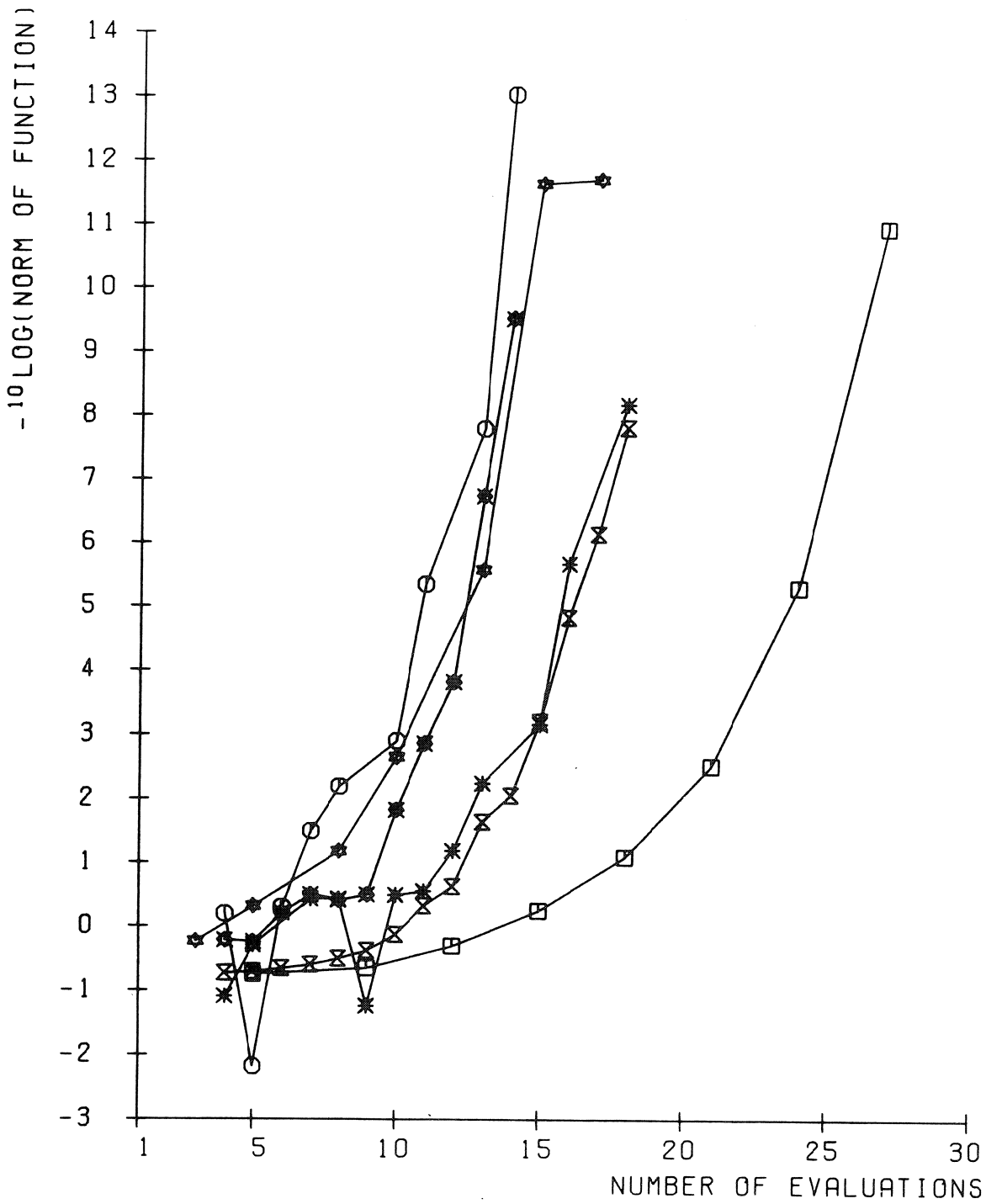
FUNCTION 6 , ORDER 2 , CASE 1



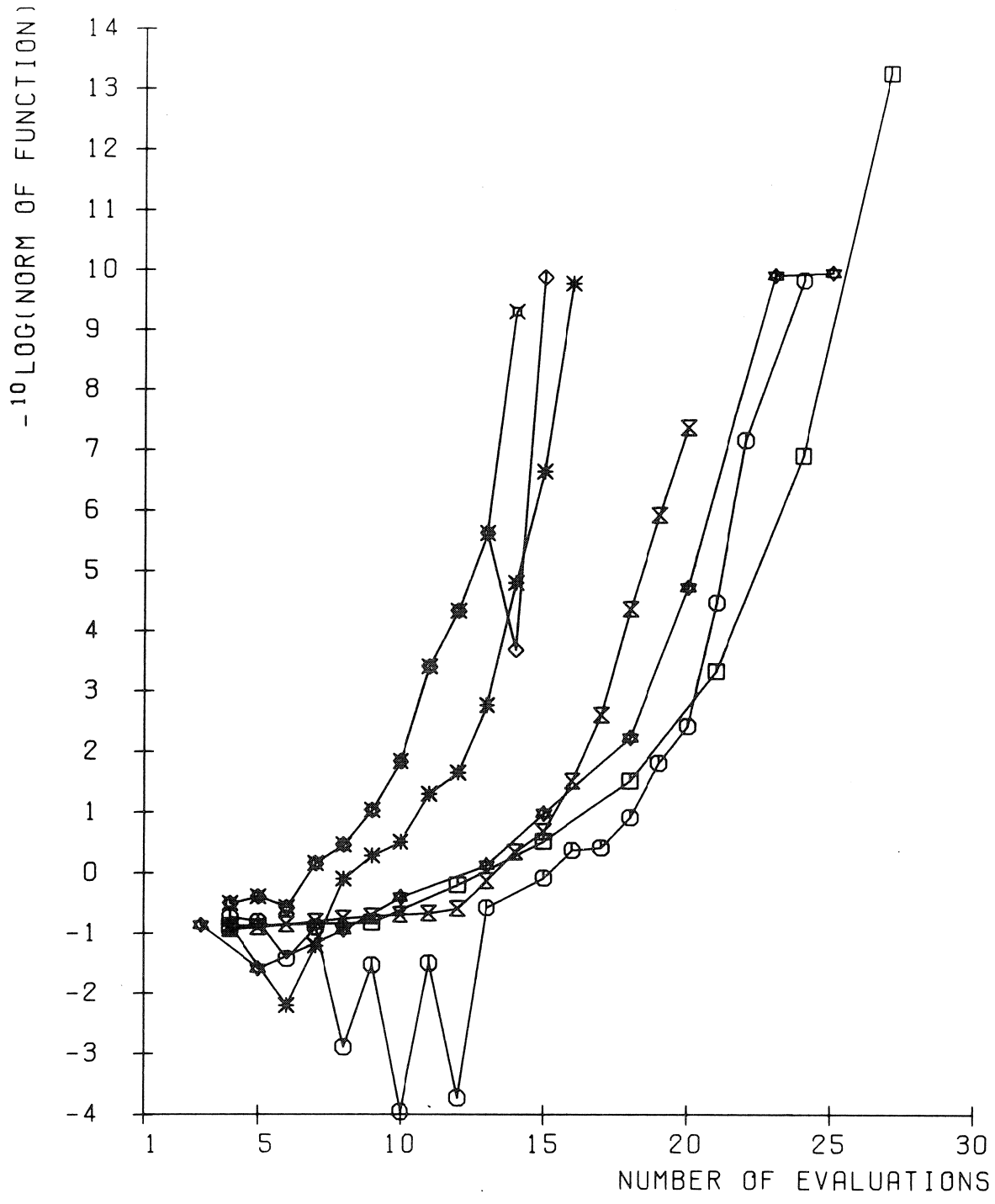
FUNCTION 8 , ORDER 2 , CASE 0

PROGRAMS IN ALGOL 60

DIAGRAM 7.6



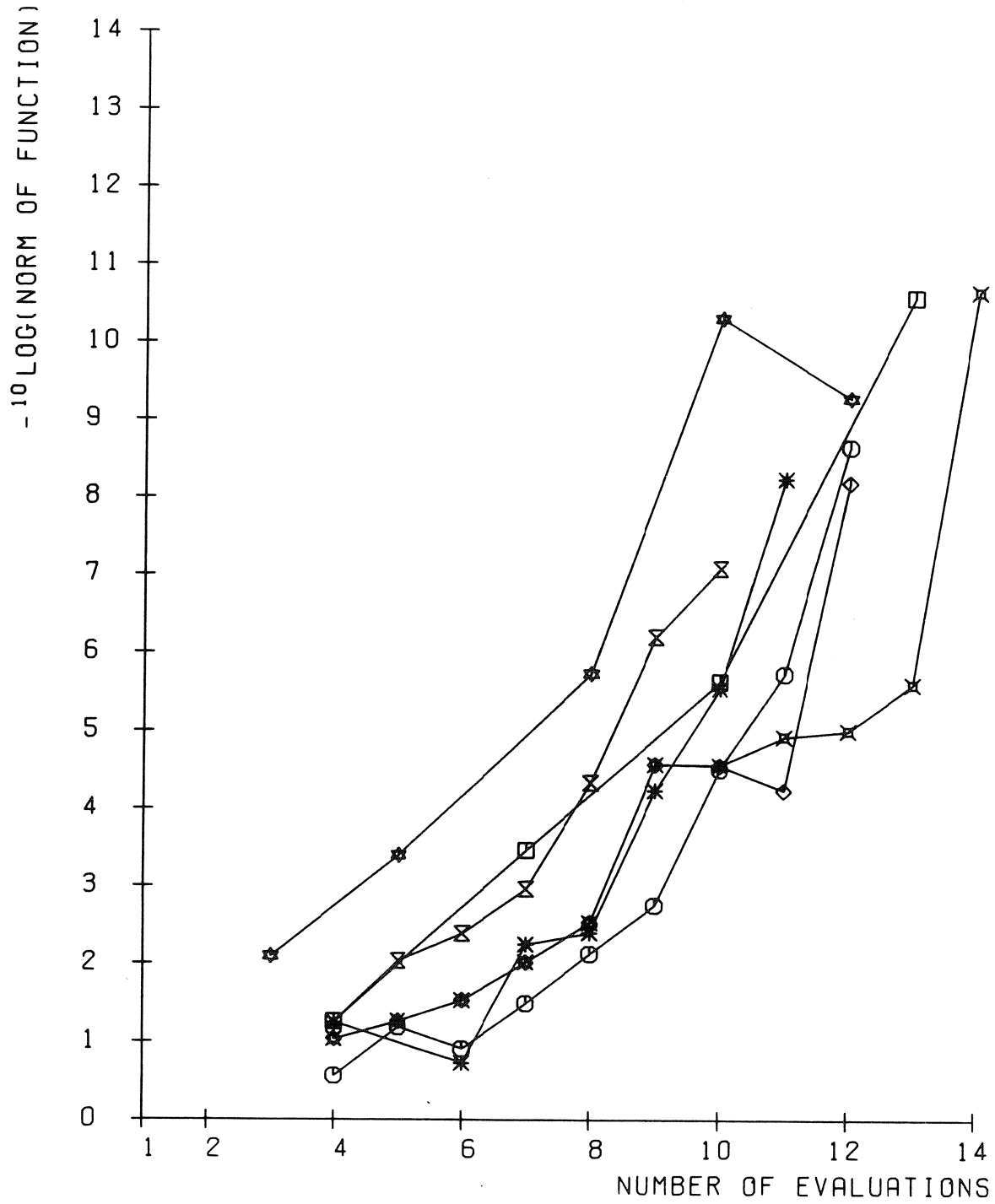
FUNCTION 2 , ORDER 2 , CASE 0



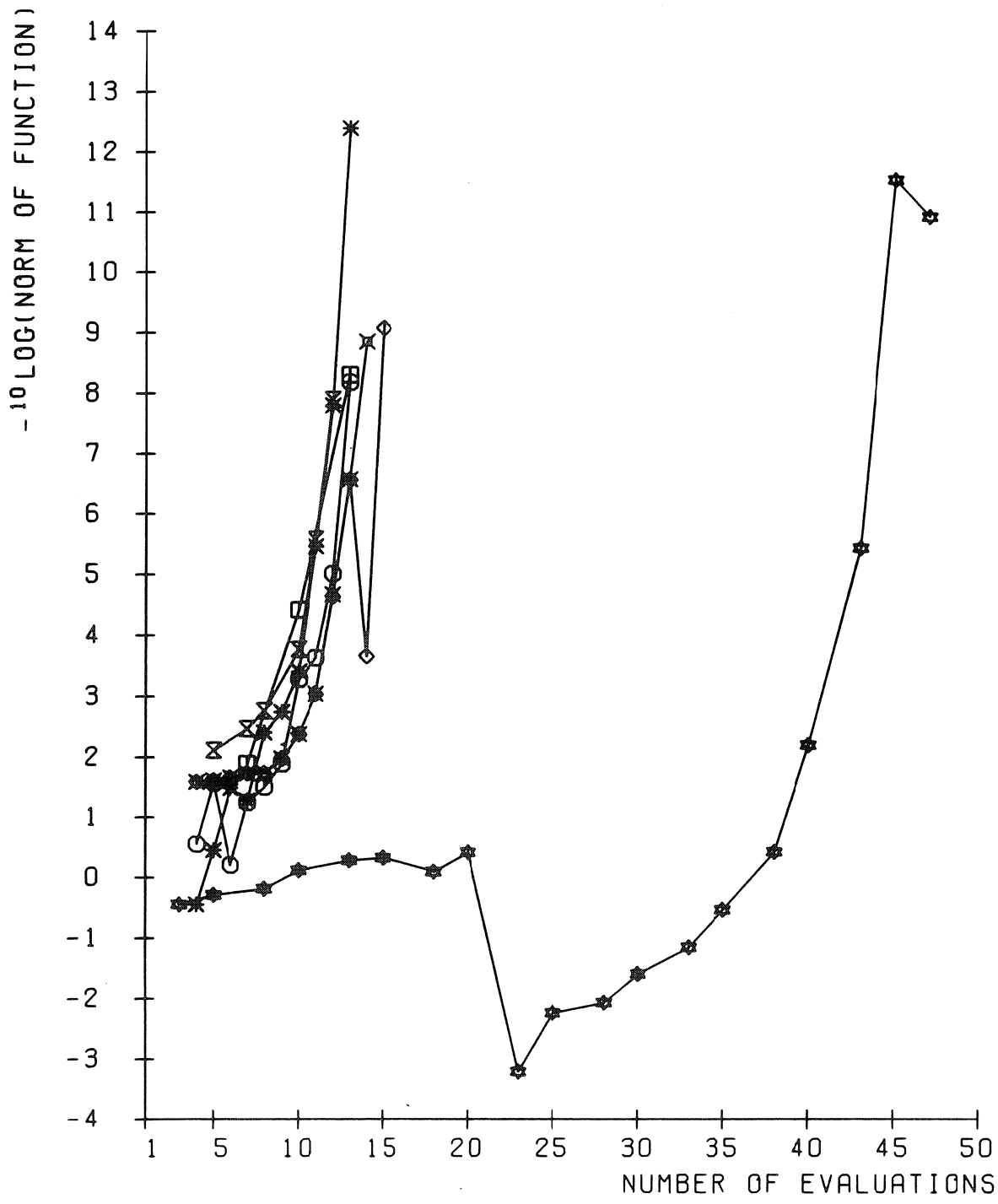
FUNCTION 2 , ORDER 2 , CASE 2

PROGRAMS IN FORTRAN

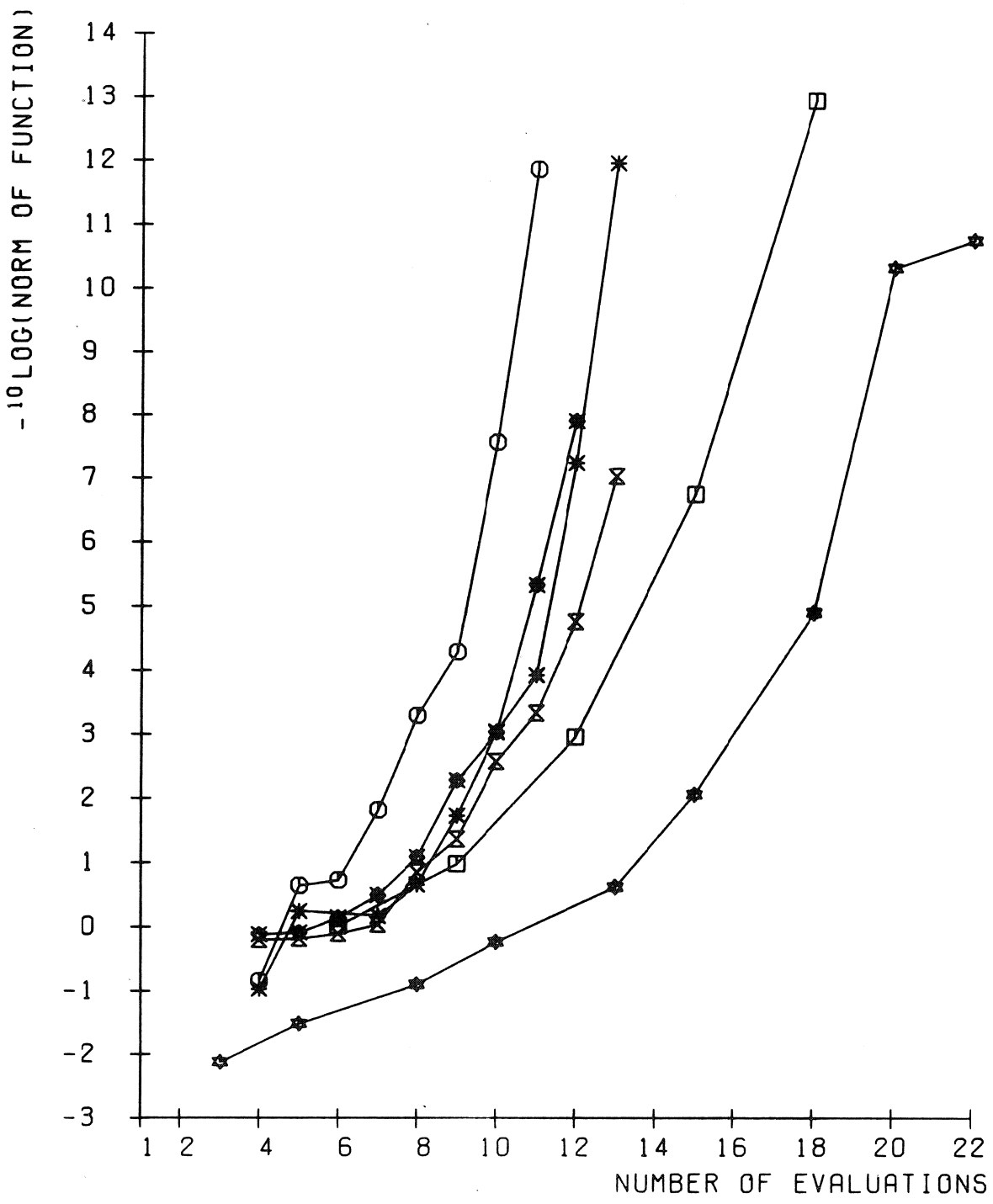
DIAGRAM 7.8



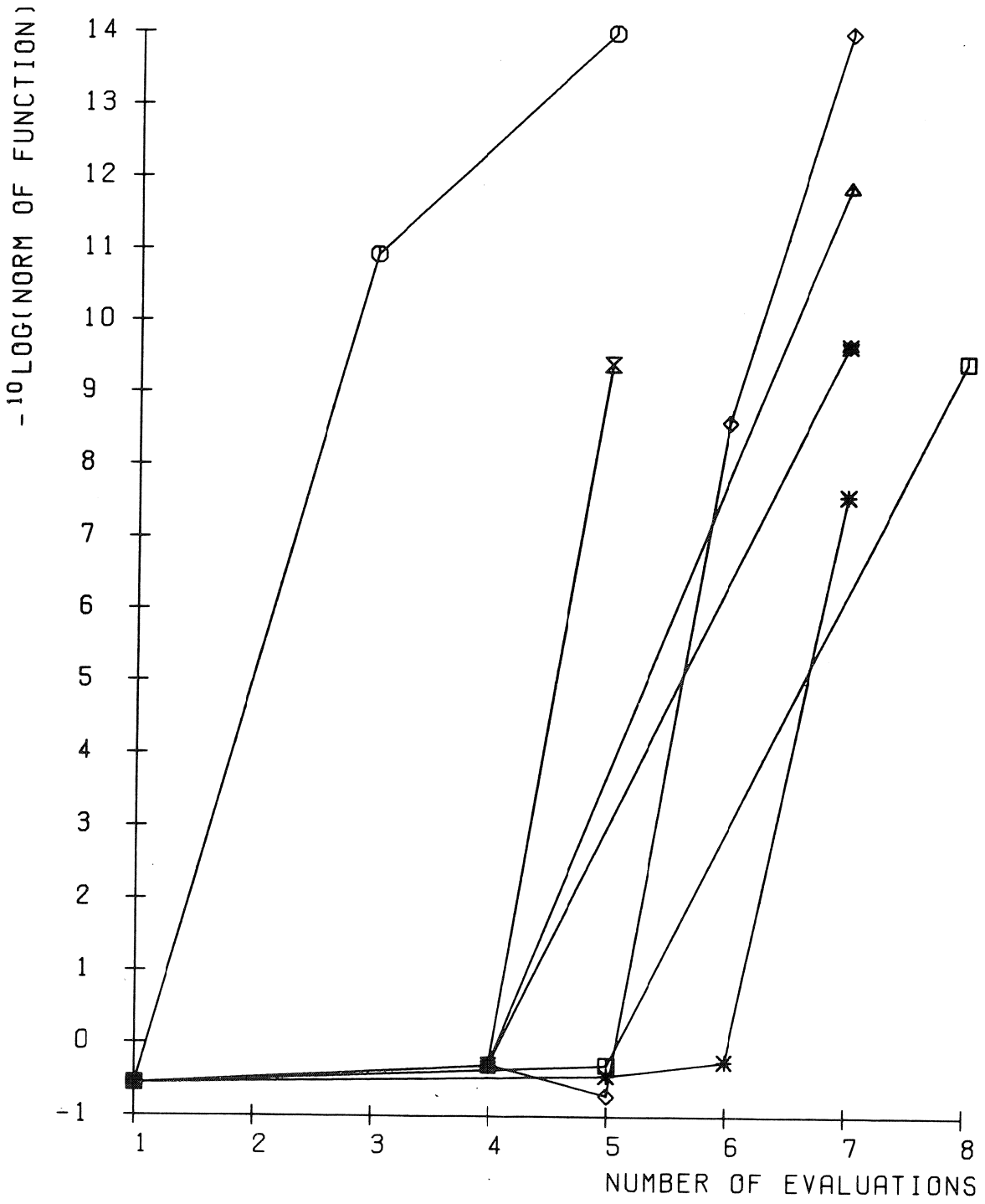
FUNCTION 4 , ORDER 2 , CASE 0



FUNCTION 4 , ORDER 2 , CASE 1



FUNCTION 6 , ORDER 2 , CASE 1



FUNCTION 8 , ORDER 2 , CASE 0

Hence, as long as the precision is not too high, relative to the round-off error in the function and/or its Jacobian, the efficiency of the program is not influenced by the precision required.

7.4.2. Special properties and features

The use of most programs is about the same. The user should provide the function and sometimes its Jacobian matrix, the precision required and sometimes some controlling parameters. For most programs the function has to be programmed such that for a given argument vector the whole function vector is calculated. However, the programs I and O require the programming of the function such that only one component of the function vector is calculated for a given argument vector. This may have severe consequences for the efficiency of programs I and O when the evaluation of one component is almost as expensive as evaluation of the whole function vector.
An advantage of the programs I and O, which is induced by the underlying algorithm, is that solution of problems for which part of the function components are linear can be done relatively very efficient if the function components are ordered in the right way. We will illustrate this feature by the following example.
If we reorder the function components in problem (1,n,0) (see section 5.1.1) such that the first (n-1) are linear and the last one is non-linear, thus:

$$F_i(x) = -(n+1) + x_i + \sum_{j=1}^{n} x_j \ , \ i = 1,\ldots,n-1,$$
$$F_n(x) = -1 + \prod_{j=1}^{n} x_j ,$$

then, this problem is solved by the program I and O and solution is remarkably efficient. The difference between solving problem (1,n,0) and its reordered analogue with the programs I and Oa is illustrated by table 7.19.

TABLE 7.19

Influence of reordering function components such that linear ones come first for the programs I and 0 and problem (1,n,0)

order	program I				program 0a			
	normal		reordered		normal		reordered	
	n_s	n_F	n_s	n_F	n_s	n_F	n_s	n_F
2	1	2.5	4	10	1	2.5	4	10
3	6	18	5	15	6	18	5	15
5	17	68	5	20	17	68	6	24
10	76	T	6	39	50	T	7	46
15	83	T	7	63	50	T	7	63
25	1	D	7	98	1	D	7	98

The reason for this behaviour is that in the reordered case the linear components are treated first so that a much better approximation to the solution is used by the time that the nonlinear component is approximated. All other programs use a method of vector-wise approximation so that reordering does not influence the behaviour of the program.

However, program K has a mechanism for treating linear components apart. The user may define the problem function as an underdetermined linear system together with an underdetermined system of nonlinear equations. For program K the number of nonlinear equations has to be at least two. The results of using this mechanism for problem (1,n,0) are given in table 7.20.

TABLE 7.20

Influence of the use of the feature
for linear components in the programs
Ka and Kb for problem (1,n,0)

order	program Ka				program Kb			
	normal		with feature		normal		with feature	
	n_S	n_F	n_S	n_F	n_S	n_F	n_S	n_F
3	12	23	5	10	10	20	5	11
5	10	23	8	16	7	22	5	12
10	16	49	6	13	5	26	5	12
15	16	60	8	16	6	38	5	12
25	1	D	7	14	1	D	5	12

Clearly, the reliability of the programs I, K and O is influenced by the use of these features. When we assume that the user takes full advantage of these features, then we should do the same if we compute the reliability of these programs and we should replace the values for the programs I, Ka, Kb and Oa in table 7.17 by those given in table 7.21. We see from this table that program Oa is fully reliable if we give only one significant digit for these values. In fact, it failed only once by solving 40 difficultly solvable problems. As is seen from table 7.16 it failed to solve problem (12,4,0). This problem has a singular Jacobian matrix (rank 2) at the solution.

TABLE 7.21

Reliability for some programs if special features are used.

R	$Z(R,T_s^d,\Psi_s^d)$	$Z(R,T_\ell^d,\Psi_\ell^d)$	$Z(R,T^d,\Psi^d)$
I	0.9	0.5	0.8
Ka	0.9	0.7	0.8
Kb	0.9	1	0.9
Oa	1	1	1

The last remark about the behaviour of the programs which is induced by the experimental results is concerning the failure detection of the programs. Most of the programs do only generate an error exit if the matrix of the linear system appears to be (numerically) singular. Sometimes, step size control (program B and D) or a resetting mechanism (program N) gives a possibility to detect divergence or convergence to a stationary point which is no solution, so that an error exit can be generated. However, we see from the tables 7.15 and 7.16 that rather often the program has to be terminated by the user by choosing some upper bound for the number of function evaluations. Sometimes, this facility is built in in the program, so that the best results obtained so far are given as output, however, in other cases the user himself should build in a jump out of the program by programming the function in such a way, which is very undesirable.
In either case, the user has to choose some upper bound on the number of function evaluations or iterations without having any reasonable idea about it, since this depends heavily on the method used and the problem to be solved.

We feel that good failure exits are essential for a good program, however, we do not judge the given programs on this criterion in this report.

8. CONCLUSIONS

8.1. *General remarks*

As we mentioned before, the method of choosing a program for solving a system of nonlinear equations will usually be a method of trial and error as long as we do not know whether the problem is easily solvable. However, with the results given in section 7 we feel that we can give the user reliable information about what program he should try first and if it fails what will be the best to try subsequently and so on. We distinguish between the six classes of problems Ψ_{s1}, Ψ_{s2}, $\Psi_{\ell 1}$, $\Psi_{\ell 2}$, $\Psi_{\ell 3}$ and $\Psi_{\ell 4}$ defined in section 5.1 and we assume that the user is not able to determine whether his problem is easily solvable or not. Our method of choosing is as follows:

-first of all drop all programs that are not reasonable (see section 7.1); these are the programs B,D,Ga,Gb,Jb,Ka,L and Ob;
-then the most efficient program in ALGOL 60 and FORTRAN is chosen; if two programs are equally efficient, then the most reliable is chosen; (we use $Z(R,\bar{T}^d,\Psi^d)$ from tables 7.18 and 7.21);
-as the next choice we take the next efficient program whose reliability is higher than the reliability of the program that is chosen first (for both ALGOL 60 and FORTRAN)
-we repeat this process until we do not have any choice any more;
-we will not use program C if program A can be used more efficiently and vice versa, since these programs are the same, except for the use of an analytical or approximated Jacobian matrix.

Hence, we obtain for a certain class of problems a sequence of programs in ALGOL 60 and in FORTRAN. So, all the user has to do is to determine in which class his problem should be placed, to read the conclusions given about this class and to try and solve his problem with the programs in the order given. Only if he is not interested in efficiency he should choose the last program of the sequence of programs in the language he uses, since this is the most reliable one.
The conclusions are based on the assumption that the user makes use of the features of some programs mentioned in section 7.4.2 if one or more of the function components are linear.

Furthermore, it is obvious that we assume that the programs are used in the form as described in section 4.
To simplify our conclusions we do not distinguish between programs Na and Nb. There is always a slight preference for program Nb.
For convenience we say that *a cheap Jacobian is available* if the user can supply analytical derivatives of the function and the evaluation is about as expensive as one evaluation of the function or cheaper. Furthermore, we formalize the notation of the sequences of programs as follows: a semicolon between two programs means that one should try first the program mentioned before the semicolon and if it failed then one should try the program behind the semicolon; an or-symbol (v) between two programs means that there is no preference between these two programs,

the user should try one of them. So we end up with the following conclusions for the various classes of problems.

8.2. *Solving small cheap problems* (Ψ_{s1} in section 5.1)

Programs in ALGOL 60 : A ∨ C ∨ I ,
where A can only be used if analytical derivatives are available.

Programs in FORTRAN : Oa.

8.3. *Solving small expensive problems* (Ψ_{s2} in section 5.1)

Programs in ALGOL 60 :
if a cheap Jacobian is available then: A, otherwise: H; I.

Programs in FORTRAN : Oa.

8.4. *Solving large very cheap problems* ($\Psi_{\ell 1}$ in section 5.1)

Programs in ALGOL 60 : A ∨ C ; I ,
where A can only be used if analytical derivatives are available.

Programs in FORTRAN : Ja ; Oa.

8.5. *Solving large cheap problems* ($\Psi_{\ell 2}$ in section 5.1)

Programs in ALGOL 60 :
if a cheap Jacobian is available then: A,
otherwise:
if α (defined by (7.1.1.7)) is about 1 or less then: F ; C ,
otherwise: H ; F ; I .

Programs in FORTRAN : N ; Kb ; Oa.

8.6. *Solving large expensive problems* ($\Psi_{\ell 3}$ in section 5.1)

Programs in ALGOL 60 :
if a cheap Jacobian is available then: A,
otherwise, if α is about 1 or less then: H ; F ; I ,
otherwise H ; I.

Programs in FORTRAN :
if α is about 1 or less then: N ; Kb ; Oa ,
otherwise: N ; Oa.

8.7. *Solving large very expensive problems* ($\Psi_{\ell 4}$ in section 5.1)

Programs in ALGOL 60 :
if a cheap Jacobian is available then: A,
otherwise: H ; I .

Programs in FORTRAN : N ; Oa.

8.8. *General conclusions*

We see that the following programs in ALGOL 60 are useful for having available (for instance in a software library):

A, C, F, H, I.

Except for A and C (both Newton's method) they are based on different types of algorithms. Program F is based on the secant algorithm, program H on the quasi-Newton algorithm and program I on a method of component-wise approximation. As far as programs in FORTRAN are concerned it is sufficient to have available:

Ja, Kb, N, Oa.

Here again we have Newton's method (Ja), a secant method (Kb), a quasi-Newton method (N) and a method of component-wise approximation (Oa).
The comparison of the programs in ALGOL 60 indicates that it might be useful to have a FORTRAN-version of program A.

Furthermore it should be noted that translation of the programs K and N in ALGOL 60 may change the picture and the modifications in Oa relative to its analogue in ALGOL 60, program I, seems to be worth while.

ACKNOWLEDGEMENTS,

The author is very grateful to mrs. M. Werkhoven and A.C. IJsselstein, who helped him with a lot of programming and plotting and to J. Kok for providing a version of Newton's method which was adapted to our software library.
He also likes to thank prof. P.J. van der Houwen, P.A. Beentjes and J. Kok for their contributions in the discussions about the framework of this report and for their careful reading of the manuscript.
Finally he likes to thank ms. I. Cannegieter and D. Zwarst for the work that had to be done to get this report typed and printed.

REFERENCES

[1] BOGGS, P.T. & J.E. DENNIS, *A Continuous analogue analysis of nonlinear iterative methods*, Cornell Univ., TR 200, Ithaca (1974).

[2] BROWN, K.M., *Algorithm 316; solution of simultaneous non-linear equations*, Comm. ACM, 10 (1967), 728-729.

[3] BROWN, K.M., *A quadratically convergent Newton-like method based upon Gaussian elimination*, SIAM J. Num. An., 6 (1969) 560-569.

[4] BROWN, K.M., *Computer oriented algorithms for solving systems of simultaneous nonlinear algebraic equations*,
In: G.D. Byrne & C.A. Hall (eds.), *Numerical solution of systems of nonlinear algebraic equations*, Academic Press (1973).

[5] BROWN, K.M. & S.D. CONTE, *The solution of simultaneous nonlinear equations*, Proc. ACM 22nd Nat. Conf., (1967) 111-114.

[6] BROWN, K.M. & M. FRISCH, *Private communication.*

[7] BROYDEN, C.G., *A class of methods for solving nonlinear simultaneous equations*, Math. Comp., 19 (1965) 577-593.

[8] BROYDEN, C.G., *A new method of solving nonlinear simultaneous equations; algorithm 44*, Comp. J., 12 (1969) 94-99, 406-408.

[9] BROYDEN, C.G., *The convergence of single-rank quasi-Newton methods*, Math. Comp., 24 (1970) 365-382.

[10] BROYDEN, C.G., *Recent developments in solving nonlinear algebraic systems*,
In: P. Rabinowitz (ed.), *Numerical methods for nonlinear algebraic equations*, Gordon & Breach, (1970).

[11] BROYDEN, C.G., *The convergence of an algorithm for solving sparse nonlinear systems*, Math. Comp., 25 (1971) 285-294.

[12] BROYDEN, C.G., *Quasi-Newton, or modification methods.*
In: G.D. Byrne & C.A. Hall (eds.), *Numerical solution of nonlinear algebraic equations*, Academic Press (1973).

[13] BROYDEN, C.G., J.E. DENNIS & J.J. MORÉ, *On the local and superlinear convergence of quasi-Newton methods*, J.I.M.A., 12 (1973) 223-245.

[14] BUS, J.C.P., *An analysis of the convergence of Newton-like methods for solving systems of nonlinear equations*, Mathematisch Centrum, report NW 20/75, Amsterdam (1975).

[15] CARNAHAN, B., H.A. LUTHER & J.O. WILKES, *Applied numerical methods*, Wiley (1964).

[16] COLLATZ, L., *Funktional Analysis und numerische Mathematik*, Springer (1964), English ed., Academic Press (1966).

[17] DAVIDENKO, D.F., *On a new method of numerical solution of systems of nonlinear equations (russian)*, Dokl. Akad. Nauk. SSSR. 88 (1953) 601-602.

[18] DAVIDON, W.C., *Variable metric methods for minimization*, Argonne Nat. Lab. report 5990 (1959).

[19] DEIST, F.H. & L. SEFOR, *Solution of systems of nonlinear equations by parameter variation*, Comp. J., 10 (1967) 78-82.

[20] DENNIS, J.E. & J.J. MORÉ, *A characterization of superlinear convergence and its application to quasi-Newton methods*, Math. Comp. 28 (1974) 549-560.

[21] DENNIS, J.E. & J.J. MORÉ, *Quasi-Newton methods, motivation and theory*, Cornell Univ. report TR 217 (1974).

[22] DULLEY, D.B. & M.L.V. PITTEWAY, *Algorithm 314, Finding a solution of n functional equations in n unknowns*, Comm. ACM 10 (1967) 726.

[23] EINARSSON, B., *Testing and evaluating of some subroutines for numerical quadrature*,
In: D.J. Evans (ed.), *Software for numerical mathematics*, Academic Press (1974).

[24] FLETCHER, R., *Function minimization without evaluating derivatives - a review*, Comp. J. 8 (1965) 33-41.

[25] FLETCHER, R. & M.J.D. POWELL, *A rapidly convergent descent method for minimization*, Comp. J. 6 (1963) 163-168.

[26] FREUDENSTEIN, F. & B. ROTH, *Numerical solutions of systems of nonlinear equations*, J. ACM 10 (1963) 550-556.

[27] GHERI, G. & O.G. MANCINO, *A significant example to test methods for solving systems of nonlinear equations*, Calcolo 8 (1971) 107-113.

[28] GRAGG, W.B. & G.W. STEWART, *A stable variant of the secant method for solving nonlinear equations*, Carnegie Mellon Univ. rep. (1974).

[29] HAGUE, S.J. et al., *NAG Project note, number 5*, (1974).

[30] HILLSTROM, K.E., MINPACK I, *A study in the modularization of a package of computer algorithms for the unconstrained nonlinear optimization problem*, Argonne Nat. Lab. rep. TM-252 (1974).

[31] HULL, T.E. et al., *Comparing numerical methods for ordinary differential equations*, SIAM J. Num. An. 9 (1974) 603-637.

[32] IMSL, *International Mathematical and Statistical libraries*, Reference manual.

[33] LEVENBERG, K., *A method for the solution of certain nonlinear problems in least squares*. Quart. Appl. Math. 2 (1944) 164-168.

[34] LOOTSMA, F.J., *Non-linear optimization in industry and the development of optimization programmes*, Paper to be presented at the Conf. on Optimization in Action, Bristol (1975).

[35] MARQUARDT, D.W., *An algorithm for least-squares estimation of nonlinear parameters*, SIAM J. 11 (1963) 431-441.

[36] MEYER, G.H., *On solving nonlinear equations with a one-parameter operator imbedding*, SIAM J. Num. An. 5 (1967) 739-752.

[37] MSL, *Math. Science Library*, Reference manual.

[38] NAG, *Numerical Algorithms Group*, Library manual.

[39] NUMAL, *A library of numerical procedures in* ALGOL 60, Reference manual, Mathematisch Centrum, Amsterdam (1974).

[40] ORTEGA, J.M. & W.C. RHEINBOLDT, *Iterative solution of nonlinear equations in several variables*, Academic Press (1970).

[41] PANKIEWICZ, W., *Algorithm 378, discretized Newton-like method for solving a system of simultaneous nonlinear equations*, Comm. ACM 13 (1970) 259-260.

[42] PARLETT, B.N. & WANG, Y., *The influence of the compiler on the cost of mathematical software; in particular on the cost of triangular factorization*, TOMS 1 (1975) 35-46.

[43] POWELL, M.J.D., *An iterative method for finding stationary values of a function of several variables*, Comp. J. 5 (1962) 147-151.

[44] POWELL, M.J.D., *A hybrid method for nonlinear equations; A* FORTRAN *subroutine for solving systems of nonlinear algebraic equations*,
In: P. Rabinowitz (ed.), *Numerical methods for nonlinear algebraic equations*, Gordon & Breach (1970).

[45] RALL, L.B., *Computational solution of nonlinear operator equations*, Wiley (1969).

[46] ROBINSON, S.M., *Interpolative solution of systems of nonlinear equations*, SIAM J. Num. An. 3 (1966) 650-658.

[47] SCHWETLICK, H., *Algorithm 12; a discrete method for the solution of finite-dimensional systems of nonlinear equations*, Comp. 5 (1970) 82-88.

[48] VANDERGRAFT, J. & C. MESZTENYI, *Remark on algorithm 314*, Comm. ACM 12 (1969), 38-39.

[49] WILKINSON, J.H., *Rounding errors in algebraic processes*, Her Majesty's Stationary Office (1963)

[50] WILKINSON, J.H., *The algebraic eigenvalue problem*, Clarendon Press (1965).

APPENDIX

In this appendix we give source texts of some programs which have been changed with respect to the text given in the references. Some of them are already adapted to the software library NUMAL [39]. Some other programs are changed. We give these texts, mainly to show what source texts are tested. Therefore, source texts of programs which are not changed by us, can be found in literature and we did not list those here. Furthermore the source texts of the subroutines NEWT (program J), NONLIQ (program L) and QNWT (program M) from the MSL software library are not listed since they are not available.

Since some of the programs in ALGOL 60 make use of procedures declared by code numbers, we will give a short explanation of their performance. Detailed descriptions and source texts are given in NUMAL [39].

real procedure vecvec(1,u,shift,a,b) ;
vecvec delivers the inner product of the vectors given in a[1:u] and b[1+shift : u+shift].

real procedure matvec(1,u,i,a,b) ;
matvec delivers the inner product of the vector given in b[1:u] and the row-vector given in a[i:i,1:n].

real procedure tamvec(1,u,j,a,b) ;
tamvec delivers the inner product of the vector given in b[1:u] and the column-vector given in a[1:u,j:j].

procedure dupvec(1,u,s,a,b) ;
dupvec duplicates the vector given in b[1+s : u+s] to a[1:u].

procedure elmvec(1,u,s,a,b,x) ;
elmvec adds x times the vector given in b[1+s : u+s] to the vecor given in a[1:u].

procedure elmcolvec(1,u,j,a,b,x) ;
elmcolvec adds x times the vector given in b[1:u] to the column-vector given in a[1:u,j:j].

procedure gsssol(a,n,aux,b) ;

gsssol solves the linear system of order n, whose matrix is given in a[1:n,1:n] and whose right-hand side is given in b[1:n]. The solution is overwritten on b[1:n]. The matrix elements are overwritten. In the auxiliary array aux one should give in aux [2] the precision of arithmetic and in aux [4] some controlling parameter (advised value 8). The rank of the matrix is delivered in aux [3].

The parameter lists of the tested procedures in ALGOL 60 are made as uniformly as possible. The parameters have the following meaning:

n : order of system ;

x : the initial guess as input and the solution as output ;

f : the functionvector; on exit the functionvector at the calculated solution ;

funct : a boolean type procedure;
boolean procedure funct(n,x,f) ; the parameters have the same meaning as above and the program is terminated if the procedure delivers *false* for some argument vector ;

jacobian : a procedure for calculating the Jacobian matrix ;
procedure jacobian(n,x,f,jac,funct) ;
the Jacobian matrix is delivered in jac[1:n,1:n] ;
the other parameters have the same meaning as above ;

in : some auxiliary array to provide tolerance and values for control parameters (input) ;

out : some auxiliary array in which some by-products are delivered;
For the programs E up to I out [5] $\neq$ 0 means that no solution is found; for the programs A up to D out [6] $\neq$ 6,4 means that no solution is found.

Since the given texts are not really intended for use, but only to validate our conclusions and to show what changes are made to the original source texts, we assume that the short description above will be sufficient.

```
"COMMENT" NEWTONS METHOD, PROGRAM A OR C;
"PROCEDURE" PROGRAM A(N, X, F, FUNCT, JACOBIAN, IN, OUT);
"VALUE" N; "INTEGER" N;
"ARRAY" X, F, IN, OUT;
"BOOLEAN" "PROCEDURE" FUNCT;
"PROCEDURE" JACOBIAN;

"BEGIN" "INTEGER" TEXT, IT, ITMAX, FEVAL, FEVALMAX;
    "REAL" RN, RELTOLPAR, ABSTOLPAR, ABSTOLRES, STAP, NORMX;
    "BOOLEAN" TESTTHF;
    "ARRAY" JAC[1:N + 1,1:N], SOL[1 : N], AUX[1 : 7];

    "REAL""PROCEDURE" VECVEC(L, U, SHIFT, A, B); "CODE" 34010;
    "PROCEDURE" DUPVEC(L, U, S, A, B); "CODE" 31030;
    "PROCEDURE" ELMVEC(L, U, S, A, B, X); "CODE" 34020;
    "PROCEDURE" GSSSOL(A, N, AUX, B); "CODE" 34232;

    "BOOLEAN""PROCEDURE" LOCFUNCT(N, X, F);
    "VALUE" N; "INTEGER" N; "ARRAY" X, F;
    "BEGIN" LOCFUNCT:= TESTTHF:= FUNCT(N, X, F)
        "AND" TESTTHF; FEVAL:= FEVAL + 1
    "END" LOCFUNCT;

    ITMAX:= FEVALMAX:= IN[4]; AUX[2]:= N * IN[0];
    AUX[4]:= 0; RELTOLPAR:= IN[1] ** 2; ABSTOLPAR:= IN[2] ** 2;
    ABSTOLRES:= IN[3] ** 2; TEXT:= 0; TESTTHF:= "TRUE";
    STAP:= OUT[1]:= OUT[5]:= OUT[7]:= 0;
    FUNCT(N, X, SOL); RN:= VECVEC(1, N, 0, SOL, SOL);
    OUT[3]:= SQRT(RN); FEVAL:= 1;
    "FOR" IT:= 1, IT + 1 "WHILE" IT <= ITMAX "AND"
          FEVAL < FEVALMAX "DO"
    "BEGIN" OUT[5]:= IT; JACOBIAN(N, X, SOL, JAC, LOCFUNCT);
       "IF" "NOT" TESTTHF "THEN"
       "BEGIN" TEXT:= 3; "GO TO" FAIL "END";
       GSSSOL(JAC, N, AUX, SOL);
       "IF" AUX[3] <= N "THEN""BEGIN" TEXT:= 1; "GO TO" FAIL "END";
       STAP:= VECVEC(1, N, 0, SOL, SOL);
       NORMX:= VECVEC(1, N, 0, X, X);
       "IF" STAP > RELTOLPAR * NORMX + ABSTOLPAR
          "OR" IT = 1 "AND" STAP > 0 "THEN"
       "BEGIN" ELMVEC(1, N, 0, X, SOL, -1); FEVAL:= FEVAL + 1;
          "IF" "NOT" FUNCT(N, X, F) "THEN"
          "BEGIN" TEXT:= 2; "GO TO" FAIL "END";
          RN:= VECVEC(1, N, 0, F, F);
          "IF" RN <= ABSTOLRES "THEN"
          "BEGIN" TEXT:= 4; ITMAX:= IT "END"
          "ELSE" DUPVEC(1, N, 0, SOL, F)
       "END" ITERATION AND TESTS "ELSE"
       "BEGIN" TEXT:= 6; ITMAX:= IT "END"
    "END" OF ITERATIONS;

FAIL :
    OUT[1]:= SQRT(STAP); OUT[2]:= SQRT(RN); OUT[4]:= FEVAL;
    OUT[6]:= TEXT; OUT[8]:= AUX[3]; OUT[9]:= AUX[5]
"END" PROGRAM A;
```

```
"COMMENT" NEWTONS METHOD WITH STEP SIZE CONTROL, PROGRAMS B AND D;
"PROCEDURE" PROGRAM B(N, X, F, FUNCT, JACOBIAN, IN, OUT);
"VALUE" N; "INTEGER" N;
"ARRAY" X, F, IN, OUT;
"BOOLEAN" "PROCEDURE" FUNCT;
"PROCEDURE" JACOBIAN;

"BEGIN" "INTEGER" I, J, INR, MIT, TEXT,
    IT, ITMAX, INRMAX, TIM, FEVAL, FEVALMAX;
    "REAL" RHO, RES1, RES2, RN, RELTOLPAR, ABSTOLPAR, ABSTOLRES,
       STAP, NORMX;
    "BOOLEAN" CONV, TESTTHF, DAMPING ON;
    "ARRAY" JAC[1:N + 1,1:N], PR, FU2, SOL[1 : N], AUX[1 : 7];

    "REAL""PROCEDURE" VECVEC(L, U, SHIFT, A, B); "CODE" 34010;
    "PROCEDURE" DUPVEC(L, U, S, A, B); "CODE" 31030;
    "PROCEDURE" ELMVEC(L, U, S, A, B, X); "CODE" 34020;
    "PROCEDURE" GSSSOL(A, N, AUX, B); "CODE" 34232;

    "BOOLEAN""PROCEDURE" LOC FUNCT(N, X, F);
    "VALUE" N; "INTEGER" N; "ARRAY" X, F;
    "BEGIN" LOC FUNCT:= TEST THF:= FUNCT(N, X, F)
        "AND" TEST THF; FEVAL:= FEVAL + 1
    "END" LOC FUNCT;

    ITMAX:= FEVALMAX:= IN[4]; AUX[2]:= N * IN[0]; TIM:= IN[7];
    AUX[4]:= 8; RELTOLPAR:= IN[1] ** 2; ABSTOLPAR:= IN[2] ** 2;
    ABSTOLRES:= IN[3] ** 2; INRMAX:= IN[6];
    DUPVEC(1, N, 0, PR, X);
    TEXT:= 8; MIT:= 0; TEST THF:= "TRUE";
    RES2:= STAP:= OUT[1]:= OUT[5]:= OUT[7]:= 0;
    FUNCT(N, X, SOL); RN:= VECVEC(1, N, 0, SOL, SOL);
    OUT[3]:= SQRT(RN); FEVAL:= 1; DAMPING ON:= "FALSE";
    "FOR" IT:= 1, IT + 1 "WHILE" IT <= ITMAX "AND"
          FEVAL < FEVALMAX "DO"
    "BEGIN" OUT[5]:= IT; JACOBIAN(N, X, SOL, JAC, LOCFUNCT);
       "IF" ~ TEST THF "THEN"
       "BEGIN" TEXT:= 3; "GO TO" FAIL "END";
       GSSSOL(JAC, N, AUX, SOL);
       "IF" AUX[3] ~= N "THEN"
       "BEGIN" TEXT:= 1; "GO TO" FAIL "END";
       STAP:= VECVEC(1, N, 0, SOL, SOL);
       RHO:= 2; NORMX:= VECVEC(1, N, 0, X, X);
       "IF" STAP > RELTOLPAR * NORMX + ABSTOLPAR
          "OR" IT = 1 "AND" STAP > 0 "THEN"
```

```
        "BEGIN""FOR" INR:= 0, INR + 1
           "WHILE""IF" INR = 1 "THEN" DAMPING ON "OR" RES2 >= RN
              "ELSE" = CONV "AND" (RN <= RES1 "OR" RES2 < RES1) "DO"
           "BEGIN""COMMENT" DAMPING STOPS WHEN
              R0 > R1 "AND" R1 <= R2 (BEST RESULT IS X1, R1)
              WITH  X1 = X0 + I * DX, I:= 1, .5, .25, .125, ETC.  ;
              RHO:= RHO / 2; "IF" INR > 0 "THEN"
              "BEGIN" RES1:= RES2; DUPVEC(1, N, 0, F, FU2);
                 DAMPING ON:= INR > 1
              "END";
              "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
                 PR[I]:= X[I] - RHO * SOL[I];
              TEST THF:= FUNCT(N, PR, FU2); FEVAL:= FEVAL + 1;
              "IF" "NOT" TEST THF "THEN"
              "BEGIN" TEXT:= 2; "GO TO" FAIL "END";
              RES2:= VECVEC(1, N, 0, FU2, FU2); CONV:= INR >= INRMAX
           "END" DAMPING OF STEP VECTOR;
           "IF" CONV "THEN"
           "BEGIN""COMMENT" RESIDUE CONSTANT; MIT:= MIT + 1;
              "IF" MIT < TIM "THEN" CONV:= "FALSE"
           "END" "ELSE" MIT:= 0;
           "IF" INR > 1 "THEN"
           "BEGIN" RHO:= RHO * 2; ELMVEC(1, N, 0, X, SOL, - RHO);
              RN:= RES1; "IF" INR > 2 "THEN" OUT[7]:= IT
           "END""ELSE"
           "BEGIN" DUPVEC(1, N, 0, X, PR); RN:= RES2;
              DUPVEC(1, N, 0, F, FU2)
           "END";

           "IF" RN <= ABSTOLRES "THEN"
           "BEGIN" TEXT:= 4; ITMAX:= IT "END""ELSE"
           "IF" CONV "AND" INRMAX > 0 "THEN"
           "BEGIN" TEXT:= 5; ITMAX:= IT "END"
           "ELSE" DUPVEC(1, N, 0, SOL, F)
        "END" ITERATION WITH DAMPING AND TESTS "ELSE"
        "BEGIN" TEXT:= 6; RHO:= 1; ITMAX:= IT "END"
     "END" OF ITERATIONS;

FAIL :
     OUT[1]:= SQRT(STAP) * RHO; OUT[2]:= SQRT(RN); OUT[4]:= FEVAL;
     OUT[6]:= TEXT; OUT[8]:= AUX[3]; OUT[9]:= AUX[5]
"END" PROGRAM B;
```

```
"COMMENT" DISCRETIZED NEWTON METHOD OF PANKIEWICZ, PROGRAM E;
"PROCEDURE" PROGRAM E(N, X, F, FUNCT, IN, OUT);
"VALUE" N; "INTEGER" N; "ARRAY" X, F, IN, OUT;
"BOOLEAN" "PROCEDURE" FUNCT;
"COMMENT" ALGORITHM 378 FROM CACM BY W. PANKIEWICZ, ALGOR 378 SOLVES
    A SYSTEM OF NONLINEAR EQUATIONS;
"BEGIN"
   "REAL" "PROCEDURE" VECVEC(L,U,S,A,B); "CODE" 34010;
   "INTEGER" "PROCEDURE" NIELIN (N, H, W, EPS, Y, Z);
   "VALUE" N, H, W, EPS; "INTEGER" N; "REAL" H, W, EPS;
   "ARRAY" Y, Z;
   "BEGIN" "INTEGER" M, I, K; "REAL" ALPHA, R; "BOOLEAN" B1, B2;
      "ARRAY" A [1 : N, 1 : N + 1], V [1 : N], AUX[1:7];


      "PROCEDURE" GSSSOL (A, N, AUX, B);
      "CODE" 34232;

      "PROCEDURE" GAUSS (U, A, Y); "INTEGER" U; "ARRAY" A, Y;
      "BEGIN" "INTEGER" I, J; "ARRAY" HA [1 : U, 1 : U], HY [1 : U];
         "FOR" I:= 1 "STEP" 1 "UNTIL" U "DO"
         "BEGIN" HY [I]:= A [I, U + 1];
            "FOR" J:= 1 "STEP" 1 "UNTIL" U
            "DO" HA [I, J]:= A [I, J]
         "END"; AUX[2]:= "=10; AUX[4]:= 8;
         GSSSOL(HA, U, AUX, HY);
         "IF" AUX [3] < U "THEN" "GOTO" ERROR;
         "FOR" I:= 1 "STEP" 1 "UNTIL" U "DO" Y [I]:= HY [I]
      "END";
      "PROCEDURE" FC(X,F);
      "ARRAY" X, F;
```

```
      "BEGIN" CNT:= CNT + 1;
         "IF" CNT > IN [4] "THEN"
         "BEGIN" NIELIN:= - 4; "GOTO" END "END";
         "IF" * FUNCT (N, X, F) "THEN"
         "GOTO" ALARM
      "END" FC;
      M:= 0;
   POCZATEK: B1:= "TRUE";  FC(Y,Z);
      "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
      "BEGIN" A [I, N + 1]:= R:= Z [I]; R:= ABS (R);
         B1:= B1 "AND" R < EPS;
      "END";
      "IF" B1 "THEN" "GOTO" KONIEC;
      "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
      "BEGIN" R:= Y [I]; Y [I]:= R + H; FC (Y, Z);
         "FOR" K:= 1 "STEP" 1 "UNTIL" N "DO" A [K, I]:= Z [K];
         Y [I]:= R
      "END";
      GAUSS (N, A, V); ALPHA:= 1;
      "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO" ALPHA:= ALPHA - V [I];
      "IF" ALPHA = 0 "THEN" "GOTO" ALPH; ALPHA:= H / ALPHA;
      "FOR" I:= 1 "STEP" 1 "UNTIL" N
      "DO" Y [I]:= Y [I] - V [I] * ALPHA; H:= H * W;
      M:= M + 1;
      "GOTO" POCZATEK;
   KONIEC: NIELIN:= M; "GOTO" END;
   ALARM: NIELIN:= - 1; "GOTO" END;
   ERROR: NIELIN:= - 2; "GOTO" END;
   ALPH: NIELIN:= - 3;
   END: OUT [4]:= M + OUT[4];
   "END" NIELIN;

   "INTEGER" TEL, CNT, ITT;
   OUT[4]:=0;
   TEL:= CNT:= 0;
REPEAT: ITT:= NIELIN (N, IN [9], IN [10], IN [1], X, F);
   TEL:= TEL + 1;
   "IF" (ITT = - 2 "OR" ITT = - 3) "AND" TEL < 3
   "THEN" "GOTO" REPEAT;
   "IF" ITT > 0 "THEN" OUT [5]:= 0 "ELSE" OUT[5]:=-ITT;
   OUT [1]:= SQRT (VECVEC (1, N, 0, X, X));
   OUT [2]:= SQRT (VECVEC (1, N, 0, F, F)); OUT [3]:= CNT;
   OUT [6]:= TEL
"END" PROGRAM E;
```

```
"COMMENT" MODIFIED GENERALIZED SECANT METHOD. PROGRAM F;
"PROCEDURE" PROGRAM F(N, X, F, FUNCT, IN, OUT);
"VALUE" N; "INTEGER" N; "ARRAY" X, F, IN, OUT;
"BOOLEAN" "PROCEDURE" FUNCT;
"COMMENT" ALGORITHM 12 FROM COMPUTING BY H. SCHWETLICK,
ALGOR 12 SOLVES A SYSTEM OF NON LINEAR EQUATIONS;
"BEGIN"
   "PROCEDURE" FU (N,X,FA); "VALUE" N; "INTEGER" N;
   "ARRAY" X,FA;
   "BEGIN" CNT:=CNT+1;
      "IF" CNT > IN[4] "THEN" "BEGIN" OUT[5]:=4; "GOTO" L4 "END";
      "IF" *FUNCT(N,X,FA) "THEN" "BEGIN" OUT[5]:=5; "GOTO" L4 "END";
   "END" FU;
   "REAL" "PROCEDURE" VECVEC(L,U,S,A,B); "CODE" 34010;
   "REAL" RES; "INTEGER" CNT,IT,I; "ARRAY" Y [1:N];
   "SWITCH" DIV:=L1, L2, L3;
   "PROCEDURE" REGULA(D,FU,EPS,PIVOT,IMAX) TRANS: (X,Y) EXIT: (DIV);
   "VALUE" D,EPS,PIVOT,IMAX;
   "INTEGER" D,IMAX;
   "REAL" EPS,PIVOT;
   "ARRAY" X,Y;
   "PROCEDURE" FU;
   "SWITCH" DIV;
   "BEGIN" "INTEGER" I,J,K,L,P,Q,NR,KMAX;
      "REAL" G,H;
      "BOOLEAN" TEST;
      "ARRAY" FF[1:D],DELTA[1:D,1:D];
      "INTEGER" "ARRAY" PERM[1:D];
      "COMMENT" BESTIMMUNG VON KMAX;
      K:= 0; H:= 0; G:= LN(1.618033989)/D;
      "FOR" K:= K+1 "WHILE" G >= H "DO"
      "BEGIN" H:= G; G:= LN((SQRT((K+2) * 5+5) + K+1) * 0,5)/(K+D)
      "END" K;
      KMAX:= K+2;
      FU(D,X,F);
      "COMMENT" ITERATION$BEGINN;
      "FOR" L:= 1 "STEP" 1 "UNTIL" IMAX "DO"
```

```
"BEGIN" "COMMENT" BERECHNUNG DER STEIGUNG DELTA;
   "FOR" K:= D "STEP" -1 "UNTIL" 1 "DO"
   "BEGIN" G:= Y[K] - X[K]; X[K]:= Y[K];
      FU(D,X,FF); TEST:= "TRUE";
   CORR: "FOR" I:= 1 "STEP" 1 "UNTIL" D "DO"
      "BEGIN" H:= FF[I] - F[I];
         "IF" ABS(H) + ABS(G) = ABS(H) "THEN"
         "BEGIN" "COMMENT" KORREKTUR VON X;
            "IF" = TEST "THEN"
            "BEGIN" OUT[4]:=L; "GOTO" DIV[1] "END";
            G:=Y[K] * EPS + EPS * EPS; X[K]:=X[K]-G; FU(D,X,F);
            X[K]:= Y[K]; TEST:= "FALSE"; "GOTO" CORR
         "END" KORREKTUR;
         DELTA[I,K]:= H/G; F[I]:= FF[I]
      "END" I
   "END" K;
   "COMMENT" DREIECKZERLEGUNG VON DELTA;
   "FOR" K:= 1 "STEP" 1 "UNTIL" D "DO" PERM[K]:= K;
   "FOR" P:= 1 "STEP" 1 "UNTIL" D-1 "DO"
   "BEGIN" H:= 0;
      "FOR" K:= P "STEP" 1 "UNTIL" D "DO"
      "BEGIN" G:= ABS(DELTA[K,P]);
         "IF" G > H "THEN"
         "BEGIN" H:= G; Q:= K "END"
      "END" PIVOTSUCHE;
      "IF" H < ABS(PIVOT) "THEN"
      "BEGIN" OUT[4]:=L; "GOTO" DIV[2] "END";
      NB:= PERM[Q]; H:= 1/DELTA[Q,P];
      "FOR" K:= 1 "STEP" 1 "UNTIL" D "DO" FF[K]:= DELTA[Q,K];
      J:=D;
      "FOR" I:=Q "STEP" -1 "UNTIL" P "DO"
      "BEGIN" "IF" I=Q "THEN" "GOTO" WEITER;
         "FOR" K:=1 "STEP" 1 "UNTIL" P-1 "DO"
         DELTA[J,K]:=DELTA[I,K];
         G:= DELTA[J,P]:= DELTA[I,P] * H;
         "FOR" K:= P+1 "STEP" 1 "UNTIL" D "DO"
         DELTA[J,K]:= DELTA[I,K] - FF[K] * G;
         PERM[J]:= PERM[I]; J:= J-1;
```

```
    WEITER: "END" I;
            "FOR" K:= 1 "STEP" 1 "UNTIL" D "DO" DELTA[P,K]:= FF[K];
            PERM[P]:= NR
        "END" P, DREIECKZERLEGUNG;
        "COMMENT" STUFENITERATION;
        "FOR" I:= 0 "STEP" 1 "UNTIL" KMAX "DO"
        "BEGIN" "IF" I > 0 "THEN" FU(Q,Y,F);
            "FOR" K:= 1 "STEP" 1 "UNTIL" D "DO"
            "BEGIN" J:= PERM[K]; FF[K]:= F[J]
            "END" K, PERMUTATION DER RECHTEN SEITE;
            "COMMENT" ELIMINATION DER RECHTEN SEITE;
            "FOR" P:= 2 "STEP" 1 "UNTIL" D "DO"
            "BEGIN" H:= FF[P];
                "FOR" K:= 1 "STEP" 1 "UNTIL" P-1 "DO" H:= H - DELTA[P,K]
                * FF[K]; FF[P]:= H
            "END" P;
            "FOR" P:= D "STEP" -1 "UNTIL" 1 "DO"
            "BEGIN" H:= FF[P];
                "FOR" K:= P+1 "STEP" 1 "UNTIL" D "DO" H:= H-DELTA[P,K]
                * FF[K]; FF[P]:= H/DELTA[P,P]
            "END" P, ELIMINATION DER RECHTEN SEITE;
            "COMMENT" ABBRUCHTEST;
            RES:=SQRT(VECVEC(1,D,0,F,F)); TEST:= RES <= IN[1];
            "FOR" K:= 1 "STEP" 1 "UNTIL" D "DO"
            "BEGIN" H:= FF[K]; G:= X[K]:= Y[K]; G:= Y[K]:= G-H;
                "IF" ABS(H) > ABS(EPS * G) + ABS(EPS) "THEN"
                TEST:= "FALSE"
            "END" K;
            "IF" TEST "THEN" "GOTO" SCHLUSS
        "END" I, STUFENITERATION;
      "END" L;
      OUT[4]:=L-1; "GOTO" DIV[3];
SCHLUSS: OUT[2]:=SQRT(VECVEC(1,D,0,F,F)); OUT[4]:=L
   "END" REGULA;

   "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
   Y[I]:= X[I] * (1 + IN[8]) + IN[8]; CNT:= 0;
   REGULA(N, FU, IN[8], IN[0], IN[4], X, Y, DIV);
   OUT[5]:= 0; "GOTO" L4;
L1: OUT[5]:= 1; "GOTO" L4;
L2: OUT[5]:= 2; "GOTO" L4;
L3: OUT[5]:= 3; "GOTO" L4;
L4: OUT[1]:= SQRT(VECVEC(1, N, 0, Y, Y)); OUT[3]:= CNT
"END" PROGRAM F;
```

```
"COMMENT" METHOD OF DULLFY AND PITTEWAY, BASED ON GENERALIZED SECANT
METHOD, PROGRAM G;
"PROCEDURE" PROGRAM G(N, X, F, FUNCT, IN, OUT); "VALUE" N; "INTEGER" N;
"ARRAY" X, F, IN, OUT; "BOOLEAN" "PROCEDURE" FUNCT;
"BEGIN" "REAL" "PROCEDURE" VECVEC(L, U, I, A, B); "CODE" 34010;
   "PROCEDURE" FC(F, X); "ARRAY" F, X;
   "BEGIN" "IF" FUNCT(N, X, F) "THEN"
      "BEGIN" OUT[5]:= 5; "GOTO" EXT "END";
      CNT:= CNT + 1; "IF" CNT > IN[4] "THEN"
      "BEGIN" OUT[5]:= 4; "GOTO" EXT "END"
   "END" FC;

   "INTEGER" CNT, COUNT; "ARRAY" ACCEST[1:N];

   "PROCEDURE" NDINVT(FUNCTIONS,INITSTEP,ERROR,CYCLES,X,F,ACCEST,N);
   "VALUE" N; "PROCEDURE" FUNCTIONS; "REAL" INITSTEP,ERROR;
   "INTEGER" CYCLES,N; "ARRAY" X,F,ACCEST;
   "BEGIN" "REAL" WORK,SUMSQRES,PREVRES;
      "INTEGER" I,J;
      "BOOLEAN" SWITCH;
      "ARRAY" PREVF[1:N],COPYDELF[1:N,1:N],DELX,DELF[1:N,1:N+1],
      AUX[1:7];
      "PROCEDURE" GSSOL(A,N,AUX,B); "CODE" 34232;
      AUX[2]:="-10; AUX[4]:=8; COUNT:=0; SUMSQRES:=1"30;
      FUNCTIONS(PREVF,X);
      "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
      "BEGIN" X[I]:= X[I] + INITSTEP;
         FUNCTIONS(F,X);
         "FOR" J:= 1 "STEP" 1 "UNTIL" N "DO"
         "BEGIN" DELF[I,J]:= F[J] - PREVF[J];
            "COMMENT" IF THE REMARK OF VANDERGRAFT AND MESZTENYI SHOULD
            BE INCORPORATED, THEN THE LAST STATEMENT SHOULD START WITH
            DELF[J,I]:= ;
            DELX[I,J]:= 0;
         "END" DIFFERENCING INITIAL POINT;
         DELX[I,I]:= INITSTEP;
         X[I]:= X[I] - INITSTEP;
      "END" SETTING UP THE INITIAL MATRIX OF POINTS;
```

```
   ITERATE;
      SWITCH:= "TRUE";
      PREVRES:= SUMSQRES;
   TRYAGAIN;
      "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
      "BEGIN" F[I]:= PREVF[I];
         "FOR" J:= 1 "STEP" 1 "UNTIL" N "DO" COPYDELF[I,J]:= DELF[I,J]
      "END" COPYING DELF FOR DESTRUCTIVE USE IN PROCEDURE EQNSOLVE;

      GSSOL(COPYDELF,N,AUX,F); "IF" AUX[3]<N "THEN" "GOTO" INLINE;
      SUMSQRES:= 0;
      "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
      "BEGIN" WORK:= 0;
         "FOR" J:= 1 "STEP" 1 "UNTIL" N "DO"
         WORK:= WORK - DELX[I,J] * F[J]; ACCEST[I]:= WORK;
         X[I]:= X[I] + WORK;
         SUMSQRES:= SUMSQRES + WORK * WORK
      "END" CALCULATION OF NEXT POINT;
      COUNT:= COUNT + 1;
      FUNCTIONS(F,X);
      "IF" COUNT > CYCLES "THEN" "BEGIN" OUT[5]:=3; "GOTO" EXIT "END";
      "IF" SUMSQRES < ERROR * ERROR "AND"
         (ERROR > 0 "OR" SUMSQRES > PREVRES) "THEN"
      "BEGIN" OUT[5]:=0; "GOTO" EXIT "END";
      "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
      "BEGIN" WORK:= F[I] - PREVF[I];
         PREVF[I]:= F[I];
         "FOR" J:= N "STEP" -1 "UNTIL" 1 "DO"
         "BEGIN" DELX[I,J+1]:= DELX[I,J] - ACCEST[I];
            DELF[I,J+1]:= DELF[I,J] - WORK
         "END" CALCULATION OF NEW DIFFERENCES;
         DELX[I,1]:= -ACCEST[I];
         DELF[I,1]:= -WORK
      "END" MOVING POINTS UP ONE PLACE IN TABLES;
      "GOTO" ITERATE;
   INLINE;
      "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
      "BEGIN" DELX[I,N]:= DELX[I,N+1];
         DELF[I,N]:= DELF[I,N+1]
      "END" DISCARDING ALTERNATIVE POINT;
      SWITCH:= ~ SWITCH;
      "IF" SWITCH "THEN" OUT[5]:=1 "ELSE" "GOTO" TRYAGAIN;
   EXIT;
   "END" NDINVT;

   CNT:= 0; NDINVT(FC, IN[0], IN[2], IN[4], X, F, ACCEST, N);
EXT: OUT[1]:= SQRT(VECVEC(1, N, 0, ACCEST, ACCEST));
   OUT[3]:= CNT; OUT[2]:= SQRT(VECVEC(1, N, 0, F, F));
   OUT[4]:= COUNT
"END" PROGRAM G;
```

```
"COMMENT" QUASI-NEWTON METHOD OF BROYDEN, PROGRAM H;
"PROCEDURE" PROGRAM H(N, X, F, FUNCT, IN, OUT);
"VALUE" N; "INTEGER" N; "ARRAY" X, F, IN, OUT;
"BOOLEAN" "PROCEDURE" FUNCT;
"BEGIN" "INTEGER" I, J, FCOUNT, MAXF, ERR, IT;
    "REAL" SA, TOLRES, RELTOL, ABSTOL, RES;
    "ARRAY" Y, P, V[1:N], H[1:N,1:N];
    "SWITCH" LABEL:= LB1, LB2, LB3, LB4, LB5;

    "REAL" "PROCEDURE" VECVEC(L, U, S, A, B); "CODE" 34010;
    "REAL" "PROCEDURE" MATVEC(L, U, I, A, B); "CODE" 34011;
    "REAL" "PROCEDURE" TAMVEC(L, U, I, A, B); "CODE" 34012;
    "PROCEDURE" DUPVEC(L, U, S, A, B); "CODE" 31030;
    "PROCEDURE" ELMVEC(L, U, S, A, B, X); "CODE" 34020;
    "PROCEDURE" ELMCOLVEC(L, U, I, A, B, X); "CODE" 34022;

    "PROCEDURE" STEP(TP1, TP2); "VALUE" TP1, TP2;
    "INTEGER" TP1, TP2;
    "BEGIN" "INTEGER" I; "REAL" SBB; "ARRAY" SB[1:N];
        ELMVEC(1, N, 0, X, P, 1);
        DUPVEC(1, N, 0, V, F); FUNCT(N, X, F);
        FCOUNT:= FCOUNT + 1;
        DUPVEC(1, N, 0, Y, F);
        ELMVEC(1, N, 0, Y, V, -1);
        "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
        "BEGIN" SBB:= SB[I]:= MATVEC(1, N, I, H, Y);
            V[I]:= SBB - P[I]
        "END";
        SA:=VECVEC(1, N, 0, SB, P);
        "IF" SA = 0 "THEN" "GOTO" LABEL[TP2];
        "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
        ELMCOLVEC(1, N, I, H, V, -TAMVEC(1, N, I, H, P
        ) / SA)
    "END" STEP;
    RELTOL:= IN[1]; ABSTOL:= IN[2]; TOLRES:= IN[3];
    MAXF:= IN[4];
    FUNCT(N, X, F); FCOUNT:= 1; IT:= ERR:= 0;
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" P[I]:= 0; H[I,I]:= 1;
        "FOR" J:= I + 1 "STEP" 1 "UNTIL" N "DO"
        H[I,J]:= H[J,I]:= 0
    "END" INITIALIAZATION;
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    "BEGIN" P[I]:= "-6 * ABS(X[I]) + "-10; STEP(5, 4);
        P[I]:= 0
    "END" CALCULATION OF INITIAL ITERATION MATRIX;
REPEAT: IT:= IT + 1;
    "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"
    P[I]:= -MATVEC(1, N, I, H, F);
    STEP(3, 2);
    RES:= SQRT(VECVEC(1, N, 0, F, F));
    "IF" SQRT(VECVEC(1, N, 0, P, P)) <
    SQRT(VECVEC(1, N, 0, X, X)) * RELTOL + ABSTOL "AND"
    RES < TOLRES "THEN" "GOTO" EXIT;
    "IF" FCOUNT < MAXF "THEN" "GOTO" REPEAT;
LB1: ERR:= 1; "GOTO" EXIT;
LB2: ERR:= 5; "GOTO" EXIT;
LB3: ERR:= 2; "GOTO" EXIT;
LB4: ERR:= 6; "GOTO" EXIT;
LB5: ERR:= 7; "GOTO" EXIT;
EXIT: OUT[1]:= SQRT(VECVEC(1, N, 0, P, P)); OUT[2]:= RES;
    OUT[3]:= FCOUNT; OUT[4]:= IT; OUT[5]:= ERR
"END" PROGRAM H;
```

```
"COMMENT" BROWNS METHOD OF COMPONENT-WISE APPROXIMATION,
PROGRAM I;
"PROCEDURE" PROGRAM I(N, X, FA, FUCOM, IN, OUT);
"VALUE" N; "INTEGER" N; "ARRAY" X, FA, IN, OUT;
"BOOLEAN" "PROCEDURE" FUCOM;
"COMMENT" ALGORITHM 316 FROM CACM BY K.M. BROWN,
    ALGOR 316 SOLVES A SYSTEM OF NON LINEAR EQUATIONS;
"BEGIN" "INTEGER" I,J,K,M,ITEMP,JSUB,KMAX,KPLUS,TALLY,TIM,CNT,
    MAXIT, ERR, FMAX;
    "REAL" F,H,HOLD,FPLUS,DERMAX,TEST,FACTOR,PT,HCOE,XI,TM,
    RELTOL, ABSTOL;
    "INTEGER" "ARRAY" POINTER[1:N,1:N],ISUB[1:N];
    "ARRAY" TEMP,PART[1:N],COE[1:N,1:N + 1];

    "REAL" "PROCEDURE" VECVEC(L, U, S, A, B); "CODE" 34010;
    "PROCEDURE" DUPVEC(L, U, S, A, B); "CODE" 31030;
    "PROCEDURE" FLMVEC(L, U, S, A, B, X); "CODE" 34020;
    "PROCEDURE" BACK SUBST(K); "VALUE" K; "INTEGER" K;
    "BEGIN" "INTEGER" KM,KMAX,JSUB; "REAL" XKMAX;
        "FOR" KM:= K "STEP" - 1 "UNTIL" 2 "DO"
        "BEGIN" KMAX:= ISUB[KM - 1]; XKMAX:= 0;
            "FOR" J:= KM "STEP" 1 "UNTIL" N "DO"
            "BEGIN" JSUB:= POINTER[KM,J];
                XKMAX:= XKMAX + COE[KM  - 1,JSUB] * X[JSUB]
            "END";
            X[KMAX]:= XKMAX + COE[KM - 1,N + 1]
        "END"
    "END" BACK SUBST;

    "PROCEDURE" THEORFU(N, K, X, F); "VALUE" N, K;
    "INTEGER" N, K; "REAL" F; "ARRAY" X;
    "BEGIN" CNT:= CNT + 1; "IF" CNT > FMAX "THEN"
        "BEGIN" ERR:= 1; "GOTO" EXIT "END";
        "IF" ~ FUCOM(N, K, X, F) "THEN"
        "BEGIN" ERR:= 2; "GOTO" EXIT "END"; FA[K]:= F
    "END" THEORFU;

    RELTOL:= IN[1]; ABSTOL:= IN[2];
    FMAX:= IN[4]; MAXIT:= IN[4] * 2 / N; ERR:= CNT:= 0;
    "FOR" M:= - 1 "STEP" 1 "UNTIL" MAXIT "DO"
    "BEGIN" "FOR" J:= 1 "STEP" 1 "UNTIL" N "DO"
        POINTER[1, J]:= J;
        "FOR" K:= 1 "STEP" 1 "UNTIL" N "DO"
        "BEGIN" "IF" K > 1 "THEN" BACK SUBST(K);
            THEORFU(N,K,X,F); FACTOR:= "-3;
        AGAIN; TALLY:= 0;
            "FOR" I:= K "STEP" 1 "UNTIL" N "DO"
```

```
            "BEGIN" ITEMP:= POINTER[K,I]; HOLD:= X[ITEMP];
                H:= FACTOR * HOLD; "IF" H = 0 "THEN" H:= FACTOR;
                X[ITEMP]:= HOLD + H;
                "IF" K > 1 "THEN" BACK SUBST(K);
                THEORFU(N,K,X,FPLUS);
                PT:= PART[ITEMP]:= (FPLUS - F) / H;
                X[ITEMP]:= HOLD;
                "IF" ABS(F / PT) > "20 "THEN" TALLY:= TALLY + 1
            "END";

            "IF" TALLY > N - K "THEN"
            "BEGIN" FACTOR:= FACTOR * 10;
                "IF" FACTOR > .5 "THEN"
                "BEGIN" ERR:= 4; "GOTO" EXIT "END";
                "GOTO" AGAIN
            "END";
            "IF" K = N "THEN"
            "BEGIN" "IF" ABS(PT) = 0 "THEN"
                "BEGIN" ERR:= 5; "GOTO" EXIT "END";
                HCOE:= 0; KMAX:= ITEMP; "GOTO" END K
            "END";
            KMAX:= POINTER[K,K]; DERMAX:= ABS(PART[KMAX]);
            KPLUS:= K + 1;
            "FOR" I:= K PLUS "STEP" 1 "UNTIL" N "DO"
            "BEGIN" JSUB:= POINTER[K,I]; TEST:= ABS(PART[JSUB]);
                "IF" TEST < DERMAX "THEN"
                POINTER[KPLUS,I]:=JSUB "ELSE"
                "BEGIN" DERMAX:= TEST; POINTER[KPLUS,I]:= KMAX;
                    KMAX:= JSUB
                "END"
            "END";
            "IF" DERMAX = 0 "THEN"
            "BEGIN" ERR:= 3; "GOTO" EXIT "END";
            ISUB[K]:= KMAX; HCOE:= 0;
            "FOR" J:= KPLUS "STEP" 1 "UNTIL" N "DO"
            "BEGIN" JSUB:= POINTER[KPLUS,J]; PT:= PART[JSUB];
                COE[K,JSUB]:= - PT / PART[KMAX];
                HCOE:= HCOE + PT * X[JSUB]
            "END";
        END K; HCOE:= COE[K,N + 1]:= (HCOE - F) / PART[KMAX] +
            X[KMAX]
        "END" K;
        X[KMAX]:= HCOE; "IF" N > 1 "THEN" BACKSUBST(N);
        "IF" M = 1 "THEN" "GOTO" JUMP;
        ELMVEC(1, N, 0, TEMP, X, -1);
        "IF" SQRT(VECVEC(1, N, 0, TEMP, TEMP)) <
        SQRT(VECVEC(1, N, 0, X, X)) * RELTOL + ABSTOL
        "THEN" "GOTO" EXIT;
    JUMP: DUPVEC(1, N, 0, TEMP, X);
    "END" M;
EXIT: OUT[1]:= SQRT(VECVEC(1, N, 0, TEMP, TEMP));
    OUT[3]:= CNT / N; OUT[5]:= ERR;
    OUT[4]:= "IF" ERR = 0 "THEN" M + 2 "ELSE" M + 1
"END" PROGRAM I;
```