NUMERICAL SOLUTION OF SYSTEMS OF NONLINEAR EQUATIONS

ACADEMISCH PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR
IN DE WISKUNDE EN NATUURWETENSCHAPPEN
AAN DE UNIVERSITEIT VAN AMSTERDAM,
OP GEZAG VAN DE RECTOR MAGNIFICUS
DR J. BRUIJN,
HOOGLERAAR IN DE FACULTEIT DER LETTEREN,
IN HET OPENBAAR TE VERDEDIGEN
IN DE AULA DER UNIVERSITEIT
(TIJDELIJK IN DE LUTHERSE KERK, INGANG SINGEL 411, HOEK SPUI)
OP WOENSDAG 16 APRIL 1980 DES NAMIDDAGS TE 4.00 UUR

DOOR

JACOBUS CORNELIS PETRUS BUS

GEBOREN TE HILLEGOM

1980

MATHEMATISCH CENTRUM, AMSTERDAM

PROMOTOREN: PROF.DR. T.J. DEKKER
            PROF.DR. M.N. SPIJKER
COREFERENT: PROF.DR. W.W.E. WETTERLING.

ACKNOWLEDGEMENTS

First of all I want to express my gratitude to promotor prof.dr. T.J. Dekker. He introduced me to the field of numerical mathematics and guided me through many research problems. I am greatly indebted to his constant efforts to provide valuable criticism and suggestions.

I am also grateful to promotor prof.dr. M.N. Spijker. His careful criticism and suggestions during my work on this thesis are greatly appreciated.

I want to thank prof.dr. W.W.E. Wetterling for his willingness to be coreferent of my thesis.

Furthermore, I like to mention the stimulating discussions with prof.dr. F.A. Lootsma, dr. J.L. de Jong, K. Schittkowski and members of the Committee on Algorithms of the Mathematical Programming Society about developing methodologies for testing software for nonlinear programming problems.

Special thanks are due to M. de Leeuw, who assisted me with the programming. I appreciate her excellent programming and her patient willingness to accept my changing ideas about software and algorithms.

I want to thank dr. J.H. van Schuppen for his careful reading of the manuscript.

The research described in this thesis has been done at the Mathematisch Centrum, Amsterdam. I wish to express my gratitude to the governing board of the "Stichting Mathematisch Centrum" for giving me the opportunity to carry out these investigations.

As far as the technical realization of this book is concerned, I would like to thank A.K. van den Berg for her careful and diligent typing of the manuscript; T. Baanders for the cover design and the figures, D. Zwarst for organizing the production process and E. Michel, J. Schipper, J. Suiker and J. van der Werf for their quick and accurate printing.

CONTENTS

NOTATIONS

$\mathbb{R}^n$	the n-dimensional real space.
$L(\mathbb{R}^n)$	the linear space of linear operators from $\mathbb{R}^n$ to $\mathbb{R}^n$.
$\lVert . \rVert$	the euclidean norm for vectors and the spectral norm for matrices.
$[x,y]$	if $x,y \in \mathbb{R}^n$ then the usual inner product in $\mathbb{R}^n$, if $x,y \in \mathbb{R}$ then a closed interval in $\mathbb{R}$.
$U(x,\delta)$	$= \{y \mid y \in \mathbb{R}^n, \lVert x-y \rVert < \delta\}$.
$\bar{U}$, int(U)	for $U \subset \mathbb{R}^n$, are the closure and interior, respectively, of U.
F, n, D	F is a function, with order n, on an open nonempty domain $D \subset \mathbb{R}^n$, $F : D \to \mathbb{R}^n$ (see notation 1.3).
$J(x)$	$= \frac{d}{dx} F(x)$.
$S_F(x,A)$	the levelset of F with respect to A and x (see definition 1.4).
$\mathcal{F}$	The class of functions F which have a continuous Fréchet derivative on D (see definition 1.18).
$\mathcal{U}(u)(B,y,x)$	a jacobian update function from Broyden's class (see formula (3.17)).
$\mathcal{V}(u)(H,y,x)$	an inverse-jacobian update function from Broyden's class (see formula (3.18)).
ε	the machine precision (see notation 3.8)
$fl_\varepsilon(.)$	the expression within the parentheses computed with machine precision ε.
$A\backslash b$	the solution of the linear system $Ax = b$, computed by triangular decomposition of A, followed by forward and backward substitution (see formula (1.13)).
A^+	the generalized inverse of matrix A.
U_n	a standard time unit of order n (see definition 7.1).
$\mathcal{C}$	a class of problems of solving a system of nonlinear equations, satisfying certain conditions (see definition 7.3).

$T(P,p)$	the standard time required by program P to solve problem p (see formula (7.1)).
$\mathcal{E}(n,t_F,t_J)$	the relative efficiency of a program, for solving problems of order n and with function evaluation time t_F and jacobian evaluation time t_J (see description 7.13).
$\mathcal{T}_n$	representative set of test problems of order n in $\mathcal{C}$ (see subsection 7.5.1).

INTRODUCTION

In this thesis we treat the problem of analysis and design of methods for the numerical solution of systems of nonlinear equations. Solving systems of nonlinear equations often arises in problems of numerical analysis, such as two-point boundary value problems, elliptic boundary value problems, integral equations, two-dimensional variational problems or optimal control problems. These problems motivate a detailed analysis of methods for solving systems of nonlinear equations.

An analysis of systems of nonlinear equations is a prerequisite for any synthesis of numerical methods. Basic to this analysis are questions of existence and uniqueness of solutions. These topics are treated in chapter 2. Our method of analyzing existence and uniqueness issues is based on a theory given by RHEINBOLDT [1969] (see also ORTEGA & RHEINBOLDT [1970]). The results are given in such a way that they fit the framework of the convergence theory presented in later chapters.

We restrict attention to methods which can be classified as *Newton-like methods,* which notion is defined in chapter 4. These methods are *local methods,* i.e. methods which require an initial estimate of a solution which is relatively close to an exact solution, in a sense which depends on the smoothness of the problem . These methods produce at most one approximation to a solution. Almost all local methods currently used are Newton-like methods. For these methods we present a comprehensive convergence theory in chapter 5. This leads, in chapter 6, to the construction of new Newton-like methods. As basic references for this convergence theory we mention: KANTOROVICH & AKILOW [1964], RHEINBOLDT [1969], ORTEGA & RHEINBOLDT [1970], DENNIS [1971], DENNIS & MORÉ [1974], DEUFLHARD [1974a,1974b] and DEUFLHARD & HEINDL [1979]. In fact, the theory about *global convergence* (section 5.2) is an extension of Deuflhard's theory for Newton's method. The theory about *semi-local* (section 5.3) and *local convergence* (section 5.4) is essentially based on the well known Newton-Kantorovich theorem and extensions of this result given by Dennis, Deuflhard and Heindl. The notion of *affine invariancy* (invariancy of the results with respect to affine transformation of the function), which was first introduced in this field by Deuflhard, plays an important role in the convergence theory.

In the study of numerical methods we distinguish two issues. First the investigation of these methods from a theoretical viewpoint. This is done primarily by studying convergence behaviour (chapter 5). Secondly, a thorough comparative study, based on practical tests, is performed. This study meets as much as possible the requirements as given in CROWDER, DEMBO & MULVEY [1977] about the design of computational experiments. This comparison will not only involve the Newton-like methods described in this thesis, but also the method of BROWN [1969] which is known to be competitive with certain Newton-like methods. Brown's method is not a Newton-like method according to our definition. It is based on successive linear interpolation of the nonlinear equations separately, while Newton-like methods handle these equations simultaneously. We refer to Brown's paper for a description of this method. We also mention some efficient implementations of this method described in BROWN [1973], BRENT [1973a], GAY [1975] and MORE & COSNARD [1979]. The experimental design, as well as the actual experiments are discussed in chapter 7. Based on the experimental as well as theoretical evaluation of the algorithms, we present in section 7.8 two new poly-algorithms (combinations of Newton-like algorithms) for solving systems of nonlinear equations. In this thesis we use ALGOL 68 as a reference language in order to provide an unambiguous description of the various algorithms. For practical reasons, the experiments are performed in ALGOL 60. An ALGOL 60 implementation of the poly-algorithms of chapter 7, together with a users manual is given in BUS [1980].

CHAPTER 1

PRELIMINARIES

1.1. ANALYSIS

Let $x,y \in \mathbb{R}^n$. With $[x,y]$ we denote the usual innerproduct of x and y. Unless specified otherwise, we use the euclidean norm:

$$(1.1) \qquad \|x\| = \sqrt{[x,x]}, \quad x \in \mathbb{R}^n.$$

With $L(\mathbb{R}^n)$ we denote the space of linear operators from $\mathbb{R}^n$ to $\mathbb{R}^n$ with spectral norm

$$(1.2) \qquad \|A\| = \sup_{\substack{x\neq 0\\ x\in\mathbb{R}^n}} \|Ax\|/\|x\|, \quad A \in L(\mathbb{R}^n).$$

Let $a_i \in \mathbb{R}^n$ $(i=1,\ldots,n)$. Then $(a_1,a_2,\ldots,a_n)$ denotes the $n\times n$ matrix with columns $a_i\,(i=1,\ldots,n)$. The following result is valid.

<u>1.1. LEMMA</u>. *Let* $A = (a_1,\ldots,a_n) \in L(\mathbb{R}^n)$. *Then, for* $i = 1,\ldots,n$,

$$\|a_i\| \le \|A\| \le \|A\|_F,$$

where $\|\cdot\|_F$ *denotes the <u>Frobenius norm</u>:*

$$\|A\|_F = \left(\sum_{j=1}^{n} \|a_j\|^2\right)^{\frac{1}{2}}.$$

<u>PROOF</u>. See WILKINSON [1965, section 52 to 54]. □

For $x \in \mathbb{R}^n$ and any real number $\delta > 0$, $U(x,\delta)$ denotes the *open* δ*-neighbourhood of* x:

(1.3) $\quad U(x,\delta) = \{y \mid y \in \mathbb{R}^n, \|x-y\| < \delta\}$.

Let U be some subset of $\mathbb{R}^n$, then $\bar{U}$ denotes its *closure* in $\mathbb{R}^n$ and int(U) its *interior*.

<u>1.2. DEFINITION</u>. Let U be a subset of $\mathbb{R}^n$. Then U is *path-connected* if for any $x,y \in U$ there consists a continuous mapping $p : [0,1] \to U$ such that $p(0) = x$, $p(1) = y$.

<u>1.3. NOTATION</u>. D is a nonempty open subset of $\mathbb{R}^n$ and F a function with domain D and range in $\mathbb{R}^n$:

$$F : D \to \mathbb{R}^n,$$

where n is said to be the *order* of F. Moreover, if the Fréchet-derivative $F'(x)$ exists at $x \in D$, then $F'(x)$ is a linear operator from $\mathbb{R}^n$ to $\mathbb{R}^n$, which can be represented by an $n \times n$ matrix. This matrix is called the *jacobian (matrix)* of F at x and is denoted by $J(x)$. ($J(x)$ equals the matrix of partial derivatives of F at x.)

In the sequel notation 1.3 is used without further comments.

<u>1.4. DEFINITION</u>. Let $A \in L(\mathbb{R}^n)$ be nonsingular and $x \in D$. Define the set $U \subset D$ by

$$U = \{y \mid y \in D, \|AF(y)\| \leq \|AF(x)\|\}.$$

Then, the *levelset of* F with respect to A and x, denoted by $S_F(x,A)$, is defined to be that path-connected component of U which contains x.

We shall now give some standard conditions which appear to be useful in the sequel.

We say that F satisfies
<u>1.5. CONDITION</u> if the Fréchet-derivative $F'(x)$ of F at x exists and is continuous for all $x \in D$.

Let $x \in D$ and $U \subset D$ with $x \in U$ be given. Then F and x satisfy on U:
<u>1.6 CONDITION</u> if condition 1.5 is satisfied and there is a constant $\gamma = \gamma(x) \geq 0$ such that for all $y \in U$:

$$\|J(y) - J(x)\| \le \gamma(x)\|y-x\|;$$

1.7. CONDITION if condition 1.5 is satisfied, $J(x)$ is nonsingular and there exists a constant $\omega = \omega(x) \ge 0$ such that for all $y \in U$:

$$\|(J(x))^{-1}J(y) - I\| \le \omega(x)\|y-x\|.$$

Let $x \in D$, $A \in L(\mathbb{R}^n)$. Then F, x and A satisfy
1.8. CONDITION if A is nonsingular, F satisfies condition 1.5, $J(z)$ is nonsingular for all $z \in S_F(x,A)$ and $S_F(x,A)$ is compact;
1.9. CONDITION if condition 1.8 is satisfied and there exists a $\bar{\gamma} \ge 0$ such that, for all $z \in S_F(x,A)$, F and z satisfy condition 1.6 on $S_F(x,A)$ with $\gamma(z) \le \bar{\gamma}$;
1.10. CONDITION if condition 1.8 is satisfied and there exists a $\bar{\omega} \ge 0$ such that, for all $z \in S_F(x,A)$, F and z satisfy condition 1.7 on $S_F(x,A)$ with $\omega(z) \le \bar{\omega}$.

Conditions 1.7 and 1.10 are so-called affine invariant analoga of conditions 1.6 and 1.9, respectively. This means that, if F in conditions 1.7 or 1.10 is affinely transformed yielding $\tilde{F} = TF$, for any nonsingular $T \in L(\mathbb{R}^n)$, then these conditions remain unchanged. This is easily shown by the observation

$$\|(\tilde{J}(x))^{-1}\tilde{J}(y) - I\| = \|(J(x))^{-1}J(y) - I\|$$

where $\tilde{J}$ denotes the jacobian of $\tilde{F}$. In particular, the constant $\omega(x)$ (and $\bar{\omega}$) is independent of affine transformation of F. This is not true for $\gamma(x)$ (and $\bar{\gamma}$) in condition 1.6 (and 1.9). The justification of conditions 1.9 and 1.10 lie in their use in the convergence theorems in chapter 5.

As we are concerned with the numerical solution of systems of nonlinear equations, we are confronted with round-off errors during computation of function values. In order to be able to deal with this we use the following definitions.

1.11. DEFINITION. Let be given a function $f : D \to \mathbb{R}$, with $D \subset \mathbb{R}$, and a real number $\delta > 0$. Then f is called δ-*monotonous* on a certain interval (a,b) if $\delta < b-a$ and either

(1.4) $\quad f(t+\delta) \geq f(t) \quad$ for all $t \in (a,b-\delta)$,

or

(1.5) $\quad f(t+\delta) \leq f(t) \quad$ for all $t \in (a,b-\delta)$.

If (1.4) is satisfied then f is *δ-monotone increasing*.
If (1.5) is satisfied then f is *δ-monotone decreasing*.

1.12. DEFINITION. Let $\tilde{F}(x)$ denote an approximation to $F(x)$ for all $x \in D$. Let $\Delta(x) \geq 0$ be a given real number for all $x \in D$. Then $\tilde{F}$ is a *Δ-unimodal approximation* to F if for all $x \in D$ and all $d \in \mathbb{R}^n$ with $\|d\| = 1$ the following statement holds:
whenever τ_1 and τ_2 are real numbers with $\tau_2 > \tau_1+\Delta(x)$ and $x+td \in D$ for $t \in (\tau_1,\tau_2)$, such that $\|F(x+td)\|$ is monotone increasing (decreasing) for $t \in (\tau_1,\tau_2)$, then $\|\tilde{F}(x+td)\|$ is $\Delta(x)$-monotone increasing (decreasing) on $[\tau_1,\tau_2]$.

The following lemmas are well known results. For proofs of these lemmas see ORTEGA & RHEINBOLDT [1970, sections 2.3 and 3.2].

1.13. LEMMA (Perturbation lemma). *Let $A \in L(\mathbb{R}^n)$. Then A^{-1} exists if and only if there exists a $B \in L(\mathbb{R}^n)$ such that B^{-1} exists and*

$$\|B-A\| < 1 / \|B^{-1}\|.$$

Moreover, if A^{-1} exists then

$$A^{-1} = \sum_{i=0}^{\infty} (I - B^{-1}A)^i B^{-1}, \tag{1.6}$$

$$\|A^{-1}\| \leq \frac{\|B^{-1}\|}{1-\|B^{-1}\| \, \|B-A\|}. \tag{1.7}$$

1.14. LEMMA. *Let F satisfy condition 1.5 and let D_0 be a convex subset of D. Then, for any $x,y \in D_0$,*

$$\|F(y) - F(x)\| \leq \sup_{0<t<1} \|J(x+t(y-x))\| \, \|y-x\|.$$

An analogous result also holds for functions $F : D \subset \mathbb{R}^n \to \mathbb{R}^m$ with $n \neq m$.

1.15. LEMMA. *Let* D_0 *be a convex subset of* D *and* $x \in D_0$. *Suppose* F *and* x *satisfy condition* 1.6 *on* D_0. *Then*

$$\|F(y)-F(x)-J(x)(y-x)\| \leq \tfrac{1}{2}\gamma(x)\|y-x\|^2,$$

for all $y \in D_0$. *Moreover, an analogous result also holds for functions* $F : D \subset \mathbb{R}^n \to \mathbb{R}^m$ *with* $n \neq m$.

Finally we prove a lemma, which is the affine invariant analogon of lemma 1.15.

1.16. LEMMA. *Let* D_0 *be a convex subset of* D *and* $x \in D_0$. *Suppose* F *and* x *satisfy condition* 1.7 *on* D_0. *Then*

$$\|(J(x))^{-1}(F(y)-F(x)-J(x)(y-x))\| \leq \tfrac{1}{2}\omega(x)\|y-x\|^2,$$

for all $y \in D_0$.

PROOF. Define, for all $z \in D$:

$$\tilde{F}(z) = (J(x))^{-1}F(z).$$

Then, for $z = x$, we have for the jacobian $\tilde{J}(x)$ of $\tilde{F}$: $\tilde{J}(x) = I$. So, application of lemma 1.15 yields the required result. □

1.2. ITERATIVE PROCESSES

We use definitions which are close to those of ORTEGA & RHEINBOLDT [1970].

1.17. DEFINITION. Let $C^{(k)} \subset \mathbb{R}^m$ (k=0,1,...) for certain $m > 0$. Let a sequence of operators $\{\Psi^{(k)}\}_{k=0}^{\infty}$ be given such that

$$\Psi^{(k)} : C^{(k)} \to \mathbb{R}^m, \quad k = 0,1,\dots .$$

Let $C^* \subset C^{(0)}$ be the set of all $z_0 \in C^{(0)}$ such that a sequence $\{z_k\}_{k=0}^{\infty}$ exists generated by

$$(1.8) \qquad z_{k+1} = \Psi^{(k)}(z_k), \quad k = 0,1,\dots .$$

Then C^* is the *domain* of the *iterative process* (1.8). The iterative process (1.8) is defined by the sequence of *iteration functions* $\{\Psi^{(k)}\}_{k=0}^{\infty}$. m is called the *dimension* of the process. We say that the process *converges* for a given *starting point* $z_0 \in C^*$ if there exists a $z^* \in \mathbb{R}^m$, called a *limit* of the process, such that $\lim_{k\to\infty} z_k = z^*$, where z_k is generated by (1.8) with z_0 as given. The process is called a *stationary iterative process* defined by iteration function Ψ if $\Psi^{(k)} = \Psi$ (k=0,1,...).

Using the terminology of ORTEGA & RHEINBOLDT [1970] definition 1.17 defines a sequential 1-step iterative process. They give a more general definition of an iterative process defined by iteration functions that may depend on several preceding iterates in any non-specified order. As we shall not consider such general processes in this thesis we restrict ourselves to definition 1.17.

For short we denote some sequence $\{w_k\}_{k=0}^{\infty}$ by $\{w_k\}$ and $\{w_k\} \subset S$ means that $w_k \in S$ for all $k = 0,1,\dots$.

Let F be given. Then the problem we are concerned with is to obtain a solution of the equation $F(x) = 0$. Hence, we want to construct an iterative process (depending on F) such that for an arbitrary starting point the process converges to a limit which provides us a solution of $F(x) = 0$. Therefore, we look for a method to construct an iterative process for arbitrary

F. This leads us to the concept of iterative method. In the following definition we restrict attention to functions F satisfying condition 1.5.

1.18. DEFINITION. Define

$$\mathcal{F} = \{F \mid F \text{ satisfies condition } 1.5\}$$

and for arbitrary positive integer m

$$\mathcal{P}(m) = \{\Psi \mid \Psi : C_\Psi \to \mathbb{R}^m,\ C_\Psi \subset \mathbb{R}^m,\ C_\Psi \neq \emptyset\}.$$

Let $\{\mathcal{M}^{(k)}\}$ be a sequence of mappings:

$$\mathcal{M}^{(k)} : \mathcal{D}_k \to \mathcal{P}(m),\quad \mathcal{D}_k \subset \mathcal{F},\quad k = 0,1,2,\ldots ,$$

Suppose $\bigcap_{k=1}^{\infty} \mathcal{D}_k \neq \emptyset$, then $\{\mathcal{M}^{(k)}\}$ is called an *iterative method*.
If $\mathcal{M}^{(k)} = \mathcal{M}$ $(k=0,1,2,\ldots)$ then we say that $\mathcal{M}$ is a *stationary iterative method*.

Note that, for a given $F \in \mathcal{F}$, the iterative method $\{\mathcal{M}^{(k)}\}$ gives a sequence of iteration functions $\{\mathcal{M}^{(k)}(F)\}$ which defines an iterative process.

1.19. REMARK. The definition of iterative methods is such that, for given F of order n, the dimension m of the resulting iterative process is not necessarily equal to n. This is essential for our purposes.
For instance, the iterates may be of the form $z_k = (x_k, H_k) \in \mathbb{R}^{n+n^2}$, with $x_k \in D$, $H_k \in L(\mathbb{R}^n)$ and $\mathbb{R}^n \times L(\mathbb{R}^n)$ identified with $\mathbb{R}^{n+n^2}$. Here the sequence $\{x_k\}$ may be interpreted as a sequence of approximations to a solution and $\{H_k\}$ may be interpreted as a sequence of approximations to the jacobian at x_k $(k=0,1,\ldots)$. So in this case $m > n$. We can also give an example with $m < n$. Suppose, for instance, that it is known that the solution x^* has the form $x^* = x_0 + Vy^*$, with $x_0 \in D$ known, V a known operator $V : \mathbb{R}^m \to \mathbb{R}^n$ and $y^* \in \mathbb{R}^m$ to be determined. Then we may construct an iterative process with iterates in $\mathbb{R}^m$, which for a given starting point converges to y^*.

1.20. DEFINITION. Let $\{z_k\}$ be a sequence of elements in $\mathbb{R}^m$. Suppose there exists a $z^* \in \mathbb{R}^m$ such that $\lim_{k\to\infty} z_k = z^*$ and $z_k = z^*$ for at most finitely many indices k. Let $k_0, k_1, k_2, \ldots$ be the sequence of indices obtained from $0,1,2,\ldots$ by omitting those for which $z_k = z^*$ and suppose

$$(1.9) \qquad \liminf_{j\to\infty} (-\log\|z_{k_j}-z^*\|)^{1/k_j} = \rho.$$

Then, we say that $\{z_k\}$ *converges* to z^* with *order of convergence* ρ. If there is no finite ρ satisfying (1.9) then we say that $\{z_k\}$ converges to z^* with order of convergence ∞. Let

$$c = \limsup_{i\to\infty} \|z_i-z^*\|^{1/i},$$

then $0 \le c \le 1$. We say that convergence is *linear* if $0 < c < 1$, *sublinear* if $c = 1$ and *superlinear* if $c = 0$.

In the literature one sometimes uses the term "weak order of convergence" where we have used the "order of convergence" (see van de GRIEND [1978]). Other definitions of order of convergence are possible which may lead to different results (see also ORTEGA & RHEINBOLDT [1970, chapter 9]). For our purposes definition 1.20 suffices.

The following definition and lemmas appear to be useful in proving convergence of sequences of vectors

<u>1.21. DEFINITION</u>. Let $\{z_k\}$ be a sequence in $\mathbb{R}^m$. Then a sequence $\{t_k\} \subset [0,\infty)$ is called a *majorizing sequence* for $\{z_k\}$, if

$$\|z_{k+1}-z_k\| \le t_{k+1}-t_k \qquad (k=0,1,\dots).$$

<u>1.22. LEMMA</u>. *Let $\{t_k\} \subset [0,\infty)$ be a majorizing sequence for $\{z_k\} \subset \mathbb{R}^m$. Suppose that $t^* < \infty$ exists such that $\lim_{k\to\infty} t_k = t^*$. Then there exists a* $z^* \in \mathbb{R}^m$ such that

$$\lim_{k\to\infty} z_k = z^*, \qquad \|z^*-z_k\| \le t^*-t_k \qquad (k=0,1,\dots).$$

<u>PROOF</u>. See ORTEGA & RHEINBOLDT [1970, sect. 12.4.2]. □

<u>1.23. LEMMA</u>. *Let $\eta \ge 0$ be some constant and let p_i $(i=1,2,3,4)$ be constants satisfying $p_i \ge 0$ $(i=1,2,3,4)$, $p_1 > 0$, $p_2 < 1$, $p_3+p_4 = 2p_1$ and $0 \le \eta \le (1-p_2)^2/(4p_1)$. Then we can define a sequence $\{t_k\}$ by $t_0 = 0$, $t_1 = \eta$ and the difference equation*

$$(1.10) \qquad t_{k+1}-t_k = \frac{1}{1-p_4t_k}(p_1(t_k-t_{k-1})+p_2+p_3t_{k-1})(t_k-t_{k-1}) \qquad (k=1,2,\dots).$$

If $\eta \neq 0$ *then* $\{t_k\}$ *is increasing and*

$$(1.11) \qquad \lim_{k\to\infty} t_k = \frac{1}{2p_1}\left((1-p_2) - \sqrt{(1-p_2)^2-4p_1\eta}\right)$$

PROOF. See ORTEGA & RHEINBOLDT [1970, section 12.6.3]. □

Finally, we give a theorem about existence and uniqueness of fixed points of iteration functions. This theorem is essentially based on Kantorovich lemma (see ORTEGA & RHEINBOLDT [1970, lemma 12.5.3].

1.24. THEOREM. *Let* $\Psi : D_\Psi \to \mathbb{R}^m$, $D_\Psi \subset \mathbb{R}^m$, *be an iteration function which is differentiable on a convex set* $D_0 \subset D_\Psi$. *Assume that for some constant* $\gamma > 0$

$$\|\Psi'(z_1) - \Psi'(z_2)\| \leq \gamma\|z_1-z_2\|, \quad \text{for all } z_1,z_2 \in D_0$$

and that there is a $z_0 \in D_0$ *such that* $\|\Psi'(z_0)\| \leq \sigma < 1$. *Suppose* $\alpha = \gamma\beta/(1-\sigma)^2 \leq \frac{1}{2}$, *where* $\beta = \|z_0-\Psi(z_0)\|$. *Set*

$$t^* = ((1-\delta)/\gamma)(1-\sqrt{1-2\alpha}), \quad t^{**} = ((1-\delta)/\gamma)(1+\sqrt{1-2\alpha})$$

and assume $\overline{U(z_0,t^*)} \subset D_0$. *Then the sequence* $\{z_k\}$ *generated by the iterative process with starting point* z_0 *remains in* $\overline{U(z_0,t^*)}$ *and converges to a fixed point* z^* *of* Ψ *which is unique in* $D_0 \cap U(z_0,t^{**})$.

PROOF. See ORTEGA & RHEINBOLDT [1970, theorem 12.5.5]. □

1.3. NUMERICAL ALGEBRA

1.3.1. Triangular decomposition of matrices and solution of systems of linear equations

Let A be an arbitrary nonsingular $n\times n$ matrix. Then, by the process of triangularization (see e.g. WILKINSON [1965, sect. 4.15]), we can obtain an $n\times n$ lower-triangular matrix L and an $n\times n$ upper-triangular matrix R such that

(1.12) $$P_1AP_2 = LR,$$

where one can choose either L or R unit-triangular, i.e. with diagonal elements equal to 1, and where P_1 and P_2 are some permutation matrices. These permutation matrices are induced by some strategy for monitoring the stability of numerical computation, called pivoting.
The number of basic arithmetical operations (+ and $\times$) required to perform such a triangular decomposition is $\frac{1}{3}n^3 + o(n^2)$. In order to solve the linear system

$$Ax = b,$$

with $b \in \mathbb{R}^n$, one solves subsequently

$$Ly = P_1b,\ Rz = y \text{ and } x = P_2z,$$

i.e. *forward substitution* and *backward substitution* respectively, followed by a permutation. This requires only $o(n^2)$ additional basic operations to obtain x.
We like to point out that calculation of A^{-1}, which is done usually by calculating the triangular decomposition first, will require $\frac{2}{3}n^3 + o(n^2)$ basic operations in addition to those required for the triangular decomposition. Therefore, if A^{-1} is not explicitly needed, one should calculate the solution of a linear system $Ax = b$ by computing the triangular decomposition and, subsequently, performing forward and backward substitution. In order to avoid ambiguity we write

(1.13) $$x = A\backslash b$$

for the vector x, the solution of $Ax = b$, which is obtained in this way. If the inverse is calculated explicitly and then multiplied by the right hand side b to obtain the solution, then we write

$$x = A^{-1}b.$$

Note that mathematically, but not numerically, the identity $A\backslash b = A^{-1}b$ holds. So the notation (1.13) will be used only if algorithms are described or numerical aspects are considered.

1.3.2. Generalized inverse and singular value decomposition

Let A be an arbitrary $n\times m$ matrix. Then an $m\times n$ matrix X is said to be the *generalized inverse* of A, if the following identities hold:

$$\text{(1.14)} \qquad \begin{array}{llll} AXA & = A \;, & XAX & = X, \\ (AX)^T & = AX \;, & (XA)^T & = XA. \end{array}$$

The generalized inverse of A is uniquely determined by (1.14) (see PENROSE [1955]). We denote the generalized inverse of A by A^+. In the literature the term *pseudo-inverse* is sometimes used to denote the generalized inverse. We can calculate the generalized inverse A^+ of A by using the so-called *singular value decomposition* of A (see GOLUB & KAHAN [1965]). For all $n\times m$ matrices A there exists a decomposition

$$\text{(1.15)} \qquad A = U\,\Sigma\,V^T,$$

where U is an $n\times n$ orthonormal matrix, V is an $m\times m$ orthonormal matrix and Σ is an $n\times m$ diagonal matrix : $\Sigma = \mathrm{diag}(\sigma_1,\ldots,\sigma_k)$, with $k = \min(n,m)$ and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k \geq 0$. The values σ_i $(i=1,\ldots,k)$ are called the *singular values* of A. The generalized inverse of Σ is given by the $m\times n$ diagonal matrix

$$\Sigma^+ = \mathrm{diag}(\sigma_1^+,\ldots,\sigma_k^+),$$

where

$$\sigma_i^+ = \begin{cases} 1/\sigma_i, & \text{if } \sigma_i \neq 0 \\ 0 \;, & \text{if } \sigma_i = 0 \end{cases} \qquad (i=1,\ldots,k).$$

We obtain the generalized inverse A^+ of A by

$$(1.16) \qquad A^+ = V\,\Sigma^+\,U^T.$$

The *rank of* A, denoted by rank(A), is defined as the number of nonzero singular values of A.

Let us split the matrices U and V such that

$$(1.17) \qquad U = (U_1 \mid U_2), \quad V = (V_1 \mid V_2),$$

where U_1 and V_1 consist of the first r = rank(A) columns of U and V respectively, corresponding to the nonzero singular values of A. Furthermore, write Σ_r for the r×r diagonal matrix $\Sigma_r = \mathrm{diag}(\sigma_1,\ldots,\sigma_r)$. Then

$$(1.18) \qquad A = U_1\Sigma_r V_1^T, \quad A^+ = V_1\Sigma_r^+ U_1^T$$

and consequently

$$(1.19) \qquad AA^+ = U_1U_1^T, \quad A^+A = V_1V_1^T.$$

For the n×n matrix $P = U_1U_1^T$ we have $P^2 = P$ and $[x-Px,Py] = 0$ for all $x,y \in \mathbb{R}^n$, since $U_1^TU_1 = I_r$ (identity in $\mathbb{R}^r$). Therefore, P is an orthogonal projector in $\mathbb{R}^n$, projecting on the subspace in $\mathbb{R}^n$ spanned by the columns of U_1. Similarly $V_1V_1^T$ is an orthogonal projector in $\mathbb{R}^m$, projecting on the subspace of $\mathbb{R}^m$ spanned by the columns of V_1. Before stating a lemma based on these observations we give some notational conventions.

<u>1.25. NOTATION</u>. Let A be an arbitrary n×m matrix. Then we denote

(i) the *range* of A:

$$\mathrm{range}(A) = \{y \mid y \in \mathbb{R}^n,\ \exists x \in \mathbb{R}^m : Ax = y\},$$

(ii) the *kernel* of A:

$$\ker(A) = \{x \mid x \in \mathbb{R}^m, Ax = 0\},$$

(iii) the *span* of A:

span(A) = the subspace of $\mathbb{R}^n$ spanned by the columns of A.

(iv) if $S \subset \mathbb{R}^n$, then S^c denotes the orthogonal complement in $\mathbb{R}^n$.

1.26. LEMMA. *Let* A *be an arbitrary* $n\times m$ *matrix. Suppose its singular value decomposition is given by* (1.15) *and* U_1, U_2, V_1 *and* V_2 *are given by* (1.17). *Define* $r = \mathrm{rank}(A)$. *Then*

(i) $\mathrm{range}(A) = \mathrm{span}(U_1)$, $AA^+ = U_1U_1^T$ *is an orthogonal projector on* $\mathrm{range}(A)$;

(ii) $(\ker(A))^c = \mathrm{span}(V_1)$; $A^+A = V_1V_1^T$ *is an orthogonal projector on* $(\ker(A))^c$;

(iii) *if* $b \in \mathbb{R}^n$, $b \in \mathrm{range}(A)$, *then the equation* $Ax = b$ *has at least one solution and all solutions can be written in the form*

(1.20) $$x = A^+b + V_2z \quad \textit{for arbitrary } z \in \mathbb{R}^{m-r};$$

(iv) *if* $b \in \mathbb{R}^n$ *then* $\|Ax-b\|$ *is minimal for* $x = A^+b + V_2z$ *for arbitrary* $z \in \mathbb{R}^{m-r}$.

PROOF. Statements (i) and (ii) follow immediately from (1.18), (1.19) and the obervations given after these formulas.
To prove (iii) we observe that (1.19) yields

$$AA^+b = U_1U_1^Tb = b,$$

as $b \in \mathrm{range}(A) = \mathrm{span}(U_1)$ (see (i)). Hence A^+b is a solution of $Ax = b$.
As $\ker(A) = \mathrm{span}(V_2)$ (by (ii)) statement (iii) follows easily.
To prove (iv) consider $\|Ax-b\|^2$ and write $y = V_1^Tx$. Note that $y \in \mathbb{R}^r$ and

$$\|Ax-b\|^2 = \|U_1\Sigma_ry-b\|^2.$$

Differentiation with respect to y and equating to zero yields

(1.21) $$2[U_1\Sigma_r, U_1\Sigma_ry-b] = 0.$$

It is easily seen that $y = V_1^T(A^+b+V_2z) = \Sigma_r^+U_1^Tb$ satisfies (1.21).
As the second derivative with respect to y equals $[U_1\Sigma_r, U_1\Sigma_r]$, which is a positive definite $r\times r$ matrix, we see that $y = V_1^T(A^+b+V_2z)$ yields a minimum.
This proves statement (iv). $\square$

To calculate $x = A^+b$ according to (1.18) for some vector $b \in \mathbb{R}^n$, we successively multiply b by U_1^T, Σ_r^+ and V_1. This requires $(n+m+1)r$ basic arithmetical operations. This is much more efficient than first computing A^+ explicitly, which requires multiplication of an $m\times r$ and an $r\times n$ matrix ($m\times n\times r$ basic operations). In the sequel we always assume that A^+b is calculated in the economical way, as sketched above. As we do not require A^+ explicitly, we do not introduce different notations for different ways of computing, as we did in section 1.3.1 for the solution of a linear system by means of triangular decomposition followed by backward and forward substitution.

In the next two subsections we discuss applications of the singular value decomposition to the problem of solving systems of nonlinear equations.

1.3.3. Reduction of problems with linear components

When the problem $F(x) = 0$ can be decomposed as

$$(1.22)\qquad \begin{pmatrix} Ax+b \\ \\ \bar{F}(x) \end{pmatrix} = 0,$$

where we have a linear part with p linear equations say ($p < n$), and a nonlinear part for some $\bar{F} : D \rightarrow \mathbb{R}^{n-p}$, then we may reduce the n-th order nonlinear problem of solving $F(x) = 0$ to an (n-p)-th order nonlinear problem by first solving the p-th order linear problem explicitly.

1.27. THEOREM. *Let* A *be a* $p\times n$ *matrix* ($p < n$) *and* $b \in \mathbb{R}^p$, *both independent of* x. *Suppose* $\operatorname{rank}(A) = p$. *Let be given a function* $\bar{F} : D \rightarrow \mathbb{R}^{n-p}$. *Let the singular value decomposition of* A *be given by* (1.15) *and* U *and* V *split according to* (1.17). *Define a function* $G : D_1 \rightarrow \mathbb{R}^{n-p}$ *with* $D_1 = \{z \mid z \in \mathbb{R}^{n-p},\ A^+b+V_2z \in D\}$ *by*

$$(1.23)\qquad G(z) = \bar{F}(A^+b+V_2z).$$

Consider the problem

$$(1.24)\qquad G(z) = 0, \quad x = A^+b + V_2z.$$

Then x *is a solution of* (1.22) *if and only if* x *is a solution of* (1.24).

PROOF. First let, for given $x \in D$, (1.22) be satisfied. By lemma 1.26 (iii) x can be written as

$$x = A^+b + V_2 z, \qquad \text{for some } z \in \mathbb{R}^{n-p}.$$

Substitution in (1.22) yields

(1.25) $\quad \bar{F}(A^+b+V_2z) = 0.$

Hence, there exists a $z \in D_1$ such that $G(z) = 0$. This proves the first part of the theorem. Now suppose (1.24) is satisfied. Then, again (1.25) holds and by lemma 1.26 (iii) $A^+b + V_2z$ is a solution of $Ax = b$. So, (1.22) is satisfied. □

<u>1.28. REMARK</u>. Let F satisfy condition 1.5 and suppose $F(x) = 0$ is equivalent to (1.22). Then $\bar{F}'(x)$ exists for all x in D. Moreover

(1.26) $\quad G'(z) = \bar{F}'(A^+b+V_2z)V_2.$

So, from theorem 1.27 we see that, instead of solving the system of n equations (1.22) as a system of n nonlinear equations, we can calculate A^+b and solve the system of n-p nonlinear equations $G(z) = 0$.

1.3.4. <u>Projection and a singular jacobian</u>

We assume that F satisfies condition 1.5. Suppose that for some $x_0 \in D$, $J(x_0)$ is singular and its singular value decomposition is given by

(1.27) $\quad J(x_0) = U_1\Sigma_r V_1^T,$

where $r = \text{rank}(J(x_0))$ and (1.18) is used rather than (1.15).
Consider the function $f(x) = \|F(x)\|^2$. Then the derivative (gradient) of f at $x = x_0$ is given by

$$f'(x_0) = 2J^T(x_0)F(x_0).$$

Substituting (1.27) yields

$$f'(x_0) = 2V_1\Sigma_r U_1^T F(x_0).$$

So $f'(x_0) \in \text{span}(V_1)$. Moreover

(1.28) $\quad f'(x_0) = 0 \iff U_1^T F(x_0) = 0 \iff F(x_0) \in (\mathrm{span}(U_1))^c$.

In fact, only the projection of $F(x_0)$ on $\mathrm{span}(U_1)$ contributes to the gradient as $U_1^T F(x_0) = U_1^T(U_1U_1^T)F(x_0)$ (see lemma 1.26 (i)). We are interested in calculating a solution of the equation $F(x) = 0$, i.e. in calculating a zero-minimum of $f(x)$. A necessary condition for a minimum of $f(x)$ is $U_1^T F(x_0) = 0$. Moreover, if we search for a new approximation to a solution of $F(x) = 0$ with a technique based on exploiting only the gradient of $f(x)$ at $x = x_0$, then a new point will be found in $x_0 + \mathrm{span}(V_1) = \{x \mid x \in \mathbb{R}^n,\ x = x_0 + v,\ v \in \mathrm{span}(V_1)\}$. That means that, with such methods we naturally restrict ourselves to searching a zero of a function $\tilde{F}$ with domain in $\mathrm{span}(V_1)$ and range in $\mathrm{span}(U_1)$ defined by

$$\tilde{F}(v) = U_1^T F(x_0+v), \qquad \text{for } v \in \{y \mid y \in \mathrm{span}(V_1),\ x_0+y \in D\}.$$

Choosing the columns of V_1 as a basis for $\mathrm{span}(V_1)$, we can define a function $G : D_1 \to \mathbb{R}^r$ by

(1.29) $\quad G(z) = U_1^T\, F(x_0+V_1z)$,

for all $z \in D_1 = \{y \mid y \in \mathbb{R}^r,\ x_0+V_1y \in D\}$. G is called the *projected function* of F with respect to x_0. Note that

(1.30) $\quad G'(z) = U_1^T J(x_0+V_1z)V_1$

and

(1.31) $\quad G'(0) = \Sigma_r$.

Furthermore, if z^* is a solution of $G(z) = 0$, then, in general, $x_0+V_1z^*$ is not a solution of $F(x) = 0$, as all zeroes of F may lie outside the set $x_0+\mathrm{span}(V_1)$.

1.29. REMARK. Let $F(x_0) \neq 0$ and let the projected function G of F with respect to x_0, given by (1.29), satisfy $G(0) = 0$. Then, $F(x_0) \in (\mathrm{span}(U_1))^c$ and $f'(x_0) = 0$ (see (1.28)). Hence x_0 is a stationary point of $f(x)$. In this case, an analysis of the eigenvalues and eigenvectors of $f''(x)$ is necessary to obtain the answer to the question whether this stationary point is a local maximum or minimum, or a saddle point. If some eigenvalues are equal to zero this analysis is not sufficient to answer this question. In such a case analysis of the third or higher order derivatives is necessary.

1.3.5. Scaling of matrices

In this subsection we assume that $A \in L(\mathbb{R}^n)$ is nonsingular. Consider the linear system $Ax = b$, for some $b \in \mathbb{R}^n$. Then, the solution $x = A\backslash b$ is well defined and it is well known that the error in the computed solution due to round-off during the computational process, depends on the condition number of the matrix defined by

$$(1.32) \qquad \kappa(A) = \|A\| \, \|A^{-1}\|.$$

In fact, we may expect that this error is small if $\kappa(A)$ is small (about 1) and large if $\kappa(A)$ is large (see WILKINSON [1965, chapter 4]). One reason that $\kappa(A)$ may be large is that rows or columns of A are badly scaled. In order to remove as much as possible the negative effect of bad scaling of a matrix on its condition number, one often uses scaling by diagonal matrices. The following example shows that premultiplying and/or postmultiplying a matrix by diagonal matrices may improve its condition number considerably.

1.30. EXAMPLE. Let for some constant c, $0 < c << 1$:

$$A = \begin{pmatrix} 1 & 0 \\ c & c^2 \end{pmatrix}$$

then

$$\kappa(A) = c^{-2}, \text{ approximately.}$$

Premultiplying A with $D_1 = \text{diag}(1,1/c)$ yields

$$D_1 A = \begin{pmatrix} 1 & 0 \\ 1 & c \end{pmatrix}$$

and

$$\kappa(D_1 A) = \sqrt{2}/c.$$

Moreover, postmultiplication of $D_1 A$ with $D_2 = \text{diag}(1,1/c)$ yields

$$D_1AD_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

with

$$\kappa(D_1AD_2) = \frac{3+\sqrt{5}}{3-\sqrt{5}} = 6.9, \text{ approximately.}$$

Based on these observations one may solve the linear system $Ax = b$ in the following steps:

1. Find appropriate diagonal matrices D_1 and D_2 such that row and column norms of the matrix D_1AD_2 are roughly equal to 1.
2. Perform triangular decomposition of the scaled matrix (cf. (1.12)):

   $$P_1D_1AD_2P_2 = LR,$$

   with P_1 and P_2 permutation matrices.
3. Perform forward substitution yielding y by:

   $$Ly = P_1D_1b.$$

4. Perform backward substitution yielding z by:

   $$Rz = y.$$

5. Repermute and rescale

   $$x = D_2P_2z.$$

Then the computation error depends on $\kappa(D_1AD_2)$ if scaling is exact.

The following results give information how the diagonal matrices D_1 and D_2 might be chosen such that row and column norms of the scaled matrix are roughly equal to 1.

<u>1.31. LEMMA</u>. *Let* $A \in L(\mathbb{R}^n)$ *be arbitrary nonsingular. Define for* $i = 1,2$ $D_i = \mathrm{diag}(d_i^{(1)},\ldots,d_i^{(n)})$ *with*

(1.33) $$d_1^{(i)} = 2^{\mathrm{entier}(-{}^2\log\|A_{i\cdot}\|)}, \qquad i = 1,\ldots,n,$$

(1.34) $$d_2^{(j)} = 2^{\mathrm{entier}(-{}^2\log\|(D_1A)_{\cdot j}\|)}, \qquad j = 1,\ldots,n,$$

where $B_{i\cdot}$ *and* $B_{\cdot j}$ *denotes the* i-*th row and* j-*th column of a matrix* B, *respectively.*

Define

$$(1.35) \qquad \bar{A} = D_1 A D_2 .$$

Then

$$(1.36) \qquad \tfrac{1}{2} \le \|\bar{A}_{\cdot j}\| \le 1, \qquad j = 1,\dots,n,$$

$$(1.37) \qquad \frac{1}{4n} \le \|\bar{A}_{i\cdot}\| \le \sqrt{n}, \quad i = 1,\dots,n.$$

PROOF. From the choice of $d_2^{(j)}$, $j = 1,\dots,n$, we have (1.36). Moreover, the choices of $d_1^{(i)}$ and $d_2^{(i)}$ $(i=1,\dots,n)$ imply

$$(1.38) \qquad \tfrac{1}{2} \le d_1^{(i)} \left(\sum_{j=1}^{n} A_{ij}^2\right)^{\frac{1}{2}} \le 1, \qquad i = 1,\dots,n,$$

$$(1.39) \qquad \tfrac{1}{2} \le d_2^{(j)} \left(\sum_{i=1}^{n} \left(d_1^{(i)} A_{ij}\right)^2\right)^{\frac{1}{2}} \le 1, \quad j = 1,\dots,n.$$

From these inequalities it follows that

$$\left|d_2^{(j)} d_1^{(i)} A_{ij}\right| \le d_2^{(j)} \left(\sum_{i=1}^{n} \left(d_1^{(i)} A_{ij}\right)^2\right)^{\frac{1}{2}} \le 1, \quad i,j = 1,\dots,n.$$

Hence

$$\left(\sum_{j=1}^{n} \left(d_2^{(j)} d_1^{(i)} A_{ij}\right)^2\right)^{\frac{1}{2}} \le \sqrt{n}, \quad i = 1,\dots,n,$$

which proves the right hand inequality of (1.37). From the left hand inequality of (1.38) we see that there exists a k $(1 \le k \le n)$ such that

$$(1.40) \qquad \left|d_1^{(i)} A_{ik}\right| \ge \frac{1}{2\sqrt{n}} .$$

Furthermore, from (1.39):

$$\left(\sum_{i=1}^{n} \left(d_1^{(i)} A_{ij}\right)^2\right)^{\frac{1}{2}} \ge \left(2 d_2^{(j)}\right)^{-1} .$$

Combining this with

$$\left|d_1^{(i)} A_{ij}\right| \le d_1^{(i)} \left(\sum_{j=1}^{n} A_{ij}^2\right)^{\frac{1}{2}} \le 1$$

we obtain

$$d_2^{(j)} \geq \frac{1}{2\sqrt{n}}.$$

So with (1.40)

$$\left(\sum_{j=1}^{n}\left(d_2^{(j)} d_1^{(i)} A_{ij}\right)^2\right)^{\frac{1}{2}} \geq |d_2^{(k)} d_1^{(i)} A_{ik}| \geq \frac{1}{4n} . \quad \square$$

It follows from lemma 1.31 that successive row and column scaling of matrix elements by powers of 2, to assure exact scaling, yields a matrix with row and column norms which are roughly equal to 1. Although we cannot quarantee that $\kappa(\bar{A})$, with $\bar{A}$ given by (1.35), is less than $\kappa(A)$, this is very likely if column norms or row norms of A vary widely (see for instance van der SLUIS [1969]).

1.3.6. Scaling of systems of nonlinear equations

Let F satisfy condition 1.5. Then we may scale the function by premultiplying it by some diagonal matrix D_1. We can also scale the variables with a diagonal matrix D_2, i.e. we choose new variables $\bar{x} = D_2^{-1}x$. So we may consider the function

$$\bar{F}(\bar{x}) = D_1 F(D_2 \bar{x}).$$

Then we obtain for the jacobian $\bar{J}(\bar{x})$ of $\bar{F}$:

$$\bar{J}(\bar{x}) = D_1 J(x) D_2.$$

This suggests choosing of D_1 and D_2 dependent on $J(x)$ for some $x \in D$, such that norms of rows and columns of $D_1 J(x) D_2$ are roughly equal to 1.
In fact, if x_0 is an initial guess to the solution of $F(x) = 0$ and B_0 is an approximation to $J(x_0)$, then scaling of the function and the variables with diagonal matrices D_1 and D_2, satisfying (1.33) and (1.34) with A replaced by B_0 is at hand and will be used (see section 6.10).

where $B_{i\cdot}$ *and* $B_{\cdot j}$ *denotes the* i*-th row and* j*-th column of a matrix* B, *respectively.*

Define

(1.35) $\quad \bar{A} = D_1 A D_2.$

Then

(1.36) $\quad \tfrac{1}{2} \le \|\bar{A}_{\cdot j}\| \le 1, \qquad j = 1,\dots,n,$

(1.37) $\quad \frac{1}{4n} \le \|\bar{A}_{i\cdot}\| \le \sqrt{n}, \quad i = 1,\dots,n.$

PROOF. From the choice of $d_2^{(j)}$, $j = 1,\dots,n$, we have (1.36). Moreover, the choices of $d_1^{(i)}$ and $d_2^{(i)}$ $(i=1,\dots,n)$ imply

(1.38) $\quad \tfrac{1}{2} \le d_1^{(i)} \left(\sum_{j=1}^{n} A_{ij}^2\right)^{\frac{1}{2}} \le 1, \qquad i = 1,\dots,n,$

(1.39) $\quad \tfrac{1}{2} \le d_2^{(j)} \left(\sum_{i=1}^{n} \left(d_1^{(i)} A_{ij}\right)^2\right)^{\frac{1}{2}} \le 1, \quad j = 1,\dots,n.$

From these inequalities it follows that

$$|d_2^{(j)} d_1^{(i)} A_{ij}| \le d_2^{(j)} \left(\sum_{i=1}^{n} \left(d_1^{(i)} A_{ij}\right)^2\right)^{\frac{1}{2}} \le 1, \quad i,j = 1,\dots,n.$$

Hence

$$\left(\sum_{j=1}^{n} \left(d_2^{(j)} d_1^{(i)} A_{ij}\right)^2\right)^{\frac{1}{2}} \le \sqrt{n}, \quad i = 1,\dots,n,$$

which proves the right hand inequality of (1.37). From the left hand inequality of (1.38) we see that there exists a k $(1 \le k \le n)$ such that

(1.40) $\quad |d_1^{(i)} A_{ik}| \ge \frac{1}{2\sqrt{n}}\,.$

Furthermore, from (1.39):

$$\left(\sum_{i=1}^{n} \left(d_1^{(i)} A_{ij}\right)^2\right)^{\frac{1}{2}} \ge \left(2 d_2^{(j)}\right)^{-1}.$$

Combining this with

$$|d_1^{(i)} A_{ij}| \le d_1^{(i)} \left(\sum_{j=1}^{n} A_{ij}^2\right)^{\frac{1}{2}} \le 1$$

we obtain

$$d_2^{(j)} \geq \frac{1}{2\sqrt{n}}.$$

So with (1.40)

$$\left(\sum_{j=1}^{n}\left(d_2^{(j)}d_1^{(i)}A_{ij}\right)^2\right)^{\frac{1}{2}} \geq |d_2^{(k)}d_1^{(i)}A_{ik}| \geq \frac{1}{4n}\ . \quad \square$$

It follows from lemma 1.31 that successive row and column scaling of matrix elements by powers of 2, to assure exact scaling, yields a matrix with row and column norms which are roughly equal to 1. Although we cannot quarantee that $\kappa(\bar{A})$, with $\bar{A}$ given by (1.35), is less than $\kappa(A)$, this is very likely if column norms or row norms of A vary widely (see for instance van der SLUIS [1969]).

1.3.6. Scaling of systems of nonlinear equations

Let F satisfy condition 1.5. Then we may scale the function by premultiplying it by some diagonal matrix D_1. We can also scale the variables with a diagonal matrix D_2, i.e. we choose new variables $\bar{x} = D_2^{-1}x$. So we may consider the function

$$\bar{F}(\bar{x}) = D_1F(D_2\bar{x}).$$

Then we obtain for the jacobian $\bar{J}(\bar{x})$ of $\bar{F}$:

$$\bar{J}(\bar{x}) = D_1J(x)D_2.$$

This suggests choosing of D_1 and D_2 dependent on $J(x)$ for some $x \in D$, such that norms of rows and columns of $D_1J(x)D_2$ are roughly equal to 1.
In fact, if x_0 is an initial guess to the solution of $F(x) = 0$ and B_0 is an approximation to $J(x_0)$, then scaling of the function and the variables with diagonal matrices D_1 and D_2, satisfying (1.33) and (1.34) with A replaced by B_0 is at hand and will be used (see section 6.10).

CHAPTER 2

EXISTENCE AND UNIQUENESS

In this chapter we shall derive conditions for the existence of a path (the so-called *Newton-path*) going from a given starting point $x_0 \in D$ to a solution which lies in $S_F(x_0,A)$ for all nonsingular $A \in L(\mathbb{R}^n)$. This path is independent of A and is contained in $S_F(x_0,A)$. The solution as well as the Newton-path are unique in $S_F(x_0,A)$ under the given conditions.
The results given in this chapter are based on RHEINBOLDT [1969] and ORTEGA & RHEINBOLDT [1970]. Basic to this theory is the inverse function theorem (theorem 2.2).

2.1. DEFINITION. F is called a *local homeomorphism* at $x \in D$, if their exist open neighbourhoods U and V of x and F(x), respectively, such that $U \subset D$ and the restriction of F to U is a homeomorphism of U onto V (i.e. F is a one-to-one mapping from U onto V and F and F^{-1} are continuous on U and V, respectively).

2.2. THEOREM (Inverse function theorem). *Let $x_0 \in D$. Suppose that the Frechet-derivative of F exists at each point of some open neighbourhood of x_0 in D. Suppose that F' is continuous at x_0 and $F'(x_0)$ is nonsingular. Then, F is a local homeomorphism at x_0. Suppose, in addition, that the restriction F_U of F to a certain open neighbourhood U of x_0 is one-to-one, F_U' exists and is continuous on U and $F_U'(x)$ is nonsingular for all $x \in U$. Then $(F_U^{-1})'$ exists and is continuous on an open neighbourhood V of $F(x_0)$, with*

$$(F_U^{-1})'(F(x)) = (J(x))^{-1}, \tag{2.1}$$

for all $x \in U$ such that the argument $F(x) \in V$.

PROOF. See ORTEGA & RHEINBOLDT [1970, section 5.2.1]. □

2.3. DEFINITION. F has the *continuation property* for a given continuous function $q : [0,1] \to \mathbb{R}^n$, if the existence of a continuous function $p : [0,a) \to D$, with $a \in (0,1]$, such that $F(p(t)) = q(t)$ for all $t \in [0,a)$, implies that $\lim_{t \uparrow a} p(t) = p(a)$ exists with $p(a) \in D$ and $F(p(a)) = q(a)$.

2.4. LEMMA. *Let* F *be a local homeomorphism at each point of some open set* $D_0 \subset D$. *If* F *has the continuation property for a continuous function* $q : [0,1] \to \mathbb{R}^n$ *such that* $F(x_0) = q(0)$ *for some* $x_0 \in D_0$, *then there exists a unique continuous function* $p : [0,1] \to D_0$ *which satisfies* $p(0) = x_0$ *and* $F(p(t)) = q(t)$ *for all* $t \in [0,1]$.

PROOF. See ORTEGA & RHEINBOLDT [1970, section 5.3.2]. □

2.5. LEMMA. *Let* F *be a local homeomorphism at each point of some open set* $D_0 \subset D$. Let $q : [0,1] \times [0,1] \to \mathbb{R}^n$ *and* $r : [0,1] \to D_0$ *be continuous functions such that* $F(r(s)) = q(s,0)$ *for all* $s \in [0,1]$. *If, for each fixed* $s \in [0,1]$, F *has the continuation property for* $q_s(t) = q(s,t)$, $t \in [0,1]$, *then there exists a unique continuous mapping* $p : [0,1] \times [0,1] \to D_0$ *such that* $p(s,0) = r(s)$ *and* $F(p(s,t)) = q(s,t)$ *for* $s,t \in [0,1]$. *Moreover, if* $q(s,1) = q(0,t) = q(1,t) = y$ *for all* $s,t \in [0,1]$, *then* $r(0) = r(1)$.

PROOF. See ORTEGA & RHEINBOLDT [1970, section 5.3.4]. □

2.6 LEMMA. *Let* F *be a local homeomorphism at each point of some open set* $D_0 \subset D$. *Let* $p : [0,a) \to D_0$ $(a \in (0,1])$ *be a continuous function. If* $\lim_{t \uparrow a} F(p(t)) = y$ *exists, and if there is a sequence* $\{t_k\} \subset [0,a)$ *with* $\lim_{k \to \infty} t_k = a$ *such that* $\lim_{k \to \infty} p(t_k) = x$ *and* $x \in D_0$, *then* $\lim_{t \uparrow a} p(t) = x$.

PROOF. See ORTEGA & RHEINBOLDT [1970, section 5.3.7]. □

2.7. LEMMA. *Let* $x_0 \in D$ *and* $A \in L(\mathbb{R}^n)$. *Suppose* F, x_0 and A *satisfy condition* 1.8. *Then there is an open set* $D_0 \subset D$ *such that* $S_F(x_0,A) \subset D_0$ *and* $J(x)$ *is nonsingular for all* $x \in D_0$.

PROOF. Suppose $x \in S_F(x_0,A)$. Then $J(x)$ is nonsingular and there exists a constant $\beta > 0$ such that $\|(J(x))^{-1}\| < \beta$. By the continuity of J on D we know that there exists a $\delta > 0$ such that for all $z \in U(x,\delta) \subset D$: $\|J(z) - J(x)\| < 1/\beta$. Then, use of lemma 1.13 yields nonsingularity of $J(z)$ for all $z \in U(x,\delta)$.

Thus, for each point $x \in S_F(x_0,A) \subset D$ there exists an open neighbourhood, U_x say, on which the jacobian is nonsingular. Then $D_1 = \bigcup_{x \in S_F(x_0,A)} U_x$ is an open set containing $S_F(x_0,A)$ on which the jacobian is nonsingular. □

We can now present the main result of this chapter.

<u>2.8. THEOREM</u>. *Let* $x_0 \in D$ *and* $A \in L(\mathbb{R}^n)$. *Suppose* F, x_0 *and* A *satisfy condition* 1.8. *Then, there exists a unique differentiable function* $p : [0,1] \to S_F(x_0,A)$ *satisfying*

(2.2) $$F(p(t)) = (1-t)F(x_0), \quad t \in [0,1].$$

Moreover, p *satisfies*

(2.3) $$\begin{aligned} p'(t) &= -(J(p(t)))^{-1}F(x_0), \quad t \in [0,1] \\ p(0) &= x_0. \end{aligned}$$

Furthermore, $x^* = p(1)$ *is a unique solution of* $F(x) = 0$ *in* $S_F(x_0,A)$. *The path* $\{y \mid y = p(t),\ t \in [0,1]\} \subset S_F(x_0,A)$ *is called the <u>Newton-path</u>.*

<u>PROOF</u>. As a solution of $AF(x) = 0$ is also a solution of $F(x) = 0$ and the conditions are also satisfied with F replaced by AF, without loss of generality we may assume that $A = I$. For simplicity, denote $S = S_F(x_0,A)$. Now let D_0 be an open set, $D_0 \subset D$, such that $S \subset D_0$ and $J(x)$ is nonsingular for all $x \in D_0$ (cf. Lemma 2.7).

By condition 1.8 and theorem 2.2 we conclude that F is a local homeomorphism at each $x \in D_0$

Now consider the continuous function $q(t) = (1-t)F(x_0)$ for $t \in [0,1]$. We shall prove that F has the continuation property for q. Therefore, assume that $p : [0,a) \to D_0$, $a \in (0,1]$, is a continuous function such that $F(p(t)) = q(t)$ for all $t \in [0,a)$. Then $p(t) \in S$ for $t \in [0,a)$ and by theorem 2.2 there exist open neighbourhoods $U \subset D_0$ and $V \subset \mathbb{R}^n$ of $p(t)$ and $q(t)$, respectively, such that F_U (the restriction of F to U) is a homeomorphism of U onto V, $(F_U^{-1})'$ exists and is continuous and

$$(F_U^{-1})'(F(x)) = (J(x))^{-1}, \quad \text{for all } x \in U.$$

Hence, we conclude that p is continuously differentiable on $[0,a)$ with

$$(2.4) \qquad p'(t) = (J(p(t)))^{-1}q'(t) = -(J(p(t)))^{-1}F(x_0).$$

Now let $\{t_k\} \subset [0,a)$ be a monotone increasing sequence converging to a. Then

$$(2.5) \qquad p(t_j)-p(t_k) = \int_{t_k}^{t_j} p'(t)dt = -\int_{t_k}^{t_j} (J(p(t)))^{-1}F(x_0)dt.$$

As $(J(x))^{-1}$ exists and is continuous on the compact set S, there exists a constant β such that

$$\| (J(x))^{-1}F(x_0)\| \leq \beta, \quad \text{for all } x \in S.$$

Hence (2.5) yields

$$\|p(t_j)-p(t_k)\| \leq \beta|t_j-t_k|.$$

Therefore $\{p(t_k)\}$ is a Cauchy-sequence in S. Thus, by the compactness of S we conclude that there is a $z \in S$ such that $\lim_{k\to\infty} p(t_k) = z$. By lemma 2.6 we obtain $\lim_{t\uparrow a} p(t) = z$ and by the continuity of F we have $F(z) = q(a)$. This proves that F has the continuation property for q.

Application of lemma 2.4 with q as above yields existence und uniqueness of a path $p : [0,1] \to D_0$ which satisfies $p(0) = x_0$ and $F(p(t)) = q(t)$ for all $t \in [0,1]$. Clearly $p(t) \in S$ for $t \in [0,1]$ and $F(p(1)) = 0$ by the definition of q. This proves the existence of a solution in S. We proved that existence of a path $p : [0,a) \to D_0$ satisfying (2.2) for all $t \in [0,a)$ implies that p satisfies (2.4) for all $t \in [0,a)$. Moreover, existence of such a path is proved for all $a \in (0,1]$. Therefore, the nonsingularity of $J(p(1))$ and the continuity of p and q on $[0,1]$ yields (2.3).

Finally. we have to prove that $x^* = p(1)$ is a unique solution in S. Suppose $x^*, x^{**} \in S$ and $F(x^*) = F(x^{**}) = 0$. As S is path-connected there exists a continuous function $r : [0,1] \to S$ such that $r(0) = x^*$, $r(1) = x^{**}$.

Define $q : [0,1] \times [0,1] \to \mathbb{R}^n$ by

$$q(s,t) = (1-t)F(r(s)) \quad \text{for all } s,t \in [0,1].$$

Then, for fixed s, F has the continuation property as is proved above. Moreover,

$$q(s,0) = F(r(s)) \quad \text{for all } s \in [0,1]$$

and

$$q(s,1) = q(0,t) = q(1,t) = 0, \quad \text{for all } s,t \in [0,1].$$

Hence, by lemma 2.5, $r(0) = r(1)$, so that $x^* = x^{**}$. This proves the unicity of x^* in S. □

2.9. REMARK. The Newton-path is invariant under affine transformation of the function as follows from (2.2) and (2.3). In fact, DEUFLHARD [1974a] proves, with notations as in theorem 2.8,

$$\{x \mid x = p(t),\ t \in [0,2]\} = \bigcap_{\substack{A \in L(\mathbb{R}^n) \\ A \text{ nonsingular}}} S_F(x_0,A).$$

2.10. REMARK. The compactness of $S_F(x_0,A)$ plays an important role in theorem 2.8. This will be illustrated by two typical examples for which compactness does not hold and where no solution exists in D.

1. $D \subset \mathbb{R}$, $D = (1,2)$; $F(x) = x$, for $x \in D$.
   Then figure 2.1 shows that, for arbitrary $x_0 \in D$, $S_F(x_0,I) = (1,x_0]$, which is not compact. Moreover $F(x) = 0$ has no solution in D.

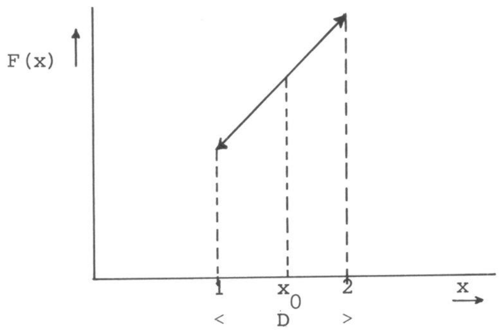


figure 2.1

2. $D = \mathbb{R}$; $F(x) = \arctan(x) + \pi$.
   Then $\pi/2 < F(x) < 3\pi/2$ and $F(x)$ has no solution in D. Choose $x_0 = 1$.
   Then $S_F(x_0,I) = (-\infty,1]$ which is not compact. Note that $J(x) = (x^2+1)^{-1}$,
   which tends to zero for x going to $\pm\infty$. Hence $\|(J(x))^{-1}\|$ is not uniformly
   bounded on $S_F(x_0,I)$.

CHAPTER 3

APPROXIMATING THE JACOBIAN

3.1. INTRODUCTION

In the sequel we shall frequently use approximations to the jacobian matrix or its inverse. Therefore, we present some methods that may be used to obtain such approximations together with results on approximation errors. In this chapter we assume that F satisfies condition 1.5. Furthermore, $B(x)$ shall denote an approximation to $J(x)$ and $H(x)$ an approximation to $(J(x))^{-1}$.

Suppose $x,y \in D$ and $x+t(y-x) \in D$ for all $t \in (0,1)$. Consider the function $g(t) = F(x+t(y-x))$. Then $g'(0)$ can be approximated by using the first divided difference formula:

$$(g(\theta)-g(0)) \;/\; \theta, \qquad \theta \in (0,1].$$

For $\theta = 1$, this yields that $J(x)(y-x)$ is approximately equal to $F(y) - F(x)$. Moreover, equality holds exactly if F is linear.

Motivated by the above reasoning an approximation $B(x)$ to $J(x)$ may be required to satisfy

$$F(y) - F(x) = B(x)(y-x), \tag{3.1}$$

for at least one $y \in D$. We shall discuss two methods for approximating the jacobian based on (3.1), viz. the divided difference approximation and the approximation obtained by updating some approximation to the jacobian at another point. Finally, at the end of this chapter, we present two results when the jacobian is approximated by some fixed matrix.

3.2. DIVIDED DIFFERENCE APPROXIMATION

3.1. DEFINITION. Let $x \in D$ and $P \in L(\mathbb{R}^n)$ be nonsingular such that $x + Pe_i \in D$ $(i=1,\dots,n)$, where e_i is the i-th unit vector in $\mathbb{R}^n$. Define $Q \in L(\mathbb{R}^n)$ by

$$(3.2) \qquad Q(x) = (F(x+Pe_1)-F(x), \;\dots\; ,F(x+Pe_n)-F(x)).$$

Then, $B(x)$ defined by

$$(3.3) \qquad B(x) = Q(x)P^{-1}$$

is called the *divided difference approximation* to $J(x)$ defined by P.

Note that, for given nonsingular P, $B(x)$ is defined uniquely by (3.3) and $B(x)$ satisfies (3.1) for $y = x + Pe_i$ $(i=1,\dots,n)$.

3.2. REMARK. Let $x \in D$ and $A,P \in L(\mathbb{R}^n)$ be nonsingular. Suppose $x+Pe_i \in D$ $(i=1,\dots,n)$. Let $B_F(x)$ and $B_{AF}(x)$ denote the divided difference approximations to the jacobians of F and AF, respectively, defined by P. Then

$$(3.4) \qquad B_{AF}(x) = AB_F(x).$$

3.3. REMARK. Consider the case where P is a diagonal matrix:

$$P = \mathrm{diag}(h_1,\dots,h_n),$$

for $h_i \in \mathbb{R}$, $h_i \neq 0$ $(i=1,\dots,n)$. Then the elements of $B(x)$ can be given explicitly by

$$(3.5) \qquad (B(x))_{ij} = (F_i(x+h_je_j)-F_i(x))/h_j \qquad (i,j=1,\dots,n),$$

where $F(x) = (F_1(x),\dots,F_n(x))^T$.
In the sequel, unless stated otherwise, the term *difference approximation* is used for a divided difference approximation defined by a diagonal matrix P. As the general form (3.3) is rarely used, no confusion can arise from this terminology.

We shall give two theorems on the error bounds for the divided difference approximation. One is based on condition 1.6, the other on the affine invariant condition 1.7.

<u>3.4. THEOREM</u>. *Let* $x \in D$. *Suppose* F *and* x *satisfy condition* 1.6 *on some open neighbourhood* $U \subset D$ *of* x. *For all nonsingular* $P \in L(\mathbb{R}^n)$ *let* $p_i = Pe_i$, $i = 1,\dots,n$, *and*

$$h(P) = \|\mathrm{diag}(\|p_1\|,\dots,\|p_n\|)P^{-1}\|\ \|P\|_F, \tag{3.6}$$

where $\|\ .\ \|_F$ *denotes the Frobenius norm (see lemma* 1.1). *Then, there is a real number* $\delta > 0$ *such that for any nonsingular* $P \in L(\mathbb{R}^n)$ *with* $\|P\| < \delta$, $B(x)$ *can be defined by* (3.3) *and satisfies*

$$\|B(x)-J(x)\| \le \tfrac{1}{2}\gamma(x)h(P). \tag{3.7}$$

Note that $h(P) \to 0$ if $\|P\| \to 0$ and the Frobenius condition number of P, $\|P^{-1}\|_F\ \|P\|_F$, remains bounded, as $h(P) \le (\sum_{i=1}^n \|p_i\|^2)^{\frac{1}{2}}\ \|P^{-1}\|_F\ \|P\|_F$.

<u>PROOF</u>. Choose $\delta > 0$ such that $U(x,\delta) \subset U$. Let $P = (p_1,\dots,p_n) \in L(\mathbb{R}^n)$ be nonsingular, satisfying $\|P\| < \delta$. Then $x + Pe_i = x + p_i \in U(x,\delta)$, for $i = 1,\dots,n$. Hence, $B(x)$ can be defined by (3.3). Now define

$$c_i = F(x+p_i) - F(x) - J(x)p_i \qquad (i=1,\dots,n).$$

Then by lemma 1.15

$$\|c_i\| \le \tfrac{1}{2}\gamma(x)\|p_i\|^2 \qquad (i=1,\dots,n).$$

Hence

$$\|B(x)-J(x)\| = \|(Q(x)-J(x)P)P^{-1}\| = \|(c_1,\dots,c_n)P^{-1}\|$$

$$= \left\|\left(\frac{c_1}{\|p_1\|},\ \dots\ ,\frac{c_n}{\|p_n\|}\right)\mathrm{diag}(\|p_1\|,\ \dots\ ,\|p_n\|)P^{-1}\right\|$$

$$\le \tfrac{1}{2}\gamma(x)\|P\|_F\|\mathrm{diag}(\|p_1\|,\ \dots\ ,\|p_n\|)P^{-1}\|.$$

So that

$$\|B(x) - J(x)\| \le \tfrac{1}{2}\gamma(x)h(P). \quad \square$$

<u>3.5. THEOREM</u>. *Let* $x \in D$. *Suppose* F *and* x *satisfy condition* 1.7 *on some open neighbourhood* $U \subset D$ *of* x. *Then there is a real number* $\delta > 0$ *such that, for all nonsingular* $P \in L(\mathbb{R}^n)$ *with* $\|P\| < \delta$ *and* $h(P) < 2/\omega(x)$ ($h(P)$ *defined by* (3.6)), $B(x)$ *can be defined by* (3.3), *is nonsingular and satisfies*

$$(3.8) \qquad \|(B(x))^{-1}J(x) - I\| \le \frac{\omega(x)h(P)}{2-\omega(x)h(P)}\;.$$

<u>PROOF</u>. Define for $y \in U$

$$\tilde{F}(y) = (J(x))^{-1}F(y).$$

Then $\tilde{F}$ and x satisfy condition 1.6 on U with $\gamma(x)$ replaced by $\tilde{\gamma}(x) = \omega(x)$. By remark 3.2 we have for the divided difference approximation, $\tilde{B}(y)$, to the jacobian of $\tilde{F}$ at y:

$$\tilde{B}(y) = (J(x))^{-1}B(y), \qquad y \in U.$$

Hence application of theorem 3.4 yields the existence of a $\delta > 0$ such that for all nonsingular $P \in L(\mathbb{R}^n)$ with $\|P\| < \delta$, $\tilde{B}(x)$ can be defined by (3.3) and satisfies

$$(3.9) \qquad \|(J(x))^{-1}B(x) - I\| \le \tfrac{1}{2}\omega(x)h(P).$$

As $\tfrac{1}{2}\omega(x)h(P) < 1$ we can use the perturbation lemma (lemma 1.13). Hence $(J(x))^{-1}B(x)$ is nonsingular (so $B(x)$ is nonsingular) and using (1.6) we obtain

$$\|(B(x))^{-1}J(x) - I\| = \|\sum_{i=1}^{\infty}(I - (J(x))^{-1}B(x))^{i}\|$$

$$\le \sum_{i=1}^{\infty}\|I - (J(x))^{-1}B(x)\|^{i}.$$

By (3.9) the result follows. $\square$

For future use we also need a result in case B is a singular approximation to J.

3.6. COROLLARY. *Let the assumptions of theorem* 3.4 *be satisfied. Then, there is a real number* $\delta > 0$ *such that for all nonsingular* $P \in L(\mathbb{R}^n)$ *with* $\|P\| < \delta$, $B(x)$ *can be defined by* (3.3) *and satisfies*

$$(3.10) \qquad \|(J(x) - B(x))(B(x))^+\| \le \tfrac{1}{2}\gamma(x)h(P)\|(B(x))^+\|.$$

PROOF. This easily follows by application of (3.7). □

3.7. REMARK. If P is a diagonal matrix, $P = \mathrm{diag}(h_1,\ldots,h_n)$, then the expression for h(P) given by (3.6) simplifies to

$$(3.11) \qquad h(P) = \|P\|_F = \left(\sum_{j=1}^{n} h_j^2\right)^{\frac{1}{2}}.$$

Up to now, we have not been concerned with the effect of numerical computation of a divided difference approximation to the jacobian. In fact, if inexact computation is used, then round-off errors due to cancellation of significant digits may cause serious difficulties. On this issue we present two theorems associated with theorem 3.5 and corollary 3.6. We restrict attention to diagonal matrices P.

3.8. NOTATION. Consider inexact floating point arithmetic with computational precision ε, called the *machine precision* (as computation is usually done with a machine). With $fl_\varepsilon(\cdot)$ we denote the expression within the brackets computed with machine precision ε.

N.B. With this notation ε can be defined to be the smallest representable number such that $fl_\varepsilon(1+\varepsilon) > 1$ and $fl_\varepsilon(1-\varepsilon) < 1$.

3.9. THEOREM. *Let the assumptions of theorem* 3.5 *be satisfied. Moreover, suppose that there are constants* ε_{rf} *and* ε_{af} *such that*

$$(3.12) \qquad \|fl_\varepsilon(F(x)) - F(x)\| \le \varepsilon_{rf}\|F(x)\| + \varepsilon_{af}, \quad \varepsilon_{rf} \ge \varepsilon,\ \varepsilon_{af} \ge 0.$$

Then, there is a real number $\delta > 0$, *such that for all nonsingular* $P \in L(\mathbb{R}^n)$ *with* $\|P\| < \delta$ *and* $h(P) < 2/\omega(x)$, $B(x)$ *can be defined by* (3.3). *Moreover, if* $P = \mathrm{diag}(h_1,\ldots,h_n)$ *for real numbers* $h_i > 10\varepsilon$ $(i=1,\ldots,n)$ *and* $fl_\varepsilon(B(x))$ *is nonsingular, then*

$$(3.13) \qquad \| (fl_\varepsilon(B(x)))^{-1}J(x) - I\| \le \frac{1}{1-c_1}(c_2+c_1),$$

where

$$\varepsilon_f(x) = (\varepsilon+\varepsilon_{rf})\|F(x)\| + \varepsilon_{af},$$

$$c_1 = \tfrac{1}{2}\omega(x)\left(\sum_{j=1}^{n} h_j^2\right)^{\frac{1}{2}},$$

$$c_2 = 2.12\varepsilon_f(x)\|(fl_\varepsilon(B(x)))^{-1}\|\left(\sum_{j=1}^{n} h_j^{-2}\right)^{\frac{1}{2}}.$$

PROOF. We have

$$(3.14) \qquad \begin{aligned} &\| (fl_\varepsilon(B(x)))^{-1}J(x) - I\| \le \|(B(x))^{-1}J(x) - I\| \\ &+ \|(fl_\varepsilon(B(x)))^{-1}\| \; \|B(x) - fl_\varepsilon(B(x))\| \; \|(B(x))^{-1}J(x)\|. \end{aligned}$$

Using the inequalities (see WILKINSON [1965, section 3.4])

$$|fl_\varepsilon(a\pm b) - (a\pm b)| \le (|a|+|b|)\varepsilon,$$
$$|fl_\varepsilon(a^\times/b) - (a^\times/b)| \le |a\ {}^\times/\ b|\ \varepsilon, \quad \text{for } a,b \in \mathbb{R},$$

and (3.12), some tedious calculations show

$$(3.15) \qquad \|fl_\varepsilon(B(x)) - B(x)\| \le 2 \times 1.06\varepsilon_f(x)\left(\sum_{j=1}^{n} h_j^{-2}\right)^{\frac{1}{2}}.$$

(For typical examples of such calculations see WILKINSON [1965, section 3.1 - 3.9].) Using (3.8) and (3.15) to bound the right hand side of (3.14) yields (3.13). □

3.10. THEOREM. *Let the assumptions of theorem* 3.4 *be satisfied. Suppose there exist constants* ε_{rf} *and* ε_{af} *satisfying* (3.12). *Then, there is a real number* $\delta > 0$, *such that for all nonsingular* $P \subset L(\mathbb{R}^n)$ *with* $\|P\| < \delta$, $B(x)$ *can be defined by* (3.3). *Moreover, if* $P = \text{diag}(h_1,\ldots,h_n)$ *for real numbers* $h_i > 10\varepsilon$ $(i=1,\ldots,n)$ *then*

$$(3.16) \qquad \|(J(x) - fl_\varepsilon(B(x)))(fl_\varepsilon(B(x)))^{+}\| \le c_2^+ + c_1^+,$$

where

$$c_1^+ = \tfrac{1}{2}\gamma(x)\eta^+(x)\left(\sum_{j=1}^{n} h_j^2\right)^{\frac{1}{2}},$$

$$c_2^+ = 2.12\varepsilon_f(x)\ \eta^+(x)\left(\sum_{j=1}^{n} h_j^{-2}\right)^{\frac{1}{2}},$$

$$\eta^+(x) = \|(fl_\varepsilon(B(x)))^+\|.$$

PROOF. We have

$$\|J(x) - fl_\varepsilon(B(x))\| \le \|J(x) - B(x)\| + \|B(x) - fl_\varepsilon(B(x))\|.$$

Using this inequality and bounding the terms on the right hand side by use of (3.7) and (3.15) yields the required result. □

3.11 REMARK. Note that in (3.13) and (3.16), the values of c_1 and c_1^+ decrease and those of c_2 and c_2^+ increase, if $\|P\|$ decreases. So, in practice we have to find for given $x \in D$, diagonal elements h_i $(i=1,\ldots,n)$ of P such that c_1 and c_2 (or c_1^+ and c_2^+) roughly have the same magnitude.

3.3. APPROXIMATION BY UPDATING

The method of approximating $J(x)$ by updating some approximation to $J(y)$ $(x,y \in D)$, such that (3.1) is satisfied, is due to DAVIDON [1959]. He suggested this method for approximating the second derivative of a functional in successive iteration steps of an iterative process for finding an optimum of this functional. Davidon called his method a *variable metric method.* Iterative methods for solving nonlinear systems which make use of Davidon's idea are usually called *quasi-Newton methods* (see DENNIS [1975], DENNIS & MORÉ [1977], BROYDEN [1965, 1969, 1970a,b, 1973]). However the term "quasi-Newton" is confusing since it is also used for other modifications of Newton's method; moreover, the prefix "quasi" is far from clear. Since the term "variable metric" is used only for optimization methods, we prefer the terms "updating methods" (see chapter 4) and "approximation by updating" for the methods described in this thesis.

3.12. DEFINITION. Let F satisfy condition 1.5. Suppose $D_U \subset L(\mathbb{R}^n) \times D \times D$. Then U is a *jacobian update function* for F with domain D_U if

$$U : D_U \to L(\mathbb{R}^n)$$

and

$$U(B,y,x)(x-y) = F(x) - F(y)$$

for all $(B,y,x) \in D_U$.

We restrict ourselves to jacobian update functions of the form

$$U(B,y,x) = B + vu^T,$$

where $v,u \in \mathbb{R}^n$ depend on F, B, y and x. This restriction implies that for arbitrarily chosen u such that $u^T(x-y) \neq 0$, the vector v is determined and given by

$$v = (F(x) - F(y) - B(x-y))/(u^T(x-y)).$$

In fact, we restrict attention to a class of jacobian update functions which can be parametrized by a vector u. The elements, denoted by $\mathcal{U}(u)$, of this class satisfy

$$(3.17)\qquad \mathcal{U}(u)(B,y,x) = B + \frac{(F(x)-F(y)-B(x-y))u^T}{u^T(x-y)},$$

for all $(B,y,x) \in D_{\mathcal{U}(u)} = \{(B,y,x) \mid B \in L(\mathbb{R}^n),\ x,y \in D,\ u^T(x-y) \neq 0\}$. This class is also known as "*Broyden's class*".

We may also consider updating of an approximation to the inverse jacobian. Analogous to definition 3.12 we can give the following definition.

<u>3.13 DEFINITION</u>. Let F satisfy condition 1.5. Suppose $D_V \in L(\mathbb{R}^n) \times D \times D$. Then V is an *inverse-jacobian update function* for F with domain D_V if

$$V : D_V \to L(\mathbb{R}^n)$$

and

$$V(H,y,x)(F(x) - F(y)) = (x-y),$$

for all $(H,y,x) \in D_V$.

Analogous to the derivation of (3.17) we can define a class of inverse-jacobian update functions $\mathcal{V}(u)$ by

$$(3.18)\qquad \mathcal{V}(u)(H,y,x) = H + \frac{(x-y-H(F(x)-F(y)))u^TH}{u^TH(F(x)-F(y))}$$

for all $(H,y,x) \in D_{\mathcal{V}(u)} = \{(H,y,x) \mid H \in L(\mathbb{R}^n),\ x,y \in D,\ u^TH(F(x)-F(y)) \neq 0\}$.

<u>3.14. REMARK</u>. Let F satisfy condition 1.5 and let $B \in L(\mathbb{R}^n)$ be nonsingular. Then, for all $(B,y,x) \in D_{\mathcal{U}(u)}$ such that $(B^{-1},y,x) \in D_{\mathcal{V}(u)}$, the matrix $\mathcal{U}(u)(B,y,x)$ is nonsingular and

$$(3.19)\qquad (\mathcal{U}(u)(B,y,x))^{-1} = \mathcal{V}(u)(B^{-1},y,x).$$

This can easily be verified by using (3.17) and (3.18).

<u>3.15. REMARK</u>. Let $T \in L(\mathbb{R}^n)$ be arbitrary nonsingular and $u \in \mathbb{R}^n$. Suppose F satisfies condition 1.5. Denote $\mathcal{U}(u)$ and $\mathcal{V}(u)$ for F by U_F and V_F, respectively, and $\mathcal{U}(u)$ and $\mathcal{V}(u)$ for TF by U_{TF} and V_{TF}, respectively. Then

$$(B,y,x) \in D_{U_F} \iff (TB,y,x) \in D_{U_{TF}},$$

$$(H,y,x) \in D_{V_F} \iff (HT^{-1},y,x) \in D_{V_{TF}}$$

and

$$U_{TF}(TB,y,x) = TU_F(B,y,x)$$

$$(3.20)\qquad V_{TF}(HT^{-1},y,x) = V_F(H,y,x)T^{-1},$$

for all B, H, y and x such that $(B,y,x) \in D_{U_F}$ and $(H,y,x) \in D_{V_F}$.

In DENNIS [1971] an upper bound on the error $\|\mathcal{U}(u)(B,y,x) - J(x)\|$ is given relative to the error $\|B - J(y)\|$. These results are based on condition 1.6. For our purpose, however, we need a similar result for functions satisfying the affine invariant condition 1.7.

3.16. THEOREM. *Let* $x,y \in D$ *and* $H \in L(\mathbb{R}^n)$. *Suppose* $D_0 = \{z \mid z = y+t(x-y), t \in [0,1]\} \subset D$ *and* F *and* y *satisfy condition* 1.7 *on* D_0. *Define*

$$(3.21)\qquad e(y) = \|HJ(y) - I\|$$

and suppose $e(y) < 1$. *Let* $u \in \mathbb{R}^n$. *Then, for* $(H,y,x) \in D_{\mathcal{V}(u)}$,

$$(3.22)\qquad \|\mathcal{V}(u)(H,y,x)J(x) - I\| \le \left(\rho_1 \frac{e(y)}{1-e(y)} + \omega(y)(\rho_1+\tfrac{3}{2}\rho_2)\|x-y\|\right)(1+e(y)),$$

where

$$\rho_1 = \frac{\|H(F(x)-F(y))\|\,\|u\|}{|u^TH(F(x)-F(y))|}, \quad \rho_2 = \frac{\|x-y\|\,\|u\|}{|u^TH(F(x)-F(y))|}.$$

PROOF. For simplicity denote

$$J(y) = J, \quad J(x) = J^*, \quad \mathcal{V}(u)(H,y,x) = H^*,$$

$$p = x-y, \quad q = F(x)-F(y), \quad e(y) = e.$$

Then

$$H^*J^* - I = (HJ-I) + H(J^*-J) + \frac{(p-Hq)u^THJ}{u^THq} + \frac{(p-Hq)u^TH(J^*-J)}{u^THq}$$

and thus

$$(3.23)\qquad H^*J^* - I = -I + \left(I + \frac{(p-Hq)u^T}{u^THq}\right)HJ + \left(I + \frac{(p-Hq)u^T}{u^THq}\right)HJ(J^{-1}J^*-I).$$

By (3.21) and lemma 1.13 we know that HJ is nonsingular. Substituting

$$p - Hq = (p-J^{-1}q) + (J^{-1}H^{-1}-I)Hq$$

in the second term of the right hand side of (3.23) we obtain:

$$H^*J^* - I = \left((I-J^{-1}H^{-1}) + (J^{-1}H^{-1}-I)\frac{Hqu^T}{u^THq} + \frac{(p-J^{-1}q)u^T}{u^THq}\right) HJ$$

$$+ \left(I - \frac{Hqu^T}{u^THq} + \frac{pu^T}{u^THq}\right) HJ(J^{-1}J^*-I).$$

Hence

(3.24) $$H^*J^* - I = (I-J^{-1}H^{-1})\left(I - \frac{Hqu^T}{u^THq}\right) HJ + \frac{(p-J^{-1}q)u^T}{u^THq} HJ$$

$$+ \left(I - \frac{Hqu^T}{u^THq} + \frac{pu^T}{u^THq}\right) HJ(J^{-1}J^*-I).$$

For the spectral norm we can verify the following equality for $w,v \in \mathbb{R}^n$ with $v^Tw = 1$ (see BROYDEN [1970a, lemma 1]):

$$\|I - vw^T\| = \begin{cases} \|v\|\,\|w\| & \text{if } n \geq 2, \\ 0 & \text{if } n = 1. \end{cases}$$

Furthermore, applying lemma 1.16 on the convex set D_0 yields

(3.25) $$\|J^{-1}q - p\| \leq \tfrac{1}{2}\omega(y)\|p\|^2$$

and use of the perturbation lemma gives

$$\|I - J^{-1}H^{-1}\| \leq \frac{e}{1-e}\ .$$

Hence, for $n \geq 2$ and ρ_1, ρ_2 as given:

$$\|H^*J^* - I\| \leq \frac{1+e}{1-e}\,\rho_1 e + \tfrac{1}{2}\omega(y)(1+e)\rho_2\|p\| + \omega(y)(1+e)(\rho_1+\rho_2)\|p\|.$$

So that

$$\|H^*J^* - I\| \le (1+e)(\rho_1 \frac{e}{1-e} + \omega(y)(\rho_1+\tfrac{3}{2}\rho_2)\|p\|).$$

Finally, for n = 1, we obtain from (3.24) with

$$I - \frac{Hqu^T}{u^THq} = 0,$$

$$\|H^*J^* - I\| \le \left\| \frac{(p-J^{-1}q)u^THJ}{u^THq} \right\| + \left\| \frac{pu^TH(J^*-J)}{u^THq} \right\|.$$

Then, application of (3.25) yields

$$\|H^*J^* - I\| \le \tfrac{3}{2}\omega(y)\rho_2\|p\|(1+e).$$

Hence (3.22) is satisfied for all $n \ge 1$. □

3.17. REMARK. The choice

(3.26) $\quad u = H(F(x) - F(y))$

seems natural, because then $\rho_1 = 1$ and the nominator of ρ_2 will be nonzero as long as $H(F(x)-F(y))$ is nonzero. In the literature many other choices are proposed. The one most frequently used is (see BROYDEN [1969])

(3.27) $\quad u = x-y.$

3.18. REMARK. If F is linear then $\omega(y)$ can be chosen equal to zero. If u is chosen as in (3.26) then (3.22) reduces to

$$\|V(u)(H,y,x)J(x) - I\| \le \frac{e(1+e)}{1-e}.$$

Hence, if $H = (J(y))^{-1}$ then $e = 0$ and the error in the new approximation equals zero too. So the bound (3.22) is sharp in the sense that, if $e = 0$, there exist functions such that (3.22) is satisfied with equality sign. For general nonlinear functions (3.22) quarantees that the error in $V(u)(H,y,x)$ as an approximation to $(J(x))^{-1}$ will be small if $\|x-y\|$ is small and the error in H as an approximation to $(J(y))^{-1}$ is small.

3.4. FIXED APPROXIMATION

Let $y \in D$ be given and let $B \in L(\mathbb{R}^n)$ be some approximation to $J(y)$ with known (upper bound on the) error. Then we give two results on the error bounds for B as an approximation to $J(x)$ for arbitrary $x \in D$. We say that B is a *fixed approximation* to $J(x)$ for $x \in D$.

3.19. THEOREM. *Let* $x,y \in D$ *and* $H \in L(\mathbb{R}^n)$. *Suppose* $D_0 = \{z \mid z = y+t(x-y),\ t \in [0,1]\} \subset D$ *and* F *and* y *satisfy condition* 1.7 *on* D_0. *Define* $e(y)$ *by* (3.21). *Then*

$$\|HJ(x) - I\| \leq e(y) + \omega(y)(1+e(y))\|x-y\|. \tag{3.28}$$

PROOF. By the nonsingularity of $J(y)$ we obtain

$$\|HJ(x) - I\| \leq \|HJ(y) - I\|\|(J(y))^{-1}J(x)\| + \|(J(y))^{-1}J(x) - I\|.$$

Application of condition 1.7 in both terms of the right hand side yields the required result. □

3.20. THEOREM. *Let* $x,y \in D$ *and* $B \in L(\mathbb{R}^n)$. *Suppose* $D_0 = \{z \mid z = y+t(x-y),\ t \in [0,1]\} \subset D$ *and* F *and* y *satisfy condition* 1.6 *on* D_0. *Define*

$$e^+(y) = \|(J(y) - B)B^+\|.$$

Then

$$\|(J(x) - B)B^+\| \leq e^+(y) + \gamma(y)\|B^+\|\|x-y\|.$$

PROOF. We have

$$\|(J(x) - B)B^+\| \leq \|(J(y) - B)B^+\| + \|J(x) - J(y)\|\|B^+\|.$$

Condition 1.6 then yields the result. □

CHAPTER 4

DESCRIPTION OF NEWTON-LIKE METHODS

4.1. GENERAL DEFINITION

Throughout this chapter we assume that F satisfies condition 1.5. Suppose $x \in D$. Then we can approximate F by the linear function $\tilde{F}$ defined by

$$\tilde{F}(y) = F(x) + J(x)(y-x), \quad y \in D. \tag{4.1}$$

If $F(x) = 0$ has a solution, x^* say, in D, then we can approximate it by the solution, y^* say, of $\tilde{F}(y) = 0$, provided it exists in D. If $J(x)$ is nonsingular and $x - (J(x))^{-1}F(x) \in D$ then

$$y^* = x - (J(x))^{-1}F(x). \tag{4.2}$$

A well-known iterative method for solving (systems of) nonlinear equations, based on repeated use of (4.2) for given F, is *Newton's method*. This method defines, for each function F, a stationary iterative process which is defined by the iteration function $\Psi : D_\Psi \to \mathbb{R}^n$, with $D_\Psi = \{x \mid x \in D, J(x) \text{ nonsingular}\} \subset D$, such that

$$\Psi(x) = x - (J(x))^{-1}F(x), \quad \text{for all } x \in D_\Psi. \tag{4.3}$$

For general nonlinear F, the linear function $\tilde{F}$ given by (4.1) is a good approximation only for y in a small neighbourhood of x. So we can expect that Newton's method performs well if the starting point for the iterative process is close to a solution of $F(x) = 0$. Moreover, convergence to a solution for starting points far away from this solution is unlikely. In other words, we expect the local behaviour of Newton's method to be good but its global behaviour to be uncertain. In order to avoid divergence, which also may occur in Newton's method, one often modifies this method in such a way that in every iteration step a decrease of some level function is quaranteed. The iteration function is then given by

$$(4.4) \qquad \Psi(x) = x - \lambda(x)(J(x))^{-1}F(x), \qquad \text{for } x \in D_\Psi,$$

where the scalar $\lambda(x) > 0$ is chosen such that monotonicity of some *level function* $\|AF(x)\|$ ($A \in L(\mathbb{R}^n)$, nonsingular) is guaranteed, e.g.

$$\|AF(\Psi(x))\| < \|AF(x)\|.$$

It follows from a general result given in theorem 4.20 that such a $\lambda(x)$ exists. Such a method will be called a *restrained Newton method*; in the literature several other terms are used (e.g. "relaxation", "step size control", "damping"). A second drawback of Newton's method is the need for analytic expressions for the elements of the jacobian matrix. As this information is often not available one uses Newton's method with $J(x)$ in (4.3) or (4.4) replaced by some approximation to it. This jacobian approximation can depend on the previous iterate(s) or even on the jacobian approximation in the previous step (e.g. updating methods). Particularly, the latter possibility complicates the general definition of Newton-like methods as it forces us to consider the jacobian approximation or its inverse as part of the iterate. In our terminology this means that Newton-like processes generate, for given starting points, sequences of iterates of the form $\{(x_k,H_k)\} \subset \mathbb{R}^n \times L(\mathbb{R}^n)$ instead of sequences of iterates $\{x_k\} \subset \mathbb{R}^n$. Note that such iterative processes fit the theory about iterative processes given in section 1.2.

We now present the definition of a Newton-like method.

4.1. DEFINITION. Let F satisfy assumption 1.5. Let be given $\Psi_1 : D(\Psi_1) \to \mathbb{R}^n$, with $D(\Psi_1) \subset D \times L(\mathbb{R}^n)$, $\Psi_2 : D(\Psi_2) \to L(\mathbb{R}^n)$, with $D(\Psi_2) \subset D \times L(\mathbb{R}^n)$. Suppose that, for all $(x,H) \in D(\Psi_1)$, Ψ_1 satisfies

$$(4.5) \qquad \Psi_1(x,H) = x - \lambda(x,H)HF(x),$$

with $0 < \lambda(x,H) \leq 1$ for all $(x,H) \in D(\Psi_1)$. Then, the stationary iterative process defined by the iteration function $\Psi : D_\Psi \to \mathbb{R}^n \times L(\mathbb{R}^n)$, $D_\Psi = D(\Psi_1) \cap D(\Psi_2)$, (4.6) $\Psi(x,H) = (\Psi_1(x,H), \Psi_2(x,H))$, for all $(x,H) \in D_\Psi$

is called a *Newton-like process*. D_Ψ is the domain of definition of Ψ. If $\lambda(x,H) = 1$ for all $(x,H) \in D_\Psi$ then the process is called a *strict Newton-like process*, otherwise the process is called a *restrained Newton-like process*. $\lambda(x,H)$ is the *step length factor*.
Let M be a mapping $M : \mathcal{D} \subset F \rightarrow P(n^2+n)$ $(\mathcal{D} \neq \emptyset)$ (see def. 1.18), such that, for $F \in \mathcal{D}$, $M(F)$ defines a (strict or restrained) Newton-like process (where $\mathbb{R}^n \times L(\mathbb{R}^n)$ is identified with $\mathbb{R}^{n+n^2}$). Then M is a (*strict* resp. *restrained*) *Newton-like method*.

In the sequel we shall use the notation of definition 4.1 without comment.

In definition 4.1, x and $\Psi_1(x,H)$ are the old and new approximation, respectively, to a solution of $F(x) = 0$; H and $\Psi_2(x,H)$ are the approximations to the inverse jacobian at x and $\Psi_1(x,H)$, respectively. Note that for strict processes $(\lambda(x,H) \equiv 1)$, Ψ_1 is defined by (4.5) and $D(\Psi_1) = D \times L(\mathbb{R}^n)$. Examples of Newton-like methods that illustrate the definition are given in the next section.

Let $T \in L(\mathbb{R}^n)$ be nonsingular. Then, clearly, a solution of the equation $TF(x) = 0$ is also a solution of the equation $F(x) = 0$ and vice versa. However, if a particular Newton-like method yields an approximation to a solution of $F(x) = 0$, then it may be the case that an approximation to a different solution is found when the method is used to solve $TF(x) = 0$, even if we use the same initial guess to the solution. Moreover, it may be possible that, for a given initial guess, the sequence of approximations to the solution generated by the process for solving $TF(x) = 0$ converges, but for solving $F(x) = 0$ does not converge. Such events are most undesirable as in practice one is interested in a solution of $F(x) = 0$ and it does not matter whether this solution is obtained by applying an iterative method to F or to TF. Furthermore, most practical problems are scaled in a way which depends on the choice of dimensions, and one often is not aware of the effect of this choice on the iterative method for solving the problem.
In order to obtain insight in the dependency of a Newton-like method on an affine transformation of the function we introduce the concept of affine invariancy.

4.2. DEFINITION. Let $\mathcal{M}$ be a Newton-like method with domain of definition $\mathcal{D} \subset \mathcal{F}$. Then $\mathcal{M}$ is *affine invariant* if for all $F \in \mathcal{D}$ and all nonsingular $T \in L(\mathbb{R}^n)$ the following conditions are satisfied:

1. $TF \in \mathcal{D}$;
2. if $\{(x_k,H_k)\}$ is generated by $\mathcal{M}(F)$, for a starting point (x_0,H_0) in the domain of $\mathcal{M}(F)$, and $\{x_k',H_k'\}$ is generated by $\mathcal{M}(TF)$ for starting point $(x_0',H_0') = (x_0,H_0T^{-1})$, then $x_k = x_k'$ (for k=1,2,...).

4.3. REMARK. Let $\mathcal{M}$ be a strict Newton-like method with domain of definition $\mathcal{D} \subset \mathcal{F}$. Denote for all $F \in \mathcal{D}$:

$$\mathcal{M}(F) = \Psi_F = (\Psi_{1,F},\Psi_{2,F}). \tag{4.7}$$

Then $\mathcal{M}$ is affine invariant if for all $F \subset \mathcal{D}$ and all nonsingular $T \in L(\mathbb{R}^n)$:

$$\Psi_{2,TF}(x,HT^{-1}) = (\Psi_{2,F}(x,H))T^{-1}. \tag{4.8}$$

This property follows directly from (4.5). In fact, starting from (x_0,H_0) and (x_0,H_0T^{-1}) respectively, we obtain sequences $\{(x_k,H_k)\}$ and $\{(x_k,H_kT^{-1})\}$ for F and TF, respectively.

4.4. REMARK. It should be noted that we consider exact methods in this section. Affine invariancy of a method can be spoiled if inexact arithmetic is used, as numerical processes may depend on scaling of the function. For instance, pivoting strategies for triangularization of a matrix may depend on scaling of the matrix.

4.2. EXAMPLES OF NEWTON-LIKE METHODS

4.2.1. Newton methods

4.5. DEFINITION. Let M be a Newton-like method. Suppose that, for all $F \in \mathcal{D}$, $\Psi = M(F)$ satisfies

$$(4.9) \qquad D(\Psi_2) = D_\Psi = \{(x,H) \mid (x,H) \in D(\Psi_1),\ y = \Psi_1(x,H) \in D,\ J(y) \text{ nonsingular}\},$$

$$(4.10) \qquad \Psi_2(x,H) = (J(\Psi_1(x,H)))^{-1}, \quad \text{for } (x,H) \in D_\Psi.$$

Then M is a *Newton method* and $M(F) = \Psi$ defines a *Newton process* for $F \in \mathcal{D}$.

Suppose $(x_0,(J(x_0))^{-1})$ belongs to D_Ψ, as given in definition 4.5. Then, the sequence $\{(x_k,H_k)\}$ generated by a Newton process for the starting point $(x_0,H_0) = (x_0,(J(x_0))^{-1})$, satisfies

$$(4.11) \qquad \begin{aligned} x_{k+1} &= x_k - \lambda_k (J(x_k))^{-1} F(x_k), \\ (H_{k+1} &= (J(x_{k+1}))^{-1}), \end{aligned} \qquad k = 0,1,\ldots,$$

where $\lambda_k = \lambda(x_k,(J(x_k))^{-1})$ $(k=0,1,\ldots)$. This is the usual definition of a Newton method. Note that in our definition it can occur that $H_0 \neq (J(x_0))^{-1}$ in a Newton process.

4.6. REMARK. The strict Newton method is affine invariant. This follows easily from (4.10) and remark 4.3. For restrained Newton methods, it depends on the choice of $\lambda(x,H)$ whether such a method is affine invariant.

4.7. REMARK. Newton methods have two drawbacks.

1. The jacobian $J(x)$ has to be calculated in every iteration step, which may be difficult or even impossible for real life problems.
2. Calculating x_{k+1} from x_k (for $k=0,1,\ldots$) requires the computation of $J(x_k)\backslash F(x_k)$ (see (1.13)) which is relatively expensive in terms of basic arithmetical operations for high dimensional problems. Moreover, this computation is undefined for singular $J(x_k)$.

4.2.2. Difference Newton methods

4.8. DEFINITION. Let M be a Newton-like method. Let $B(x)$ be the difference approximation given by (3.5) for $h_i \in \mathbb{R}$, $h_i \neq 0$ and h_i dependent on x and F $(i=1,\ldots,n)$. Suppose that for all $F \in \mathcal{D}$, $\Psi = M(F)$ satisfies

(4.12) $$D(\Psi_2) = D_\Psi = \{(x,H) \mid (x,H) \in D(\Psi_1),\ y = \Psi_1(x,H) \in D,\ y+h_je_j \in D\ (j=1,\ldots,n),\ B(y) \text{ nonsingular}\},$$

(4.13) $$\Psi_2(x,H) = (B(\Psi_1(x,H)))^{-1}, \text{ for } (x,H) \in D_\Psi.$$

Then M is a *difference Newton method* and $M(F) = \Psi$ defines a *difference Newton process* for all $F \in \mathcal{D}$.

Suppose $(x_0,(B(x_0))^{-1})$ belongs to D_Ψ, as given in definition 4.8. Then the sequence $\{(x_k,H_k)\}$ generated by a difference Newton process for the starting point $(x_0,H_0) = (x_0,(B(x_0))^{-1})$, satisfies

(4.14) $$\begin{aligned} x_{k+1} &= x_k - \lambda_k(B(x_k))^{-1}F(x_k), \\ (H_{k+1} &= (B(x_{k+1}))^{-1}), \end{aligned} \qquad k = 0,1,\ldots,$$

where $\lambda_k = \lambda(x_k,(B(x_k))^{-1})$ $(k=0,1,\ldots)$. It seems natural to choose $(x_0,(B(x_0))^{-1})$ as a starting point for the difference Newton process, although with our definition other starting points are imaginable.

4.9. REMARK. From (3.4), (4.13) and remark 4.3 it follows that the strict difference Newton method is affine invariant.

4.10. REMARK.

1. Difference Newton methods do not require the calculation of the (analytic) jacobian at every iteration step. Note, however, that calculation of a difference approximation at x, requires the calculation of the function at n extra points around x.
2. As for Newton methods, in every iteration step a linear system has to be solved (see also remark 4.7 ad 2).

4.2.3. Updating Newton methods

We distinguish methods using jacobian update functions and those using inverse-jacobian update functions (see definitions 3.12 and 3.13).

4.11. DEFINITION. Let M be a Newton-like method. Let $u : D \times L(\mathbb{R}^n) \to \mathbb{R}^n$ be given. Suppose that, for all $F \in \mathcal{D}$, $\Psi = M(F)$ satisfies

$$(4.15) \qquad D(\Psi_2) = D_\Psi = \{(x,H) \mid (x,H) \in D(\Psi_1),\ H \text{ nonsingular},\ y = \Psi_1(x,H) \in D,\ (y-x)^T u(x,H) \neq 0,\ (F(y)-F(x))^T H^T u(x,H) \neq 0\},$$

$$(4.16) \qquad \Psi_2(x,H) = (\mathcal{U}(u(x,H))(H^{-1},x,\Psi_1(x,H)))^{-1}, \text{ for } (x,H) \in D_\Psi,$$

with $\mathcal{U}$ defined by (3.17). Then M is an *updating Newton method* and $M(F) = \Psi$ defines an *updating Newton process* for all $F \in \mathcal{D}$. (Note that $\Psi_2(x,H)$ exists for $(x,H) \in D_\Psi$ because of remark 3.14 and the definition of D_Ψ.)

4.12. DEFINITION. Let M be a Newton-like method. Let $u : D \times L(\mathbb{R}^n) \to \mathbb{R}^n$ be given. Suppose that, for all $F \in \mathcal{D}$, $\Psi = M(F)$ satisfies (4.15) and

$$(4.17) \qquad \Psi_2(x,H) = \mathcal{V}(u(x,H))(H,x,\Psi_1(x,H)), \quad \text{for } (x,H) \in D_\Psi,$$

with $\mathcal{V}$ defined by (3.18). Then M is an *inverse-updating Newton method* and $M(F) = \Psi$ defines an *inverse-updating Newton process* for all $F \in \mathcal{D}$.

Let (x_0,H_0) belong to D_Ψ as given in definition 4.11, then the sequence $\{(x_k,H_k)\}$ generated by this process for starting point (x_0,H_0), satisfies

$$(4.18) \qquad \begin{aligned} x_{k+1} &= x_k - \lambda_k H_k F(x_k) \\ H_{k+1} &= (\mathcal{U}(u_k)(H_k^{-1},x_k,x_{k+1}))^{-1} \end{aligned} \quad (k=0,1,\ldots),$$

where $\lambda_k = \lambda(x_k,H_k)$, $u_k = u(x_k,H_k)$ $(k=0,1,\ldots)$. Note that H_{k+1} depends on H_k.

Similarly for an inverse-updating Newton process we obtain

$$(4.19) \qquad \begin{aligned} x_{k+1} &= x_k - \lambda_k H_k F(x_k) \\ H_{k+1} &= \mathcal{V}(u_k)(H_k,x_k,x_{k+1}). \end{aligned} \quad (k=0,1\ldots),$$

where $\lambda_k = \lambda(x_k,H_k)$, $u_k = u(x_k,H_k)$ $(k=0,1,\ldots)$.

4.13. REMARK. From the remarks 3.15 and 4.3 it follows that a strict (inverse-) updating Newton method is affine invariant.

4.14. REMARK.

1. (Inverse-) updating Newton methods neither require the calculation of the (analytic) jacobian in an iteration step, nor extra function evaluations to obtain an approximation to the (inverse) jacobian.
2. Updating Newton methods store and update an approximation to the jacobian (in (4.16) and (4.18) formally denoted by H^{-1} and H_k^{-1}) and solve a linear system with this approximation to the jacobian, at every iteration step. Note that the condition $(x,H) \in D_\Psi$ is easily checked so that problems of trespassing the domain can be avoided.
3. Inverse-updating Newton methods store and update approximations to the inverse jacobian. Thus solutions of linear systems do not have to be calculated in these methods.

4.2.4. Fixed Newton methods

4.15. DEFINITION. Let $\mathcal{M}$ be a Newton-like method. Suppose that, for all $F \in \mathcal{D}$, $\Psi = \mathcal{M}(F)$ satisfies

$$(4.20) \qquad D(\Psi_2) = D_\Psi = D(\Psi_1),$$

$$(4.21) \qquad \Psi_2(x,H) = H, \quad \text{for } (x,H) \in D_\Psi.$$

Then $\mathcal{M}$ is a *fixed Newton method* and $\mathcal{M}(F) = \Psi$ defines a *fixed Newton process* for all $F \in \mathcal{D}$.

Let (x_0,H_0) belong to D_Ψ as given in definition 4.15, then the sequence $\{(x_k,H_k)\}$ generated by this process for starting point (x_0,H_0) satisfies

$$(4.22) \qquad \begin{aligned} x_{k+1} &= x_k - \lambda_k H_k F(x_k) \\ H_{k+1} &= H_0, \end{aligned} \qquad (k=0,1,\ldots),$$

with $\lambda_k = \lambda(x_k,H_k)$ $(k=0,1,\ldots)$. Note that H_{k+1} depends on H_0 only.

4.16. REMARK. It follows easily from the definition of affine invariancy (definition 4.2) that the strict fixed Newton method is affine invariant.

4.17. REMARK.

1. Fixed Newton methods do not require calculations to obtain $\Psi_2(x,H)$.
2. Fixed Newton methods do not require the solution of a linear system during the iteration steps.

4.3. RESULTS FOR RESTRAINED NEWTON-LIKE METHODS

In this section we assume that $A \in L(\mathbb{R}^n)$ is nonsingular. We restrict attention to the class of restrained Newton-like methods M for which, for all $F \in \mathcal{D}$, $\Psi = M(F)$ satisfies the following monotonicity criterion

$$(4.23) \qquad \|AF(\Psi(x,H))\| < \|AF(x)\|, \quad \text{for all } (x,H) \in D_\Psi.$$

It should be emphasized that for these methods, in general, $D(\Psi_1) \neq D \times L(\mathbb{R}^n)$, as $\lambda(x,H) \in (0,1]$ satisfying (4.23) not always exists for $(x,H) \in D \times L(\mathbb{R}^n)$. Furthermore, if there exists a $\lambda(x,H) \in (0,1]$ satisfying (4.23), it is, in general, not uniquely determined by (4.23). It is also important to note that for arbitrary nonsingular $A \in L(\mathbb{R}^n)$, restrained Newton-like methods satisfying (4.23) may not be affine invariant. The usual methods for choosing $\lambda(x,H)$ yield only affine invariant methods for special choices of A. In particular for choices of A satisfying

$$(4.24) \qquad A_F T^{-1} = A_{TF},$$

for all nonsingular $T \in L(\mathbb{R}^n)$ and where the subscript F expresses dependency on F. For instance, if we choose A equal to the inverse jacobian at some point $x \in D$, or to some suitable approximation to it (difference or update approximation), then it satisfies (4.24).

4.18. REMARK. Let M be a restrained Newton method. Suppose that, for all $F \in \mathcal{D}$, $\Psi = M(F)$ satisfies (4.23) for some $A \in L(\mathbb{R}^n)$ (A nonsingular). Then we say that M is an *implicitly scaling* restrained Newton method, with *implicit scaling matrix* A. Similar terminology will be used for the other Newton-like methods given in section 4.2. This terminology is due to the fact that scaling of the function only influences the value of $\lambda(x,H)$, for the methods given in section 4.2 and for the usual choices of $\lambda(x,H)$ satisfying (4.23). Solving $F(x) = 0$ with an implicitly scaling restrained Newton-like method with implicit scaling matrix A is equivalent to solving $AF(x) = 0$ with the method with implicit scaling matrix I.

We shall now derive sufficient conditions for existence of a step length factor in the interval (0,1] such that (4.23) is satisfied.

4.19. LEMMA. *Let* $x \in D$, $F(x) \neq 0$ *and* F *satisfies condition* 1.5. *Let* $H \in L(\mathbb{R}^n)$ *and define*

$$(4.25) \qquad e(x) = \|HJ(x) - I\|.$$

Suppose that $J(x)$ *is nonsingular and define*

$$(4.26) \qquad \kappa(x) = \|AJ(x)\|\,\|(J(x))^{-1}F(x)\|/\|AF(x)\|,$$

for $A \in L(\mathbb{R}^n)$ *nonsingular. Denote* $z(t) = x-tHF(x)$ *and let* $t_1 > 0$ *be such that* $z(t) \in D$ *for all* $t \in [0,t_1)$. *Define*

$$(4.27) \qquad \phi(t) = \|AF(z(t))\|^2 \qquad (0 \le t < t_1).$$

Then,

$$(4.28) \qquad e(x)\kappa(x) < 1$$

implies

$$(4.29) \qquad \phi'(0) \le -2(1-\kappa(x)e(x))\|AF(x)\|^2 < 0$$

and equality holds in (4.29) *if* $e(x) = 0$. *Moreover, there exist a function* F, *a point* $x \in D$ *and a nonsingular matrix* $A \in L(\mathbb{R}^n)$ *such that for any* $\varepsilon > 0$ *with* $\varepsilon\kappa(x) < 1$ *there is a matrix* $H = H_\varepsilon$ *with* $e(x) = \varepsilon$ *and* (4.29) *is satisfied with the equality sign.*

PROOF. Using the Cauchy-Schwarz inequality we obtain

$$|[A(J(x)H-I)F(x), AF(x)]| \le \|A(J(x)H-I)F(x)\|\,\|AF(x)\|$$

$$= \|AJ(x)(HJ(x)-I)(J(x))^{-1}F(x)\|\,\|AF(x)\| \le \kappa(x)e(x)\|AF(x)\|^2.$$

Differentiation of $\phi(t)$ yields

$$\phi'(t) = -2[AJ(z(t))HF(x), AF(z(t))], \qquad t \in [0,t_1).$$

Hence

$$\phi'(0) = -2([A(J(x)H-I)F(x), AF(x)] + [AF(x), AF(x)])$$

$$\le -2(1-\kappa(x)e(x))\|AF(x)\|^2,$$

which proves (4.29).

If $e(x) = 0$, then $H = J(x)$ and $\phi'(0) = -2[AF(x),AF(x)]$, so that equality holds in (4.29).

Now consider $F(x) = (\xi_1,\xi_2)^T$, for $x = (\xi_1,\xi_2)^T$. Choose $A = I$, $x = (1,1)^T$ and for $e < 1/\kappa(x) = 1$:

$$H = \begin{pmatrix} 1 & -e \\ -e & 1 \end{pmatrix}.$$

Then

$$\|HJ(x) - I\| = e$$

and

$$\phi'(0) = -4(1-e) = -2(1-\kappa(x)e)\|AF(x)\|^2. \quad \square$$

<u>4.20. THEOREM</u>. *Let M be a restrained Newton-like method. Suppose that, for some nonsingular $A \in L(\mathbb{R}^n)$ and all $F \in \mathcal{D}$, $\Psi = M(F)$ satisfies (4.23) and*

$$(4.30) \qquad D(\Psi_1) = \{(x,H) \mid (x,H) \in D \times L(\mathbb{R}^n), \exists t \in (0,1]: z = x-tHF(x) \in D, \|AF(z)\| \le \|AF(x)\|\}.$$

Define

$$(4.31) \qquad S = \{(x,H) \mid (x,H) \in D \times L(\mathbb{R}^n), F(x) \neq 0, J(x) \text{ is nonsingular}, \|HJ(x)-I\| < 1/\kappa(x)\},$$

where $\kappa(x)$ is defined by (4.26). Then $S \subset D(\Psi_1)$.

<u>PROOF</u>. For $(x,H) \in S$, it follows from lemma 4.19 that the derivative of $\|AF(y)\|^2$ at $y = x$ is negative in the direction $-HF(x)$. As D is open and non-empty there exists a $t_1 \in (0,1]$: $x-tHF(x) \in D$ for $t \in (0,t_1)$. Hence, $(x,H) \in D(\Psi_1)$. $\square$

<u>4.21. COROLLARY</u>. *There exist restrained Newton-like methods M, such that for all nonsingular $A \in L(\mathbb{R}^n)$ and all $F \in \mathcal{D}$ for which $J(x)$ is nonsingular for some $x \in D$ with $F(x) \neq 0$, $\Psi = M(F)$ has a nonempty domain of definition.*

<u>PROOF</u>. We only have to demand that $S \cap D(\Psi_2) \neq \emptyset$. As $S \neq \emptyset$, this holds for all the examples of Newton-like methods given in section 4.2 and for all nonsingular $A \in L(\mathbb{R}^n)$. $\square$

As is shown in the next examples, the assumptions in theorem 4.20 are not very strong. In fact (4.30) imposes the natural condition on the strategy for choosing the step-length factor $\lambda(x,H)$: if there exist a $t \in (0,1]$ such that $z = x-tHF(x) \in D$ and $\|AF(z)\| \leq \|AF(x)\|$, then there exists a $\lambda(x,H)$, i.e. if an appropriate $\lambda(x,H)$ can be found, then the strategy is such that an appropriate $\lambda(x,H)$ will be found. So, all restrained Newton-like processes, given in section 4.2, which have a "natural" restraining strategy, have non-empty domain of definition.

4.22. EXAMPLES.

1. The restrained Newton method M with the following restraining strategy for some nonsingular $A \in L(\mathbb{R}^n)$: for all $(x,H) \in D \times L(\mathbb{R}^n)$:

$$\lambda(x,H) = 2^{-p},$$

where p is the smallest nonnegative integer such that

$$x - 2^{-p}HF(x) \in D$$
$$\|AF(x-2^{-p}HF(x))\| < \|AF(x)\|$$

for $F \in \mathcal{D}$. If $(x,H) \in S$ (cf. (4.31)) then existence of such a $\lambda(x,H)$ is guaranteed.

2. The restrained Newton method M with the following restraining strategy for some nonsingular $A \in L(\mathbb{R}^n)$: for all $(x,H) \in D \times L(\mathbb{R}^n)$:

$$\lambda(x,H) = t^*$$

with

$$t^* \in (0,t_1] \subset (0,1], \; x-tHF(x) \in D, \text{ for all } t \in (0,t_1],$$
$$\|AF(x-t^*HF(x))\| \leq \|AF(x-tHF(x))\|, \qquad t \in (0,t_1],$$

for $F \in \mathcal{D}$. Note that this strategy does not satisfy (4.30) for all $F \in \mathcal{F}$. As D is open it might occur that $\|AF(x-tHF(x)\|$ does not have a minimum with respect to t such that $x-tHF(x) \in D$. Compactness of $S_F(x,A)$ is a sufficient condition for existence of t^*. Therefore $\mathcal{D}$ has to be restricted to functions satisfying such a condition for some $x \in D$. Note that, in general, t^* is also not determined uniquely with the strategy.
In practice, a t^* can be approximated by successive quadratic or cubic

interpolation (see BRENT [1973b]).

Of course, restrained difference, updating or fixed Newton methods, with restraining strategies as above, are also examples of methods for which the processes for given functions in some class, have nonempty domain of definition. Such methods are affine invariant if the implicit scaling matrix satisfies (4.24) for all functions in the domain of definition of the method.

4.4. USE OF GENERALIZED INVERSE

In most examples of Newton-like methods given in section 4.2 $\Psi_2(x,H)$ is defined as the inverse of an approximation to $J(\Psi_1(x,H))$. Therefore, the domain of definition of Ψ_2, $D(\Psi_2)$, is restricted to those (x,H) for which this inverse exists. We can avoid problems with nonsingularity of $J(\Psi_1(x,H))$, or an approximation to it, by using the generalized inverse. We obtain the following Newton-like methods:

4.23. DEFINITION. Let M be a Newton-like method. Suppose that, for all $F \in \mathcal{D}$, $\Psi = M(F)$ satisfies

$$D(\Psi_2) = D_\Psi = \{(x,H) \mid (x,H) \in D(\Psi_1),\ \Psi_1(x,H) \in D\}, \tag{4.32}$$

$$\Psi_2(x,H) = (J(\Psi_1(x,H)))^+, \quad \text{for } (x,H) \in D_\Psi. \tag{4.33}$$

Then M is a *generalized Newton method* and $M(F)$ defines a *generalized Newton process*.

4.24. DEFINITION. Let, for F satisfying condition 1.5 and $x \in D$, $B(x)$ be the difference approximation given by (3.5) for $h_i \in \mathbb{R}$, $h_i \neq 0$, h_i depending on x and F $(i=1,\dots,n)$. Let M be a Newton-like method. Suppose, for all $F \in \mathcal{D}$, $\Psi = M(F)$ satisfies

$$D(\Psi_2) = D_\Psi = \{(x,H) \mid (x,H) \in D(\Psi_1),\ \Psi_1(x,H) \in \mathbf{D}, B(\Psi_1(x,H)) \text{ exists}\}, \tag{4.34}$$

$$\Psi_2(x,H) = (B(\Psi_1(x,H)))^+, \quad \text{for } (x,H) \in D_\Psi. \tag{4.35}$$

Then M is a *generalized difference Newton method* and $M(F)$ defines a *generalized difference Newton process*.

4.25. REMARK. We shall not define "generalized updating Newton methods". Use of the generalized inverse of update approximations to the jacobian is undesirable in our opinion, as we should not allow that the update approximation becomes singular for the following reasons:

1. If B is a singular approximation to $J(y)$, then $Bv = 0$ for all $v \in \ker(B)$, $H = B^+$. Consider the two choices of $u(y,H)$ (see remark 3.17): $u(y,H) = H(F(x)-F(y))$ and $u(y,H) = x-y = \lambda(y,H)HF(y)$, for $x = \Psi_1(y,H)$.

Then we see that $u(y,H) \in (\ker(B))^c$ and therefore $(u(y,H))^T v = 0$. So, it follows easily from (3.17) that $\mathcal{U}(u(y,H))(B,y,x)$ is singular. In other words, once an update approximation, with u as above, is singular, then it remains singular after updating, independent of the function. This is highly undesirable as the sequence of approximations to a solution then remains in a subset of D which may not contain a solution.

2. It is easy to avoid nonsingularity of $\mathcal{U}(u(y,H))(B,y,x)$ if B is nonsingular. If $(x-y)u(y,H) \neq 0$ and $(F(x)-F(y))^T H^T u(y,H) \neq 0$ then $\mathcal{U}(u(y,H))(B,y,x)$ is bounded and nonsingular. Therefore, performing these a-priori checks and choosing another method of approximation if unboundedness or singularity would occur, will avoid singularity, if the other method quarantees nonsingularity. An obvious choice is using fixed approximation, if B is nonsingular and the updating approximation will yield a singular matrix.

Theorems 4.19 and 4.20 and corollary 4.21 are to restrictive for generalized restrained Newton-like methods, as in those results we consider only points in the domain of definition of processes for which the jacobian of the function is nonsingular (cf. definition of S in (4.31)). In fact, the definition of the error in H with respect to J(x), for some $x \in D$, as given in (4.25) is not appropriate if H is obtained as the generalized inverse of some approximation to J(x). Let $H = (J(x))^+$ for some $x \in D$ and suppose J(x) is singular. Then

$$HJ(x) = V_1 I_r V_1^T$$

where the singular value decomposition is given by $J(x) = U_1 \Sigma_r V_1^T$ (cf. (1.18)). Hence, if $r < n$, then

$$\|HJ(x) - I\| = 1.$$

So, although H is the best approximation possible, we obtain for the error according to (4.25) the value 1. Nevertheless, the domain of definition of generalized Newton-like processes is not empty. This will be shown by the following results.

<u>4.26. LEMMA</u>. *Let* $x \in D$, $F(x) \neq 0$ *and* $A \in L(\mathbb{R}^n)$ *be nonsingular and let* F *satisfy condition* 1.5. *Define for* $B \in L(\mathbb{R}^n)$:

$$(4.36) \qquad e^{+}(x) = \|(J(x)-B)B^{+}\|,$$

$$(4.37) \qquad \zeta(x) = [ABB^{+}F(x), AF(x)]/\|AF(x)\|^{2}.$$

Denote $z(t) = x-tB^{+}F(x)$ *and let* $t_1 > 0$ *be such that* $z(t) \in D$ *for all* $t \in [0,t_1)$. *Suppose that*

$$(4.38) \qquad e^{+}(x)\|A\|\|A^{-1}\| < \zeta(x),$$

Then $\phi(t) = \|AF(z(t))\|^{2}$ *satisfies*

$$(4.39) \qquad \phi'(0) \le -2(\zeta(x)-e^{+}(x)\|A\|\|A^{-1}\|)\|AF(x)\|^{2}.$$

Furthermore, if B *is nonsingular then* $\zeta(x) = 1$ *and equality may hold as in lemma* 4.19. *Moreover, there exist a function* F, *a point* x *and a nonsingular matrix* A, *such that for all* $e^{+} < 1/\|A\|\|A^{-1}\|$, *there is a singular matrix* B *satisfying the conditions with* $e^{+}(x) = e^{+}$, *such that equality holds in* (4.39).

<u>PROOF</u>.

$$\phi'(t) = -2[AJ(z(t))B^{+}F(x), AF(z(t))],$$

for $t \in [0,t_1)$. Hence

$$\phi'(0) = -2\{[A(J(x)-B)B^{+}F(x), AF(x)] + [ABB^{+}F(x), AF(x)]\}$$

$$\le -2(\zeta(x)-e^{+}(x)\|A\|\|A^{-1}\|)\|AF(x)\|^{2}.$$

For

$$F(x) = (\xi_1,-\varepsilon\xi_2)^{T}, \quad x = (\xi_1,\xi_2)^{T} = (1,0)^{T}, \quad A = I$$

and

$$B = \begin{pmatrix} (1-\varepsilon)^{-1} & 0 \\ 0 & 0 \end{pmatrix}$$

equality holds in (4.39). □

<u>4.27. THEOREM</u>. *Let* M *be a generalized restrained Newton-like method. Suppose that for some nonsingular* $A \in L(\mathbb{R}^{n})$ *and all* $F \in \mathcal{D}$, $\Psi = M(F)$ *satisfies* (4.23) *and* (4.30). *Define, for* $\zeta(x)$ *given by* (4.37),

(4.40) $$S = \{(x,H) \mid (x,H) \in D \times L(\mathbb{R}^n),\ F(x) \neq 0,\ H = B^+ \textit{ for some } B \in L(\mathbb{R}^n),\ \|(J(x)-B)B^+\|\|A\|\|A^{-1}\| < \zeta(x)\}.$$

Then $S \subset D(\Psi_1)$.

PROOF. For $(x,H) \in S$ it follows from lemma 4.26 that the derivative of $\|AF(y)\|^2$ at $y = x$ is negative in the direction $-HF(x)$. As D is open and non-empty, there exists a $t_1 \in (0,1]$: $x-tHF(x) \in D$ for $t \in (0,t_1)$. Therefore $(x,H) \in D(\Psi_1)$. □

4.28. REMARK. Similar to corollary 4.21 there exists generalized restrained Newton-like methods $\mathcal{M}$, such that for all nonsingular $A \in L(\mathbb{R}^n)$ and all $F \in \mathcal{D}$ for which $F(x) \neq 0$ for some $x \in D$, $\Psi = \mathcal{M}(F)$ has a nonempty domain of definition. The generalized restrained Newton or difference Newton methods with restraining strategies as given in example 4.22 generate such processes with nonempty domains of definitions.

Finally we state some specific results for generalized Newton-like methods.

4.29. LEMMA. *The generalized strict Newton method is not affine invariant.*

PROOF. This follows from the following example.

$$F(x) = \begin{pmatrix} \xi_1^2 \\ \xi_2^2 + 1 \end{pmatrix}, \quad x_0 = \begin{pmatrix} 0.5 \\ 0 \end{pmatrix}, \quad T = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Then

$$J(x) = \begin{pmatrix} 2\xi_1 & 0 \\ 0 & 2\xi_2 \end{pmatrix}, \quad F(x_0) = \begin{pmatrix} 0.25 \\ 1 \end{pmatrix}.$$

Hence

$$J(x_0) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad (J(x_0))^+ = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

and

$$(J(x_0))^+F(x_0) = \begin{pmatrix} 0.25 \\ 0 \end{pmatrix}.$$

However,

$$TF(x) = \begin{pmatrix} \xi_1^2 \\ \xi_1^2 + \xi_2^2 + 1 \end{pmatrix}, \quad TJ(x) = \begin{pmatrix} 2\xi_1 & 0 \\ 2\xi_1 & 2\xi_2 \end{pmatrix}$$

and

$$TJ(x_0) = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \quad (TJ(x_0))^+ = \begin{pmatrix} 0.5 & 0.5 \\ 0 & 0 \end{pmatrix}.$$

With

$$TF(x_0) = \begin{pmatrix} 0.25 \\ 1.25 \end{pmatrix}$$

we obtain

$$(TJ(x_0))^+ TF(x_0) = \begin{pmatrix} 0.75 \\ 0 \end{pmatrix} \neq \begin{pmatrix} 0.25 \\ 0 \end{pmatrix} = (J(x_0))^+ F(x_0). \quad \square$$

4.30. REMARK. The reason that the strict generalized Newton method is not affine invariant is that projection (which is in fact induced if the generalized inverse of a singular jacobian is used, see subsection 1.3.4) and affine transformation do not commute. Of course, similar counterexamples can be constructed to prove that the strict generalized difference Newton method and the usual restrained versions of these methods are not affine invariant.

4.31. THEOREM. *Let* F *satisfy condition* 1.5, $x \in D$ *and* $F(x) \neq 0$. *Suppose*

(4.41) $\quad [J(x)(J(x))^+F(x),\ F(x)] = 0.$

Then, the function f*, defined by* $f(y) = \|F(y)\|^2$ *for* $y \in D$*, has a stationary point at* $y = x$.

PROOF. Suppose the singular value decomposition of $J(x)$ is given by (cf. (1.18))

$$J(x) = U_1 \Sigma_r V_1^T,$$

with r the rank of $J(x)$. By lemma 1.26 we see that $J(x)(J(x))^+ = U_1U_1^T$. It follows from (4.41) that $U_1^T F(x) = 0$. Therefore,

$$f'(x) = 2J^T(x)F(x) = 2V_1\Sigma_r U_1^T F(x) = 0.$$

So f has a stationary point at y = x. □

4.32. REMARK. It should be noted that $[AJ(x)(J(x))^+F(x), AF(x)] = 0$, for arbitrary nonsingular $A \in L(\mathbb{R}^n)$, does not necessarily imply that $\|AF(y)\|$ or $\|F(y)\|$ have a stationary point at y = x. This is also due to the fact that projection and affine transformation do not commute.

4.33. REMARK. In section 4.3 we introduced the matrix A (see (4.23)) in order to be able to construct affine invariant restrained Newton-like methods. It appears from lemma 4.29 and remark 4.30 that restrained Newton-like methods using the generalized inverse are, in general, not affine invariant. Therefore, in such methods we have lost our motivation for choosing A unequal I. The choice A = I will simplify lemma 4.26 and theorem 4.27 considerably, as in that case $\sqrt{\zeta(x)}$ in (4.37) is the ratio of the norm of the projection of F(x) on the range of B and the norm of F(x) itself.

4.5. FINAL REMARKS

In this chapter we have described most of the well-known Newton-like methods in a formal way. Of course, other methods may be designed by mixing the methods given here. For example, one may use fixed Newton or updating Newton methods in the last steps, if the approximate solution, obtained after some steps with a difference Newton method, is known to be close to a true solution. Furthermore, one may perform a fixed Newton step if updating is undefined in some step of an updating Newton method, or a step with generalized inversion if classical inversion is undefined. In fact one may create many different combinations.

The most important examples of the strict Newton-like methods of section 4.2 are affine invariant. For restrained Newton-like methods it depends on the restraining strategy whether these methods are affine invariant. Strategies satisfying the monotonicity condition (4.23) with A chosen appropriately (satisfying (4.24)) will sometimes do. For example, the bisection and interpolation strategy described in examples 4.22. If generalized inversion is used then affine invariancy can be lost for functions having a singular jacobian (approximation) at some iteration point. We think that affine invariancy is a desirable property for Newton-like methods, as one may expect, at least theoretically, that such methods do not suffer from bad scaling of the function. On the other hand, one often has to deal with singular jacobian approximations in practice. Such matrices can be handled elegantly by use of the generalized inverse. Therefore we shall not restrict attention to affine invariant methods.

CHAPTER 5

CONVERGENCE OF NEWTON-LIKE METHODS

5.1. INTRODUCTION

In this chapter we shall present convergence results for Newton-like methods as defined in section 4.1. Convergence results for iterative methods may be divided into three classes (cf. RHEINBOLDT [1974]).

(i) *Global convergence theorems*. These theorems state existence of a unique solution in the domain of the function or at least in a large part of it. Moreover, one has convergence to this solution from an arbitrary starting point in this (part of the) domain.

(ii) *Semi-local convergence theorems*. In these theorems existence and uniqueness of a solution in a neighbourhood of the starting point, as well as convergence to this solution, are guaranteed, provided some, usually stringent, conditions are satisfied at the starting point and in its neighbourhood.

(iii) *Local convergence theorems*. Theorems in this class state that, assuming a solution x^* exists, there is a neighbourhood U of x^* such that for all starting points in U, the iterative process converges to x^*. Furthermore, results about the order of convergence of sequences generated by the process are considered to be local convergence results.

We shall treat these three classes of convergence results for Newton-like methods in the next three sections.

Throughout this chapter we use the following notation.

<u>5.1. NOTATION</u>. Let F be given, $(x,H) \in D \times L(\mathbb{R}^n)$ and $A \in L(\mathbb{R}^n)$. If F, x and A satisfy condition 1.10, with $\bar{\omega} > 0$, and $F(x) \neq 0$, then we denote

$$\beta(x) = \sup_{y \in S_F(x,A)} \|(J(y))^{-1}F(x)\|, \tag{5.1}$$

$$\alpha(x) = \bar{\omega}\beta(x), \tag{5.2}$$

$$\kappa(x) = \|AJ(x)\|\|(J(x))^{-1}F(x)\|/\|AF(x)\|, \tag{5.3}$$

$$e(x) = \|HJ(x) - I\|, \tag{5.4}$$

$$\nu_0(x) = \tfrac{1}{2}\kappa(x)(1+e(x)), \tag{5.5}$$

$$\nu_1(x) = (1+2e(x)+2(e(x))^2)/(1+e(x)), \tag{5.6}$$

$$\nu_2(x) = (1-e(x)\kappa(x))/\nu_0(x), \tag{5.7}$$

$$c(x)(t) = 1 + \nu_0(x)t\Big((\alpha(x)t)^2 + \nu_1(x)\alpha(x)t - \nu_2(x)\Big), \tag{5.8}$$

$$\zeta(x) = \Big(-\nu_1(x) + \sqrt{(\nu_1(x))^2+4\nu_2(x)}\Big)/(2\alpha(x)), \tag{5.9}$$

$$\mu(x) = \Big(-\nu_1(x) + \sqrt{(\nu_1(x))^2+3\nu_2(x)}\Big)/(3\alpha(x)). \tag{5.10}$$

For simplicity, we omit the dependence on H (e.g. in $e(x)$), since H always appears together with x, as an approximation to $(J(x))^{-1}$. If $(x_k,H_k) \in D \times L(\mathbb{R}^n)$ for some k, then we denote for short:

$$e_k = e(x_k),\ \beta_k = \beta(x_k),\ \alpha_k = \alpha(x_k),\ \kappa_k = \kappa(x_k),$$
$$\nu_{i,k} = \nu_i(x_k)\ (i=0,1,2),\ c_k = c(x_k),\ \zeta_k = \zeta(x_k),\ \mu_k = \mu(x_k),$$
$$J_k = J(x_k),\ F_k = F(x_k).$$

<u>5.2. REMARK</u>. $\beta(x)$ is finite by the existence and continuity of $(J(y))^{-1}$ $(y \in S_F(x,A))$ and the compactness of $S_F(x,A)$. As $\|(J(x))^{-1}\|$ is bounded and $\|AF(x)\| \leq \|AJ(x)\|\|(J(x))^{-1}F(x)\|$ we have

$$1 \leq \kappa(x) \leq \|AJ(x)\|\|(AJ(x))^{-1}\| < \infty.$$

For every $x \in D$, $c(x)(t)$ is a cubic function of t. If $\nu_2(x) > 0$ then $\zeta(x)$ is the real positive zero of the equation $c(x)(t) - 1 = 0$, and $\mu(x)$ is the point between 0 and $\zeta(x)$ where $c(x)(t) - 1$ attains its minimum.

The following condition wll be frequently used in this chapter and its notations will be used throughout.

5.3. CONDITION. Let $A \in L(\mathbb{R}^n)$, Ψ define a Newton-like process for F and $(x_0,H_0) \in D_\Psi$. Suppose

(i) F, x_0 and A satisfy condition 1.10 with $\bar{\omega} > 0$,

(ii) $D(\Psi_1) \supset \{(x,H) \mid (x,H) \in S_F(x_0,A) \times L(\mathbb{R}^n),$
$\exists t \in (0,1] : z = x-tHF(x) \in D,\ \|AF(z)\| \le \|AF(x)\|\}$,

(iii) $D(\Psi_2) \supset \{(x,H) \mid (x,H) \in D \times L(\mathbb{R}^n),\ \Psi_1(x,H) \in S\ (x_0,A)\}$,

(iv) Ψ generates for starting point (x_0,H_0) a sequence $\{(x_k,H_k)\}_{k=0}^{N}$, where N is the largest integer such that $(x_k,H_k) \in D_\Psi$, for all $k \le N$ (note that usually $N = \infty$),

(v) $F(x_k) \neq 0$ for all $k \le N$,

(vi) there exists a constant $\theta > 1$ such that for all $k \le N$:

$$e_k \le 1/(\theta\kappa_k).$$

In the next sections we give results about the convergence of Newton-like processes with starting points satisfying condition 5.3.

5.2. GLOBAL CONVERGENCE

5.2.1. Theoretical results

The results given in this section are generalizations of results given by DEUFLHARD [1974a, 1974b] about Newton methods.

The following theorem provides the basis for the global convergence results.

5.4. THEOREM. *Let be given* $F,(x,H) \in D \times L(\mathbb{R}^n)$ *and* $A \in L(\mathbb{R}^n)$. *Suppose* $F(x) \neq 0$ *and* F,x *and* A *satisfy condition* 1.10 *with* $\bar{\omega} > 0$. *Moreover, suppose that there exists a constant* $\theta > 1$ *such that*

(5.11) $\theta e(x) < 1/\kappa(x)$.

Denote for $t \in [0,1]$: $z(t) = x - tHF(x)$. *Then, for all* t *satisfying*

(5.12) $0 \le t \le \min(1,\zeta(x))$,

we have $z(t) \in S_F(x,A)$ *and*

(5.13) $\|AF(z(t))\| \le c(x)(t)\|AF(x)\|$.

5.5. REMARK. From the definition of $c(x)(t)$ (see (5.8)) we see that

$$c(x)(0) = 1; \quad \frac{d}{dt}[(c(x)(t))^2]_{t=0} = -2(1-\kappa(x)e(x)).$$

Comparing this with (4.29) in lemma 4.19 we see that the value of the derivative of $(c(x)(t))^2$ at $r = 0$ equals the upper bound on the derivative of $\|AF(z(t))\|^2/\|AF(x)\|^2$. Moreover, this bound is sharp according to lemma 4.19. Thus, theorem 5.4 states that $\|AF(z(t))\|/\|AF(x)\|$ can be bounded above by a cubic function $c(x)(t)$ which is equal to 1 for $t = 0$ and has "the best possible" derivative at $t = 0$. Therefore, (5.13) gives a good upper bound for small $e(x)$ and t.

PROOF of theorem 5.4.
For short we denote $S = S_F(x,A)$. By theorem 2.8 there exists a unique continuous function $p : [0,1] \to S$ satisfying, for $t \in [0,1]$,

(5.14) $\quad p'(t) = -(J(p(t)))^{-1}F(x), \quad p(0) = x$

and

(5.15) $\quad F(p(t)) = (1-t)(x).$

Define, for $t \in [0,1]$, $s \in [0,1]$:

(5.16) $\quad w(t,s) = p(t) + s(z(t) - p(t))$

and

(5.17) $\quad \delta(t) = \sup\{s | s \in [0,1];\ w(t,s') \in S \text{ for all } s' \in [0,s]\}.$

For all $t \in [0,1]$, we have $\delta(t) \neq 0$. This follows for $t \in (0,1]$ from (5.15) and the continuity of F. If $t = 0$ then $w(0,s) = x$ for all $s \in [0,1]$ and therefore $\delta(0) = 1$. Now define

(5.18) $\quad S_0 = \{x | x=w(t,s),\ t \in [0,1],\ s \in [0,\delta(t)]\}.$

By (5.17) $w(t,s) \in S$ for all $t \in [0,1]$ and $s \in [0,\delta(t))$. Therefore, the compactness of S yields $w(t,\delta(t)) \in S$ for all $t \in [0,1]$ and hence $S_0 \subset S$.

Now choose $t \in [0,1]$. As $J(x)$ is continuous on S_0 we can apply the mean value theorem for $s \in (0,\delta(t)]$ yielding:

$$F(w(t,s)) = F(p(t)) + \frac{1}{s}\left(\int_0^s J(w(t,s'))ds'\right)(w(t,s) - p(t)).$$

Using (5.15) and (5.16) yields

$$AF(w(t,s)) = (1-t)AF(x) + AJ(x)\left(\int_0^s (J(x))^{-1}J(w(t,s'))ds'\right)(w(t,s)-p(t))$$

$$= (1-t)AF(x) + sAJ(x)(w(t,s)-p(t))$$

$$+ AJ(x)\int_0^s ((J(x))^{-1}J(w(t,s')) - I)(w(t,s) - p(t))ds'.$$

Hence, taking norms on both sides and using the triangle inequality gives

$$\|AF(w(t,s))\| \leq (1-t)\|AF(x)\| + \|AJ(x)\|\|w(t,s)-p(t)\|$$

(5.19)
$$+ \|AJ(x)\|\int_0^s \|((J(x))^{-1}J(w(t,s'))-I)(w(t,s)-p(t))\|ds'.$$

We can use condition 1.10 to bound the integrand, so that for $s \in (0,\delta(t)]$:

(5.20) $$\|AF(w(t,s))\| \le (1-t)\|AF(x)\| + \|AJ(x)\|\left(1 + \frac{\bar{\omega}}{s}\int_0^s \|w(t,s')-x\|ds'\right)\|w(t,s)-p(t)\|.$$

We shall prove that for all $s \in (0,\delta(t)]$:

(5.21) $$\|w(t,s)-p(t)\| \le ts(\tfrac{1}{2}\alpha(x)t+e(x))\|(J(x))^{-1}F(x)\|$$

and

(5.22) $$\int_0^s \|w(t,s') - x\|ds' \le s\beta(x)(1+e(x))t.$$

Then, using $s \le 1$, it follows easily from (5.20) that

(5.23) $$\|AF(w(t,s))\| \le c(x)(t)\|AF(x)\|.$$

a. <u>PROOF of (5.21)</u>.

Condition 1.10 and (5.14) yield for $t \in [0,1]$:

(5.24) $$\begin{aligned}\|p'(t)-p'(0)\| &\le \|((J(p(t)))^{-1}J(p(0))-I)(J(p(0)))^{-1}F(x)\| \\ &\le \bar{\omega}\|p(t)-p(0)\|\|(J(x))^{-1}F(x)\|.\end{aligned}$$

Use of lemma 1.14 and (5.1) gives

(5.25) $$\|p(t)-p(0)\| \le t\beta(x).$$

Hence
$$\|p'(t)-p'(0)\| \le \alpha(x)t\|(J(x))^{-1}F(x)\|.$$

So lemma 1.15 can be applied to function $p(t)$ yielding

(5.26) $$\|p(t)-p(0)-tp'(0)\| \le \tfrac{1}{2}\alpha(x)\|(J(x))^{-1}F(x)\|t^2$$

for all $t \in [0,1]$. For $s \in (0,\delta(t)]$ we have

(5.27) $$w(t,s)-p(t) = -s(p(t)-p(0)-tp'(0))+ts((J(x))^{-1}-H)F(x).$$

Then (5.21) follows from (5.27) by using (5.26) and (5.4).

b. PROOF of (5.22).

By the definition of w (cf. (5.16)) we have

$$w(t,s)-x = (1-s)(p(t)-p(0))-st(J(x))^{-1}F(x)-st(H-(J(x))^{-1})F(x).$$

Use of (5.25), (5.1) and (5.4) yields for $s \in (0,\delta(t)]$:

$$\|w(t,s)-x\| \le \beta(x)(1+se(x))t \le \beta(x)(1+e(x))t.$$

So that (5.22) follows easily.

Hence (5.23) holds for $t \in [0,1]$ and $s \in [0,\delta(t)]$. As $\nu_2(x)$ is positive because of (5.11), $\zeta(x)$ is the positive zero of the equation $c(x)(t)-1 = 0$. So we only have to prove yet, that $\delta(t) = 1$ for $t \in [0,\min(1,\zeta(x))]$, so that we can choose $s = 1$ in (5.23), which then reduces to (5.13). Therefore, suppose $\delta(t^*) < 1$ for some $t^* \in (0,1)$ with $t^* < \zeta(x)$. Then, $c(x)(t^*) < 1$ and by (5.23)

$$\|AF(w(t^*,s))\| < \|AF(x)\| \quad (\text{for } 0 \le s \le \delta(t^*)).$$

Consequently, $w(t^*,\delta(t^*)) \in \text{int}(S)$. By theorem 2.2 and condition 1.8, F is a local homeomorphism at each point of int(S). Hence, there exists an open neighbourhood $U(w(t^*,\delta(t^*)),\bar\delta)$ $(\bar\delta \le 1)$ such that $\|AF(z)\| < \|AF(x)\|$ (thus $z \in S$), for all $z \in U(w(t^*,\delta(t^*)),\bar\delta)$. Therefore $w(t^*,s) \in \text{int}(S)$ for all $s < \delta(t^*)+\bar\delta$, but this contradicts the definition of $\delta(t)$ in (5.17). Hence $\delta(t) = 1$ for all $t \in (0,1)$ with $t < \zeta(x)$. As $w(t,1) \in S$ for $0< t< \min(\zeta(x),1)$ and S is compact, we have $\delta(t) = 1$ for $t \in [0,\min(1,\zeta(x))]$. □

5.6. REMARK. Condition 1.10 is, in fact, too strong. This condition states that the Lipschitz condition

$$\|(J(z))^{-1}J(y) - I\| \le \bar\omega\|z-y\|$$

has to be satisfied for all $z \in S_F(x,A)$ and all $y \in D$. As appears from (5.19) and (5.24) we need the condition that an $\omega(x) > 0$ exists such that

(5.28) $$\|((J(x))^{-1}J(w(t,s)) - I)(z(t)-p(t))\| \le \omega(x)\|x-w(t,s)\|\|z(t)-p(t)\|$$

and

(5.29) $$\|((J(p(t)))^{-1}J(x) - I)(J(x))^{-1}F(x)\| \le \omega(x)\|p(t)-x\|\|(J(x))^{-1}F(x)\|.$$

In fact an $\omega(x) \geq 0$ has to exist such that

$$\| ((J(y))^{-1}J(z) - I)u\| \leq \omega(x)\|y-z\|\|u\|,$$

with $y,z \in S_0$ (see (5.18)) and $u = (J(x))^{-1}F(x)$ or $u = x - tHF(x) - p(t)$. For clarity of exposition we use condition 1.10. However, as will be shown in chapter 6, the formulas (5.28) and (5.29) are of practical importance in computing an estimate of the value of $\omega(x)$, which in turn is used to estimate $\alpha(x)$, $\zeta(x)$ etc.

5.7. THEOREM. *Let the conditions of theorem* 5.4 *be satisfied and, moreover, let* τ *satisfy*

$$(5.30) \qquad 0 < \tau \leq \min(1, \frac{\theta-1}{4\alpha(x)(1+\theta\kappa(x))}).$$

Denote for $t \in [0,1] : z(t) = x-tHF(x)$. *Then* $\tau \leq \min(1,\zeta(x)-\tau)$, *and for all* t *with*

$$(5.31) \qquad \tau \leq t \leq \min(1,\zeta(x)-\tau)$$

we have $z(t) \in S_F(x,A)$ *and*

$$(5.32) \qquad \|AF(z(t))\| \leq (1 - \frac{\tau^2}{4}(1-1/\theta))\|AF(x)\|.$$

PROOF.

1. We first prove that $\tau \leq \min(1,\zeta(x)-\tau)$. Therefore observe that $\kappa(x) \geq 1$ (remark 5.2) and by (5.11)

$$\frac{2(\theta-1)}{1+\theta\kappa(x)} \leq \nu_2(x) \leq \frac{2}{\kappa(x)}.$$

Furthermore we use the inequality

$$-b + \sqrt{b^2+x} \geq \frac{x}{2b+\sqrt{x}}, \qquad \text{for } b,x > 0,$$

so as to get

$$\zeta(x) \geq \frac{\nu_2(x)}{\alpha(x)(\nu_1(x)+\sqrt{\nu_2(x)})}.$$

As $\nu_1(x)$ is increasing as a function of $e(x)$ on the interval $[0,1]$ we have $\nu_1(x) \le 5/2$. Hence

$$\nu_1(x) + \sqrt{\nu_2(x)} \le 5/2 + \sqrt{2} < 4.$$

So

$$\zeta(x) \ge \frac{\nu_2(x)}{4\alpha(x)} \ge \frac{\theta-1}{2\alpha(x)(1+\theta\kappa(x))} . \tag{5.33}$$

Hence, by (5.30), $\zeta(x)-\tau \ge \tau$ and $\tau \le \min(1,\zeta(x)-\tau)$.

2. To prove (5.32) we show that

$$c(x)(t) \le (1 - \frac{\tau^2}{4}(1-1/\theta)), \tag{5.34}$$

so that application of theorem 5.4 yields the result.
Let $\bar{\zeta} = \min(1,\zeta(x)-\tau)$ and define

$$q(t) = (\alpha(x)t)^2 + \nu_1(x)\alpha(x)t - \nu_2(x).$$

Then $q(\zeta(x)) = 0$ and $q(t)$ is increasing and negative for $0 \le t < \min(1,\zeta(x))$. Therefore, for $t \in [\tau,\bar{\zeta}]$,

$$c(x)(t) \le 1+\nu_0(x)\tau q(\bar{\zeta}). \tag{5.35}$$

We shall prove

$$q(\bar{\zeta}) \le -\frac{\tau}{4}\nu_2(x) \tag{5.36}$$

so that (5.34) follows from (5.35) by using (5.36) and $\nu_0(x)\cdot\nu_2(x) =$ $= (1-e\kappa(x)) \ge (1-1/\theta)$ (by (5.11)).

<u>PROOF of (5.36)</u>.
Observe that, with $\Delta = \zeta(x)-\bar{\zeta}$ and using $q(\zeta(x)) = 0$,

$$q(\bar{\zeta}) = -\alpha(x)\Delta(2\alpha(x)\zeta(x)-\alpha(x)\Delta+\nu_1(x)).$$

Therefore, with $\sigma = \sqrt{(\nu_1(x))^2+4\nu_2(x)}$,

$$q(\bar{\zeta}) = \alpha(x)\Delta(\alpha(x)\Delta-\sigma). \tag{5.37}$$

We distinguish between two cases.

a. $\bar{\zeta} < 1$.

Then

$$\Delta = \tau \le \tfrac{1}{2}\zeta(x) = \frac{1}{4\alpha(x)}(-\nu_1(x)+\sigma),$$

thus

$$\alpha(x)\Delta-\sigma \le -\tfrac{1}{4}(\nu_1(x)+3\sigma) < -\tfrac{1}{4}(\nu_1(x)+\sigma).$$

Furthermore, from $\zeta(x) = \bar{\zeta}+\Delta = \bar{\zeta}+\tau \le 2$, we obtain from the definition of $\zeta(x)$:

$$\alpha(x) \ge \tfrac{1}{4}(-\nu_1(x)+\sigma).$$

Using these inequalities to bound the right hand side of (5.37)

$$q(\bar{\zeta}) \le -\frac{\tau}{16}(\sigma^2-(\nu_1(x))^2) = -\frac{\tau}{4}\nu_2(x).$$

So (5.36) is proven for $\bar{\zeta} < 1$.

b. $\bar{\zeta} = 1$.

The definition of $\zeta(x)$ yields

$$1 = \bar{\zeta} = \zeta(x)-\Delta = \frac{1}{2\alpha(x)}(-\nu_1(x)+\sigma)-\Delta.$$

So

$$\alpha(x)\Delta = \frac{\Delta}{2(1+\Delta)}(-\nu_1(x)+\sigma).$$

Consequently

$$\alpha(x)\Delta-\sigma \le \tfrac{1}{2}(-\nu_1(x)+\sigma)-\sigma = -\tfrac{1}{2}(\nu_1(x)+\sigma).$$

Since $t/(1+t)$ is isotone for $t \in [0,\infty)$.

We also have

$$\alpha(x)\Delta \ge \frac{-\tau}{2(1+\tau)}(\nu_1(x)-\sigma).$$

Using these inequalities to bound the right hand side of (5.37) yields

$$q(\bar{\zeta}) \le \frac{\tau}{4(1+\tau)}((\nu_1(x))^2-\sigma^2) = -\nu_2(x)\frac{\tau}{1+\tau} .$$

So

$$q(\bar{\zeta}) \le -\frac{\tau}{2}\nu_2(x) < -\frac{\tau}{4}\nu_2(x),$$

which proves (5.36) for $\bar{\zeta} = 1$.

Therefore (5.36) holds, which completes the proof of the theorem. □

Using theorem 5.7 we can give the following global convergence theorem for Newton-like methods.

5.8. THEOREM. *Suppose that condition* 5.3 *is satisfied. Let* τ *be a constant satisfying for all* $k \le N$

$$(5.38) \qquad 0 < \tau \le \min(1, \frac{\theta-1}{4\alpha_k(1+\theta\kappa_k)}).$$

Suppose that Ψ *satisfies, for all* $k \le N$,

$$(5.39) \qquad \tau \le \lambda(x_k,H_k) \le \min(1,\zeta_k-\tau).$$

Then (x_k,H_k) *is defined for* $k = 0,1,2,\ldots$ $(N = \infty)$, $\{x_k\} \subset S_F(x_0,A)$ *and* $\{x_k\}$ *converges to a unique point* $x^* \in S_F(x_0,A)$ *with* $F(x^*) = 0$. *Moreover, there exists an integer* $K \ge 0$ *such that* $\lambda(x_k,H_k) = 1$ *satisfies* (5.39) *for all* $k \ge K$.

PROOF. Suppose (x_k,H_k) is defined for some k and $x_k \in S_F(x_0,A)$ (this holds for $k = 0$). Then $S_F(x_k,A) \subset S_F(x_0,A)$. Moreover, $S_F(x_k,A)$ is compact by definition and the compactness of $S_F(x_0,A)$. Hence F,x_k and A satisfy condition 1.10 and we can apply theorem 5.7 yielding

$$(5.40) \qquad \begin{aligned} &x_k - tH_kF_k \in S_F(x_k,A) \\ &\|AF(x_k-tH_kF_k)\| \le (1-\frac{\tau^2}{4}(1-1/\theta))\|AF_k\| \end{aligned}$$

for t such that $\tau \le t \le \min(1,\zeta_k-\tau)$. Therefore, by condition 5.3 (ii), $(x_k,H_k) \in D(\Psi_1)$. By the choice of $\lambda(x_k,H_k)$ (cf. (5.39)) we conclude that (5.40) holds for $t = \lambda(x_k,H_k)$, so that $x_{k+1} \in S_F(x_k,A)$. Use of condition 5.3(iii) yields $(x_k,H_k) \in D(\Psi_2)$. Therefore, (x_{k+1},H_{k+1}) is defined and $x_{k+1} \in S_F(x_0,A)$. So, by induction (x_k,H_k) is well-defined and $x_k \in S_F(x_0,A)$ for all $k = 0,1,2,\ldots$ $(N = \infty)$. Substituting $t = \lambda(x_k,H_k)$ in (5.40) yields

$$(5.41) \qquad \|AF_{k+1}\| \le \left(1 - \frac{\tau^2}{4}(1-1/\theta)\right)\|AF_k\|.$$

Therefore, as τ and θ are independent of k,

$$\lim_{k\to\infty} \|AF(x_k)\| = 0.$$

By the compactness of $S_F(x_0,A)$ there exists a subsequence of $\{x_k\}$ which converges to a limit x^* say, with $x^* \in S_F(x_0,A)$ and $F(x^*) = 0$. Application of theorem 2.8 proves the uniqueness of x^*. Therefore, $\lim_{k\to\infty} x_k = x^*$. Finally, to prove the last statement note that α_k and κ_k are bounded for all k. Moreover, as $\{x_k\}$ converges to a solution of $F(x) = 0$ we have

$$(5.42) \qquad \lim_{k\to\infty} \alpha_k \le \bar{\omega} \lim_{k\to\infty} \beta_k = 0.$$

Hence, there is a $K \ge 0$ such that for all $k \ge K$

$$\zeta_k = \frac{1}{2\alpha_k}\left(-\nu_{1,k} + \sqrt{\nu_{1,k}^2 + 4\nu_{2,k}}\right) \ge 2,$$

as the expression between the parentheses is bounded away from zero (see inequalities at start of proof of theorem 5.7 and note that $\kappa_k \ge 1$ and κ_k is bounded for all k). Hence, for all $k \ge K$, $\zeta_k - \tau \ge 1$ and therefore $\lambda(x_k,H_k) = 1$ satisfies (5.39). □

5.9. REMARK. The cubic function $c(x)(t)$ achieves its minimum for $t=\mu(x)$ on $[0,\zeta(x)]$. Therefore, by (5.13) we see that the choice

$$\lambda(x_k,H_k) = \min(1,\mu_k)$$

gives the best upper bound on the ratio $\|AF(x_{k+1})\|/\|AF(x_k)\|$.

Such a choice of $\lambda(x,H)$ yields a proper iterative process as is shown by the following theorem

5.10. THEOREM. *Suppose that condition* 5.3 (i), (iii), (iv), (v) *and* (vi) *are satisfied. Let* Ψ_1 *be such that* $\lambda(x,H) = \min(1,\mu(x))$, *for* $(x,H) \in D \times L(\mathbb{R}^n)$. *Then* (x_k,H_k) *is defined for* $k = 0,1,2,\ldots$ $(N=\infty)$, $\{x_k\} \subset S_F(x_0,A)$ *and* $\{x_k\}$ *converges to a unique point* $x^* \in S_F(x_0,A)$ *with* $F(x^*) = 0$. *Moreover, there exists an integer* $K \ge 0$ *such that* $\lambda(x_k,H_k) = 1$, *for all* $k \ge K$.

PROOF. If $(x,H) \in S_F(x_0,A) \times \mathcal{L}(\mathbb{R}^n)$ and there exists a $t \in (0,1]$ such that $z = x-tHF(x) \in D$ and $\|AF(z)\| \le \|AF(x)\|$, then $\lambda(x,H) = \min(1,\mu(x))$ is well-defined and hence $(x,H) \in D(\Psi_1)$. Therefore, condition 5.3 (ii) is satisfied for this choice of λ.

From the definition of ζ_k and μ_k we obtain for $k \le N$

$$\zeta_k - \mu_k \ge \frac{1}{6\alpha_k}\left(-\nu_{1,k} + \sqrt{\nu_{1,k}^2+4\nu_{2,k}}\right) \ge \tfrac{1}{3}\zeta_k.$$

Using (5.33) yields

$$\zeta_k - \mu_k \ge \frac{\theta-1}{6\alpha_k(1+\theta\kappa_k)} .$$

As α_k and κ_k are bounded for $k \le N$ by $\bar{\alpha}$ and $\bar{\kappa}$ say, we have

$$\zeta_k - \mu_k \ge \frac{\theta-1}{6\bar{\alpha}(1+\theta\bar{\kappa})} .$$

Therefore, the choice

$$\tau = \min(1, \frac{\theta-1}{6\bar{\alpha}(1+\theta\bar{\kappa})})$$

satisfies (5.38) and (5.39) for $\lambda(x_k,H_k) = \mu_k$. So we can apply theorem 5.8 yielding well-definedness of $\{(x_k,H_k)\}$ and convergence of $\{x_k\}$ to a unique solution in $S_F(x_0,A)$. The definition of μ_k and (5.42) yields the last statement of the theorem. □

5.11. REMARK. (5.41) shows that $\{\|AF_k\|\}$ converges at least linearly. Moreover, in each step we have a decrease with a factor which is less than or equal to $(1 - \frac{\tau^2}{4}(1-1/\theta))$. If e_k (the error in H_k as an approximation to J_k^{-1}) is relatively large so that we find a constant θ close to 1 with $e_k \le 1/(\theta\kappa_k)$ $(k=0,1,\ldots)$, then we can only quarantee a small decrease in $\|AF(x)\|$. However, if $e_k = 0$ (Newton method), then we can choose θ arbitrarily large, so that the decreasing factor becomes $1 - \tau^2/4$, with τ satisfying $0 < \tau \le \min(1,1/(4\alpha_k\kappa_k))$ for all k.

5.12. REMARK. Let M be a Newton-like method. Suppose that for all $F \in \mathcal{D}$, $\Psi = M(F)$ satisfies condition 5.3 with $\lambda(x,H) = \min(1,\mu(x))$. Moreover, suppose A in condition 5.3 satisfies (4.24) and Ψ_2 satisfies (4.8). Then M is affine invariant. This follows from the affine invariancy of the quantities given in notation 5.1 under the conditions. Note that, if A does not satisfy (4.24), then the quantities $\kappa(x)$ and $\beta(x)$ are not invariant under affine transformation of the function.

5.2.2. Applications

In this subsection we shall give applications of the convergence results from the last subsection. In particular corollaries of theorem 5.8 for the methods given in section 4.2 shall be given.

A. *Newton methods*

5.13. COROLLARY. *Let, for given* F, Ψ *define a Newton process. Suppose condition* 5.3 (i), (ii), (iv) *and* (v) *is satisfied. Let* τ *be a constant satisfying for all* $k \le N$

$$(5.43) \qquad 0 < \tau \le \min\left(1, \frac{1}{4\alpha_k\kappa_k}\right).$$

Suppose $H_0 = (J(x_0))^{-1}$ *and* Ψ *satisfies for all* $k \le N$

$$(5.44) \qquad \tau \le \lambda(x_k,H_k) \le \min(1,(-1+\sqrt{1+8/\kappa_k})/(2\alpha_k)-\tau).$$

Then (x_k,H_k) *is defined for* $k = 0,1,2,\ldots$ $(N=\infty)$, $\{x_k\} \subset S_F(x_0,A)$ *and* $\{x_k\}$ *converges to a unique point* $x^* \in S_F(x_0,A)$ *with* $F(x^*) = 0$. *Moreover, there exists an integer* $K \ge 0$ *such that* $\lambda(x_k,H_k) = 1$ *satisfies* (5.44) *for all* $k \ge K$.

PROOF. Condition 5.3 (iii) is satisfied by a Newton process as $J(x)$ is nonsingular for $x \in S_F(x_0,A)$. Furthermore, the choice of Ψ_2 and H_0 yields $e_k = 0$ for $k \le N$. Hence condition 5.3 (vi) is satisfied for all θ, $1 < \theta < \infty$. Therefore we can apply theorem 5.8 for arbitrary large θ. As

$$\lim_{\theta\to\infty} \frac{\theta-1}{4\alpha_k(1+\theta\kappa_k)} = \frac{1}{4\alpha_k\kappa_k}$$

and for $e_k = 0$:

$$\zeta_k = (-1+\sqrt{1+8/\kappa_k})/(2\alpha_k)$$

the result follows. □

Corollary 5.13 is given by DEUFLHARD [1974b].

B. *Difference Newton methods*

5.14. COROLLARY. *Let, for given* F, Ψ *define a difference Newton process satisfying* (4.23) *for nonsingular* $A \in L(\mathbb{R}^n)$. *Let* $x_0 \in D$ *and* F, x_0 *and* A *satisfy condition* 1.8. *Suppose that* F *and* z *satisfy condition* 1.7 *on* D, *for all* $z \in S_F(x_0,A)$. *Furthermore, let condition* 5.3 (ii), (iv) *and* (v) *be satisfied. Define, for* $h_i = h_i(x)$ $(i=1,\dots,n)$ (*see definition* 4.8)

$$|h(x)| = \left(\sum_{j=1}^{n} (h_j(x))^2\right)^{\frac{1}{2}}. \tag{5.45}$$

Suppose that $U(x,|h(x)|) \subset D$ *for all* $x \in S_F(x_0,A)$ *and that for given* $\theta > 1$ *and all* $x \in S_F(x_0,A)$

$$|h(x)| < \frac{2}{\omega(x)(1+\theta\kappa(x))}\ . \tag{5.46}$$

Then $B(x_k)$ *is nonsingular for* $k \le N$. *Let* $H_0 = (B(x_0))^{-1}$ *and* τ *be a constant satisfying* (5.38) *for all* $k \le N$. *Define, for* $k \le N$, *with* $\sigma_k = \omega_k|h(x_k)|$,

$$\bar{\nu}_{0,k} = \frac{\kappa_k}{2-\sigma_k}\ , \quad \bar{\nu}_{1,k} = \frac{4+\sigma_k^2}{4-2\sigma_k}\ , \quad \bar{\nu}_{2,k} = \frac{2-\sigma_k(1+\kappa_k)}{\kappa_k}\ , \tag{5.47}$$

and $\bar{c}(x)(t)$ *and* $\bar{\zeta}(x)$ *similar to* $c(x)(t)$ *and* $\zeta(x)$ (*cf* (5.8) *and* (5.9)) *with* $\nu_{i,k}$ *replaced by* $\bar{\nu}_{i,k}$ $(i=0,1,2)$. *Suppose* Ψ *satisfies for all* $k \le N$

$$\tau \le \lambda(x_k,H_k) \le \min(1,\bar{\zeta}(x_k)-\tau). \tag{5.48}$$

Then, the same conclusions as in corollary 5.13 *with* (5.44) *replaced by* (5.48) *hold.*

PROOF. Applying theorem 3.5 for $x \in S_F(x_0,A)$ and $U = U(x,|h(x)|)$ yields that $B(x)$ is well-defined, nonsingular and

$$e(x) \le \frac{\omega(x)|h(x)|}{2-\omega(x)|h(x)|}\ . \tag{5.49}$$

Note that this holds for $\delta \le |h(x)|$, as follows from the proof of theorem 3.4, and that $\omega(x)|h(x)| < 2$. Therefore condition 5.3 (iii) is satisfied. Furthermore, as (4.23) is satisfied, $x_k \in S_F(x_0,A)$ for $k \le N$, so that use of (5.46) in (5.49) yields condition 5.3 (vi). As the conditions on F, x_0 and A are stronger than condition 5.3 (i), condition 5.3 is satisfied and theorem 5.8 can be applied. Now observe that ζ_k is antitone with respect to e_k and that $\bar{\nu}_{i,k}$ (i=0,1,2) is obtained by replacing in the formulas for $\nu_{i,k}$ (i=0,1,2) the upper bound $\sigma_k/(2-\sigma_k)$ for e_k. Therefore, theorem 5.8 still holds with replacing $\nu_{i,k}$, c_k and ζ_k by $\bar{\nu}_{i,k}$, $\bar{c}_k(t)$ and $\bar{\zeta}_k$, respectively (i=0,1,2). This completes the proof. □

C. *Updating Newton Methods*

Application of theorem 5.8 to updating methods runs into the following two problems.

1. The definition of $D(\Psi_2)$ in definition 4.11 is such that condition 5.3 (iii) is not satisfied.
2. Using theorem 3.16 to bound the error e_k for some k, the given upper bound depends on e_{k-1}. Moreover, the upper bound on e_k is greater or equal to the upper bound on e_{k-1}. Therefore, doing so in each iteration step the upper bound on e_k increases in each step and will usually become larger than $1/\kappa_k$ for some k. Thus condition 5.3 (vi) cannot hold and application of theorem 5.8 is not possible. In fact, as condition 5.3 (vi) can not be proven with the results given so far, we cannot guarantee that a step length factor can be found such that the level function decreases at each step so that the algorithm may terminate after finitely many steps without finding a solution.

Nevertheless, we can imagine a useful application of approximation by updating based on theorems 3.16 and 5.8.

<u>5.15. COROLLARY</u>. *Let* Ψ *define a Newton-like process for* F *and suppose that condition* 5.3 *is satisfied. Define, for some fixed* $\theta > 1$, *a Newton-like process for* F *by* $\bar{\Psi}$ *so that, for* $(x,H) \in D_\Psi$,

$$\bar{\Psi}_1(x,H) = \Psi_1(x,H),$$

(5.50)
$$\bar{\Psi}_2(x,H) = \begin{cases} V(u)(H,x,\bar{\Psi}_1(x,H)), & \textit{if } (\Psi_1(x,H)-x)^T u \neq 0,\ H \textit{ is non-singular and } \bar{e}(\bar{\Psi}_1(x,H)) < 1/(\theta\kappa(\bar{\Psi}_1(x,H))), \\ \Psi_2(x,H), & \textit{otherwise,} \end{cases}$$

where

$$(5.51)\qquad \bar{e}(y) = (\rho_1 \frac{e(x)}{1-e(x)} + \omega(x)(\rho_1+\tfrac{3}{2}\rho_2)\|y-x\|)(1+e(x)),$$

$$\rho_1 = \frac{\|H(F(y)-F(x))\|\,\|u\|}{|u^T H(F(y)-F(x))|}, \qquad \rho_2 = \frac{\|y-x\|\,\|u\|}{|u^T H(F(y)-F(x))|}$$

and u *is some vector (see theorem* 3.16 *and remark* 3.17). *If there exists a constant* τ *such that for all* k, *for which* (x_k,H_k) *is defined by* $\bar{\Psi}$, (5.38) *and* (5.39) *are satisfied, then the conclusions of theorem* 5.8 *hold for* $\bar{\Psi}$.

<u>PROOF</u>. Clearly, condition 5.3 (i), (ii), (iv) and (v) hold by assumption. By the conditions for use of inverse-updating and the condition on Ψ it follows easily that condition 5.3 (iii) is satisfied. Moreover, the condition on the error in the update approximations assures that (vi) is also satisfied if inverse-updating is used, as by theorem 3.16 $\bar{e}(\bar{\Psi}_1(x,H))$ is an upper bound on $e(\bar{\Psi}_1(x,H))$.

<u>5.16. REMARK</u>. In corollary 5.15 it is stated that one can modify, for instance, Newton methods or difference Newton methods which satisfy appropriate conditions, in such a way that in certain iteration steps approximation by inverse-updating is used. If the original Newton method yields global convergent processes based on theorem 5.8 then the modified method still yields global convergence under similar conditions. Note that the condition on $\bar{e}(\bar{\Psi}_1(x,H))$ guarantees that $-HF(x)$ is a descent direction for the level function $\|AF(x)\|^2$ at x. Of course, updating can be used instead of inverse-updating by replacing $\mathcal{V}(u)(H,x,\Psi_1(x,H))$ by $(\mathcal{U}(u)(H^{-1},x,\Psi_1(x,H)))^{-1}$ in (5.50).

D. *Fixed Newton methods*

As for updating methods, we cannot guarantee that condition 5.3 (vi) is satisfied if we use fixed approximation in every step of a Newton-like process. However, we can imagine a similar application of fixed approximation as given in corollary 5.15 for inverse-update approximation.

<u>5.17. COROLLARY</u>. *Let* Ψ *define a Newton-like process for* F *and suppose condition* 5.3 *is satisfied. Define, for some fixed* $\theta > 1$ *a Newton-like process for* F *by* $\bar{\Psi}$ *so that for* $(x,H) \in D_\Psi$:

$$\bar{\Psi}_1(x,H) = \Psi_1(x,H),$$

$$\bar{\Psi}_2(x,H) = \begin{cases} H, \; \textit{if } e(x)+\omega(x)(1+e(x))\|\bar{\Psi}_1(x,H)-x\| < 1/(\theta\kappa(x)) \\ \\ \Psi_2(x,H) \quad \textit{otherwise}. \end{cases} \tag{5.52}$$

(cf. theorem 3.19). *If there exists a constant* τ *such that for all* k, *for which* (x_k,H_k) *is defined by* $\bar{\Psi}$, (5.38) *and* (5.39) *are satisfied, then the conclusions of theorem* 5.8 *hold for* $\bar{\Psi}$.

PROOF. Condition 5.3 (i), (ii), (iv) and (v) hold by assumption and 5.3 (iii) is obvious. By theorem 3.19 and the condition for choosing $\bar{\Psi}_2(x,H) = H$, it is easily verified that condition 5.3 (vi) is also satisfied. Hence theorem 5.8 can be applied to $\bar{\Psi}$. □

E. *Use of generalized inverse*

Under the conditions of theorem 5.8 we know that J(x) is nonsingular for $x \in D$. Therefore, $(J(x))^+ = (J(x))^{-1}$ for $x \in D$ and the generalized Newton method generates the same processes as the Newton method for the same function. Hence corollary 5.13 is applicable. If we consider the generalized difference Newton method, then the conditions of corollary 5.14 also quarantee nonsingularity of $B(x_k)$. So under these conditions the generalized difference Newton method generates identical processes as the difference Newton method.

Obviously we are particularly interested in application of the generalized inverse if singular jacobian matrices occur in $S_F(x_0,A)$. Let S_0 denote the set of points in D for which the jacobian matrix is singular (usually S_0 is an (n-1)-dimensional manifold). Suppose $S_0 \cap S_F(x_0,A) \neq \emptyset$. Then the question is whether application of the generalized inverse will yield an iterate x_K for some integer K, such that $S_0 \cap S_F(x_K,A) = \emptyset$. If this is true then corollary 5.13 can be applied subsequently if the Newton-like method satisfies the conditions. However, with the results given in foregoing chapters (e.g. lemma 4.26) we can only prove that $\|AF(x_0)\| > \|AF(x_1)\| > \ldots > \|AF(x_K)\|$. In fact, the sequence $\{x_k\}$ may converge to a point $\bar{x}$ with $S_0 \cap S_F(\bar{x},A) \neq \emptyset$ (e.g. $\bar{x} \in S_0$).

From optimization theory we know that it might be useful to do a so-called "steepest descent" iteration step, which in our case with $\|F(x)\|^2$ $(A = I)$ to be optimized, turns out to be an iteration step of the form

$$x_{k+1} = x_k - \lambda_k (J(x_k))^T F(x_k),$$

with $\lambda_k > 0$ a scalar to be chosen in some specific way. It is interesting to note that both directions of search $(J(x_k))^T F(x_k)$ and $(J(x_k))^+ F(x_k)$ lie in the same subspace orthogonal to $\ker(J(x_k))$. This follows easily by using the singular value decomposition of $J(x_k) = U_1 \Sigma_r V_1^T$ (cf. 1.18)). Then

$$(J(x_k))^T F(x_k) = V_1 \Sigma_r U_1^T F(x_k),$$

$$(J(x_k))^+ F(x_k) = V_1 \Sigma_r^+ U_1^T F(x_k).$$

In fact, if all nonzero singular values are equal then these directions are the same (not the lengths).

In BRANIN [1972] another strategy is given for handling singular jacobian matrices. This strategy uses an iteration step of the form

$$x_{k+1} = x_k \pm \lambda_k \operatorname{adj}(J(x_k)) F(x_k),$$

where $\operatorname{adj}(J(x_k))$ is the adjoint of $J(x_k)$ (which satisfies $J(x_k) \cdot \operatorname{adj}(J(x_k)) = \det(J(x_k)) \cdot I$). Again, this yields a direction in the same subspace, orthogonal to $\ker(J(x_k))$. An important point in this strategy is the choice of the sign, which is changed at a singularity. A drawback of the method of Branin is the fact that it only works well if $\operatorname{rank}(J(x_k)) \geq n-1$ as his computation of the adjoint is only defined in this case. An interesting question to be asked may be which relation exists between the method with use of generalized inverse and Branin's method.

5.3. SEMI-LOCAL CONVERGENCE

In this section we present a semi-local convergence result for strict Newton-like methods. It is a so-called Kantorovich-type result. A similar theorem, restricted to the strict Newton method is given in KANTOROVICH & AKILOW [1964, section XVIII] and ORTEGA & RHEINBOLDT [1970, section 12.6.2]. In the latter a generalization to a certain class of strict Newton-like methods is also given, which is based on condition 1.6. The class of Newton-like methods considered in their theorem satisfies, for some $B : \mathbb{R}^n \to L(\mathbb{R}^n)$,

$$\Psi_2(x,H) = (B(x))^{-1},$$

where for $x_0 \in D$ and constants c_1, c_2 and c_3:

$$\|B(x) - B(x_0)\| \le c_1\|x-x_0\|,$$
$$\|J(x) - B(x)\| \le c_2+c_3\|x-x_0\|,$$

for all x in a convex set in D. So $\Psi_2(x,H)$ does not depend on H and the conditions are not affine invariant. DEUFLHARD & HEINDL [1979] give a Kantorovich type theorem for a class of strict Newton-like methods which is based on the rather unnatural Lipschitz condition (with (x_0,H_0) starting point for the iterative process):

$$(5.53) \qquad \|H_0(J(x) - J(y))\| \le \omega\|x-y\|,$$

for some constant ω and x,y in some convex set in D. This condition is affine invariant, only if H_0 is chosen dependent on F such that $H_0(TF) = H_0(F)T^{-1}$ for arbitrary nonsingular $T \in L(\mathbb{R}^n)$. The theorem given here uses the affine invariant Lipschitz condition 1.7. All these theorems are proven by finding a nonlinear majorizing sequence satisfying the conditions of lemma 1.22.

<u>5.18. THEOREM</u>. *Let* F *and* $(x_0,H_0) \in D \times L(\mathbb{R}^n)$ *be given. Suppose* F *and* x_0 *satisfy condition* 1.7 *with* $\omega(x_0) \neq 0$ *on some convex set* $D_0 \subset D$ *with* $x_0 \in D_0$. *Let* M *be a strict Newton-like method and* $F \in \mathcal{D}$. *Suppose that* $\Psi = M(F)$ *satisfies*

$$D(\Psi_2) \supset \{(x,H) \mid (x,H) \in D \times L(\mathbb{R}^n),\ \Psi_1(x,H) \in D_0\}$$

and, for starting point (x_0,H_0), Ψ *generates a sequence* $\{(x_k,H_k)\}_{k=0}^{N}$. *Suppose* $F(x_k) \neq 0$ $(k \le N)$ *and there exists a constant* $e < -1+\sqrt{2}$ *such that* $e_k \le e$ $(k \le N)$. *Define*

$$(5.54) \qquad \chi_1 = \frac{1+e}{1-e}, \quad \chi_2 = \frac{1-2e-e^2}{1-e},$$

$$(5.55) \qquad \bar{\beta}_0 = \|H_0F_0\|, \quad \bar{\alpha} = \omega(x_0)\bar{\beta}_0\chi_1, \quad \bar{\zeta} = \frac{1}{\omega(x_0)\chi_1}\left(\chi_2-\sqrt{\chi_2^2-2\bar{\alpha}}\right).$$

Moreover, suppose $\overline{U(x_0,\bar{\zeta})} \subset D_0$ *and* $\bar{\alpha} < \frac{1}{2}\chi_2^2$. *Then*

1. (x_k,H_k) *exists for all* k $(N = \infty)$ *and* $\{x_k\} \subset U(x_0,\bar{\zeta})$,
2. $J(x_k)$ *is nonsingular for all* $k = 0,1,2,\ldots$,
3. $\{x_k\}$ *converges to a point* $x^* \in \overline{U(x_0,\bar{\zeta})}$ *with* $F(x^*) = 0$.
4. *Let, in addition,* F *and* x *satisfy condition 1.7 on* D_0 *for all* $x \in D_0$, *with* $\omega(x) < \tilde{\omega}$ *for some constant* $\tilde{\omega}$ *and denote*

$$\bar{\bar{\alpha}} = \frac{2(1+e_0)\tilde{\omega}\bar{\beta}_0}{1-e_0}, \quad \bar{\bar{\zeta}} = \frac{(1-e_0)}{2(1+e_0)\tilde{\omega}}\left(1+\sqrt{1-2\bar{\bar{\alpha}}}\right),$$

$$U_1 = U\left(x_0, \frac{1-e_0}{2(1+e_0)\tilde{\omega}}\left(1-\sqrt{1-2\bar{\bar{\alpha}}}\right)\right).$$

Then, $\bar{\bar{\alpha}} < \frac{1}{2}\chi_2^2$ *and* $\bar{U}_1 \subset D_0$ *implies that* x^* *is unique in* $D_0 \cap U(x_0,\bar{\bar{\zeta}}) \supset \overline{U(x_0,\bar{\zeta})}$.

<u>PROOF</u>. Notice that χ_1 is isotone and χ_2 is antitone with respect to e on $[0, -1+\sqrt{2})$ and

$$(5.56) \qquad 1 \le \chi_1 < \frac{1}{\sqrt{2}-1}, \quad 0 < \chi_2 \le 1 \quad \text{for } e \in [0, -1+\sqrt{2}).$$

By the assumption $\bar{\alpha} < \frac{1}{2}\chi_2^2$ we see that $\bar{\zeta}$ is real, $\bar{\zeta} \ge 0$. Consider $U(x_0,\bar{\zeta})$. Then for all $x \in U(x_0,\bar{\zeta})$ we have, with $J_0 = J(x_0)$,

$$\|J_0^{-1}J(x) - I\| \le \omega_0\|x_0-x\| < \omega_0\bar{\zeta},$$

where $\omega_0 = \omega(x_0)$. Furthermore

$$\omega_0\bar{\zeta} = \frac{1}{\chi_1}\left(\chi_2-\sqrt{\chi_2^2-2\bar{\alpha}}\right) \le \frac{\chi_2}{\chi_1} < 1.$$

Use of lemma 1.13 now yields that $J(x)$ is nonsingular for all $x \in U(x_0,\bar{\zeta})$ and

$$(5.57) \qquad \|(J(x))^{-1}J_0\| \le \frac{1}{1-\omega_0\|x_0-x\|}, \qquad x \in U(x_0,\bar{\zeta}).$$

As $\bar{\beta}_0 = \bar{\alpha}/(\omega_0\chi_1)$ and elementary calculation shows that $\bar{\alpha} < \chi_2-\sqrt{\chi_2^2-2\bar{\alpha}}$ for $0 < \bar{\alpha} < \frac{1}{2}\chi_2^2$ we conclude that $\bar{\beta}_0 < \bar{\zeta}$. Hence $\|x_1-x_0\| = \bar{\beta}_0 < \bar{\zeta}$, so that $x_1 \in U(x_0,\bar{\zeta})$. Then H_1 is defined due to the condition on $D(\Psi_2)$.

We shall prove by induction that (x_k,H_k) is defined for all $k=0,1,2,\ldots$ $(N = \infty)$ and $x_k \in U(x_0,\bar{\zeta})$. Therefore assume that, for certain integer K, (x_k,H_k) is defined and $x_k \in U(x_0,\bar{\zeta})$, for all $k \le K$. Then

$$(5.58) \qquad \|H_kF_k\| \le \|H_kJ_0\|\|J_0^{-1}F_k\|.$$

Using (5.57) and the upper bound on e we obtain

$$(5.59) \qquad \|H_kJ_0\| \le \|H_kJ_k\|\|J_k^{-1}J_0\| \le \frac{1+e}{1-\omega_0\|x_0-x_k\|}.$$

Furthermore, we can bound

$$\|J_0^{-1}F_k\| \le \|J_0^{-1}(F_k-F_{k-1}-J_{k-1}(x_k-x_{k-1}))\| + \|J_0^{-1}(F_{k-1}-J_{k-1}H_{k-1}F_{k-1})\|.$$

So, application of lemma 1.16 yields

$$\|J_0^{-1}F_k\| \le \tfrac{1}{2}\omega_0\|x_k-x_{k-1}\|^2 + \|J_0^{-1}J_{k-1}\|\|J_{k-1}^{-1}H_{k-1}^{-1} - I\|\|H_{k-1}F_{k-1}\|$$

$$\le \tfrac{1}{2}\omega_0\|x_k-x_{k-1}\|^2 + (1+\omega_0\|x_{k-1}-x_0\|)\frac{e}{1-e}\|x_k-x_{k-1}\|.$$

Therefore, with use of (5.59) in (5.58) we obtain

$$\|H_kF_k\| \le \frac{1+e}{1-\omega_0\|x_0-x_k\|}\left(\tfrac{1}{2}\omega_0\|x_k-x_{k-1}\|+\frac{e}{1-e}(1+\omega_0\|x_{k-1}-x_0\|)\right)\|x_k-x_{k-1}\|.$$

With the notation

$$p_1 = \tfrac{1}{2}\chi_1\omega_0, \quad p_2 = \chi_1 e, \quad p_3 = \omega_0 p_2 \text{ and } p_4 = (1+e)\omega_0$$

we have for all $k \le K$:

(5.60) $$\|H_kF_k\| \le \frac{1}{1-p_4\|x_k-x_0\|}(p_1\|x_k-x_{k-1}\|+p_2+p_3\|x_{k-1}-x_0\|)\|x_k-x_{k-1}\|.$$

Now define (cf. lemma 1.23) the sequence $\{t_k\}$ by $t_0 = 0$, $t_1 = \bar{\beta}_0$ and, for $k = 1,2,...,$

$$t_{k+1}-t_k = \frac{1}{1-p_4t_k}(p_1(t_k-t_{k-1})+p_2+p_3t_{k-1})(t_k-t_{k-1}).$$

Then $\{t_k\}$ satisfies the assumption of lemma 1.23. Thus $\{t_k\}$ is increasing, unless $F_k = 0$, and $\lim_{k\to\infty} t_k = \bar{\zeta}$. Moreover, by (5.60) we have for all $k < K$:

(5.61) $$\|x_{k+1}-x_k\| = \|H_kF_k\| \le t_{k+1}-t_k.$$

Hence

$$\|x_K-H_KF_K-x_0\| \le \sum_{k=0}^{K}\|H_kF_k\| \le \sum_{k=0}^{K} t_{k+1}-t_k \le t_{K+1}-t_0 < \bar{\zeta}.$$

So $x_{K+1} = x_K - H_KF_K$ is defined and belongs to $U(x_0,\bar{\zeta})$ and, hence, H_{K+1} is defined. Therefore (x_k,H_k) is defined for all $k = 0,1,2,...$ $(N = \infty)$ and $x_k \in U(x_0,\bar{\zeta})$. Moreover (5.61) holds for all k and $\lim_{k\to\infty} t_k = \bar{\zeta}$. ($\{t_k\}$ is a majorizing sequence for $\{x_k\}$). Application of lemma 1.22 yields convergence of $\{x_k\}$ to some $x^* \in \overline{U(x_0,\bar{\zeta})}$. By the nonsingularity of H_k we have

$$\|F_k\| \le \|H_k^{-1}J_k^{-1}\|\|J_k(x_{k+1}-x_k)\| \le \frac{1}{1-e}\|J_k\|\|x_{k+1}-x_k\|.$$

So, convergence of $\{x_k\}$ to x^* implies $F(x^*) = 0$, which yields the third statement of the theorem.

To prove the last statement, consider the fixed Newton-like process defined by $\bar{\Psi}$:

$$\bar{\Psi}_1(x,H) = x - H_0F(x), \quad \bar{\Psi}_2(x,H) = H_0, \quad (x,H) \in D_0 \times L(\mathbb{R}^n).$$

The sequence $\{x_k\}$ generated by this process for starting point (x_0,H_0) is also generated by the simple iterative process defined by the iteration function

$$\Phi(x) = x - H_0F(x), \quad x \in D_0.$$

We see that Φ is differentiable on D_0:

$$\Phi'(x) = I - H_0 J(x).$$

Furthermore, for $x,y \in D_0$, we have

$$\|\Phi'(x)-\Phi'(y)\| \le \|H_0(J(x)-J(y))\| \le \|H_0 J(x)\| \, \|(J(x))^{-1}J(y) - I\|.$$

As $\omega_0\bar{\bar{\zeta}} = \tilde{\omega}\bar{\bar{\zeta}} < 1$ we obtain for $x,y \in D_0 \cap U(x_0,\bar{\bar{\zeta}})$:

$$\|H_0J(x)\| \le \|H_0J(x_0)\| \, \|(J(x_0))^{-1}J(x)\| \le (1+e_0)(1+\omega_0\|x-x_0\|) < 2(1+e_0)$$

and

$$\|\Phi'(x) - \Phi'(y)\| \le 2(1+e_0)\tilde{\omega}\|y-x\|.$$

Moreover, $\|\Phi'(x_0)\| \le e_0 \le e < 1$ and $\|x_0 - \Phi(x_0)\| = \|H_0F_0\| = \bar{\beta}_0$. So we can apply theorem 1.24 yielding existence of a sequence $\{x_k\} \subset U_1$ converging to a solution y^* of $F(x) = 0$ which is unique in $U(x_0,\bar{\zeta}) \cap D_0$. We shall show finally that $\bar{\zeta} < \bar{\bar{\zeta}}$, so that $x^* = y^*$ and statement 4 is proven. To do so, observe that $1 - \sqrt{1-x} \le x$, for $0 \le x \le 1$. Then

$$\bar{\zeta} = \frac{\chi_2}{\omega_0\chi_1}\left(1 - \sqrt{1 - \frac{2\bar{\alpha}}{\chi_2^2}}\right) \le \frac{2\bar{\alpha}}{\omega_0\chi_1\chi_2} = \frac{2\bar{\beta}_0}{\chi_2}$$

and

$$\bar{\bar{\zeta}} = \frac{1-e_0}{2(1+e_0)\tilde{\omega}}\,(1 + \sqrt{1-2\bar{\bar{\alpha}}}) \ge \frac{(1-e_0)(1-\bar{\bar{\alpha}})}{(1+e_0)\tilde{\omega}} \ge 2\bar{\beta}_0\left(\frac{1}{\bar{\bar{\alpha}}} - 1\right).$$

As $\bar{\bar{\alpha}} < \frac{1}{2}\chi_2^2$ and $\chi_2 \le 1$ we have

$$\bar{\bar{\zeta}} > \frac{2\bar{\beta}_0}{\chi_2}\left(\frac{1 - \frac{1}{2}\chi_2^2}{\frac{1}{2}\chi_2}\right) \ge \frac{2\bar{\beta}_0}{\chi_2} \ge \bar{\zeta}.$$

This completes the proof of the theorem. □

<u>5.19. REMARK</u>. Note that for $e = 0$ we have $\chi_1 = \chi_2 = 1$, $\bar{\alpha} = \omega_0\bar{\beta}_0$, $\bar{\zeta} = \frac{1}{\omega_0}(1 - \sqrt{1-2\bar{\alpha}})$. This expression for $\bar{\zeta}$ also appears in comparable theorems of ORTEGA & RHEINBOLDT [1970] and DEUFLHARD & HEINDL [1979].

5.4. LOCAL CONVERGENCE

In this section we give a general result on the order of convergence of sequences of approximations to a solution which are generated by strict Newton-like processes. This result is a generalization of a well known result for Newton methods (see for instance ORTEGA & RHEINBOLDT [1970, section 10.22]). DENNIS & MORE [1977] give a result for a class of Newton-like methods, using a non-affine invariant Lipschitz condition, which shows that we do not require convergence of H_kJ_k to I, but only of H_kJ_k to I with respect to its effect on a relevant direction (they use $x_{k+1}-x_k$). This idea is incorporated in our result. The second theorem in this section gives a result for restrained Newton-like methods by combining global and local convergence results.

5.20. THEOREM. *Let* $x^* \in D$, $F(x^*) = 0$ *and suppose that* F *and* x^* *satisfy condition* 1.7 *on some open neighbourhood* $U \subset D$ *of* x^*. *Let* $(x_0,H_0) \in U \times L(\mathbb{R}^n)$ *be given and let* M *be a strict Newton-like method such that* $F \in \mathcal{D}$ *and* (x_0,H_0) *lies in the domain of* $M(F)$. *Suppose* $M(F)$ *generates* $\{(x_k,H_k)\} \subset U$ *for starting point* (x_0,H_0). *Moreover, suppose that* $\{x_k\}$ *converges to* x^*, $x_k \neq x^*$ $(k=0,1,\ldots)$ *and there exist constants* $e,E \geq 0$ *such that for all* k, $e_k \leq E$ *and*

$$\|(H_kJ_k-I)(x_k-x^*)\| \leq e\|x_k-x^*\|^2. \tag{5.62}$$

Then, the order of convergence of $\{x_k\}$ *is at least* 2.

PROOF. Denote $\omega^* = \omega(x^*)$ and choose $\delta \leq \min(1/2\omega^*,1/2e)$ such that $U(x^*,\delta) \subset U$. Then, by condition 1.7, we have for $x \in U(x^*,\delta)$:

$$\|(J(x^*))^{-1}J(x) - I\| \leq \omega^*\|x^*-x\| < 1.$$

Hence, by the perturbation lemma $J(x)$ is nonsingular for $x \in U(x^*,\delta)$ and

$$\|(J(x))^{-1}J(x^*) - I\| \leq \frac{\omega^*\|x-x^*\|}{1-\omega^*\|x-x^*\|} < 2\omega^*\|x-x^*\|. \tag{5.63}$$

As $\{x_k\}$ converges to x^*, we can choose a $K \geq 0$ such that for all $k \geq K$, $x_k \in U(x^*,\delta)$. Hence, for $k \geq K$ we have

$$\|x_{k+1}-x^*\| = \|-H_kF_k+x_k-x^*\|$$

(5.64)
$$\le \|H_kJ_k\|\|J_k^{-1}J(x^*)\|\|(J(x^*))^{-1}(F_k-F(x^*)-J(x^*)(x_k-x^*))\|$$
$$+ \|(I-H_kJ(x^*))(x_k-x^*)\|.$$

and

$$\|(I-H_kJ(x^*))(x_k-x^*)\|$$
$$\le \|(I-H_kJ_k)(x_k-x^*)\| + \|H_kJ_k\|\|I-J_k^{-1}J(x^*)\|\|x_k-x^*\|.$$

So, using lemma 1.16 to bound the last factor of the first term of the right hand side of (5.64) and (5.62) and (5.63) to bound the other factors and the second term, we obtain

$$\|x_{k+1}-x^*\| \le \tfrac{1}{2}\omega^*(1+E)(1+2\omega^*\|x_k-x^*\|)\|x_k-x^*\|^2$$
$$+ e\|x_k-x^*\|^2+2\omega^*(1+E)\|x_k-x^*\|^2.$$

With the definition of δ, we obtain finally

(5.65) $$\|x_{k+1}-x^*\| \le C\|x_k-x^*\|^2,$$

where

$$C = 3\omega^*(1+E) + e$$

Therefore

$$\liminf_{k\to\infty}(-\log\|x_k-x^*\|)^{1/k} \ge 2$$

which proves the theorem. □

5.21. REMARK. If we consider the strict Newton method, then we can choose $e = E = 0$ and $C = 3\omega^*$ for the given choice of δ. In this case the theorem reduces to a well-known result (see ORTEGA & RHEINBOLDT [1970, section 10.2.2]) reformulated for the affine invariant Lipschitz condition on the jacobian. DEUFLHARD & HEINDL [1979] also give a version of theorem 5.20 for the strict Newton method. They use condition (5.53).

5.22. REMARK. Condition (5.62) is of practical importance. If we had only required that $e_k \le e\|x_k-x^*\|$ for $k = 0,1,\ldots,$ then the theorem would not have been applicable in certain situations. For instance, using an updating

Newton method it might appear that $\{x_k\}$ lies in a certain subspace of $\mathbb{R}^n$, so that H_kJ_k will not converge to I with respect to its effect on the complement of this subspace. Then $e_k < e\|x_k-x^*\|$ $(k=0,1,\ldots)$ is not satisfied while (5.62) might be.

Finally we give a theorem which combines the results of theorem 5.8 and 5.20.

5.23. THEOREM. *Let the conditions of theorem* 5.8 *be satisfied. Moreover, suppose that* $e_k \le e\|x^*-x_k\|$ *for* $k \le N$ *and some constant* $e \ge 0$ *and that* $\lambda(x_k,H_k) = 1$ *whenever this choice satisfies* (5.39). *Then*

1. (x_k,H_k) *is defined for* $k = 0,1,2,\ldots$ $(N = \infty)$,
2. $\{x_k\}$ *converges to a unique point* $x^* \in S_F(x_0,A)$ *with* $F(x^*) = 0$,
3. *the order of convergence of* $\{x_k\}$ *is at least* 2.

PROOF. Statements 1 and 2 follow directly from theorem 5.8. Moreover, it follows from theorem 5.8 that there exists a $K \ge 0$ such that $\lambda(x_k,H_k) = 1$ satisfies (5.39) for $k \ge K$. Hence, for all $k \ge K$, we have $\lambda(x_k,H_k) = 1$ by assumption. Now apply theorem 5.20 for the same process with starting point (x_K,H_K). This proves the third statement. □

CHAPTER 6

SYNTHESIS OF NEWTON-LIKE METHODS

6.1. INTRODUCTION

A Newton-like method applied to a given function yields a Newton-like process. Such a process generates for a given starting point (x_0,H_0) a, possibly infinite, sequence of iterates. In practice we are interested in solving systems of nonlinear equations with a certain precision and in finitely many iteration steps. Furthermore, the choice of the matrix H_0 will be based on the initial guess x_0 of the solution and on the method used.

6.1. TERMINOLOGY. We use the term *Newton-like algorithm* for a Newton-like method together with the choice of H_0 and a stopping criterion, described in some formal way and assuming that non-exact arithmetic is used.
A *Newton-like program* is an implementation of a Newton-like algorithm in a given programming language and for a given computer.

Note that a Newton-like program defines a Newton-like algorithm but not vice versa. The formal description of an algorithm does not have to be in a programming language and the values of computer dependent constants are not actually given in the algorithms.

Considering Newton-like algorithms we distinguish the following basic modules

1. *Initialization*, including the calculation of H_0 as an approximation to $(J(x_0))^{-1}$ (or an approximation B_0 to $J(x_0)$).
2. *Approximation of required data*, such as $\bar{\omega}$, κ_k, e_k etc. (see notation 5.1). These approximations are used for instance, for finding an appropriate step length for difference approximation or for special restraining strategies. Calculation of the step direction belongs to this module.

3. *Restraining strategy.* See examples 4.22. Based on remark 5.9 another strategy is developed.
4. *Stopping criteria* for checking convergence or failure at every iteration step.
5. *Approximation of the inverse jacobian,* including the choice between triangular and singular value decomposition if the approximation is obtained as (generalized) inverse of an approximation to the jacobian. Note that no explicit inverse is calculated in this case, only a decomposition is kept.

We will treat these modules separately in the next sections. In section 6.2 we give the algorithms for numerical algebra computations, viz. triangular decomposition, singular value decomposition and scaling of a matrix by diagonal matrices. Then, in section 6.3, we treat module 5 as it provides the basis for other modules. Modules 2, 3 and 4 are treated subsequently in sections 6.4 up to 6.6. Initialization (module 1) is treated together with the synthesis of basic algorithms (section 6.7), because e.g. the choice of H_0 (or B_0) depends on the basic algorithm chosen. In section 6.8 up to 6.11 we describe subsequently the conditional use of approximation by updating and fixed approximation, implicit and explicit scaling and reduction of problems with linear components. These strategies are considered as optional features. They can be applied to all or some of the basic algorithms.

In this chapter we assume that the function F satisfies condition 1.5 and if we write $(J(x))^{-1}$, for some $x \in D$, then it is implicitly assumed that this inverse exists. If we use notation 5.1 then it is assumed that the conditions for using this notation are satisfied. In addition to notation 5.1 we use: $\gamma(x_k) = \gamma_k$, with $\gamma(x)$ the Lipschitz constant in condition 1.6 for $x \in D$, and if $B \in L(\mathbb{R}^n)$ is some approximation to $J(x)\,(x \in D)$, then

(6.1) $$\eta(x) = \|(fl_\varepsilon(B))^{-1}\|,\ \eta^+(x) = \|(fl_\varepsilon(B))^+\|, \qquad \eta_k = \eta(x_k),\ \eta_k^+ = \eta^+(x_k),$$

(6.2) $$e^+(x) = \|(J(x)-B)B^+\|, \qquad e_k^+ = e^+(x_k).$$

Furthermore $r(x)$ denotes the rank of B and $r_k = r(x_k)$. The approximation to a certain quantity, which will be used in the algorithms, is always denoted with "^" (e.g. $\hat{\eta}_k$ is the approximation to η_k).

We assume that numerical computation of the function and its jacobian is such that constants ε_{rf}, $\varepsilon_{rj} \geq \varepsilon$, ε_{af}, $\varepsilon_{aj} \geq 0$ exists such that

(6.3) $\qquad \| fl_\varepsilon(F(x))-F(x)\| \leq \varepsilon_{rf}\|F(x)\| + \varepsilon_{af}$,

(6.4) $\qquad \| fl_\varepsilon(J(x))-J(x)\| \leq \varepsilon_{rj}\|J(x)\| + \varepsilon_{aj}$,

where ε denotes the machine precision (see notation 3.8).

We give formal descriptions of the modules and additional features in ALGOL 68 (see WIJNGAARDEN et al [1976]). We use the ALGOL 68 implementation (TORRIX 68) of the programming system TORRIX (see MEULEN & VELDHORST [1978]). The prelude used for numerical algebra routines (prelude name: naprel) is based on HEMKER & WINTER [1979]. The problem of solving a system of non-linear equations with Newton-like methods is defined in the prelude with name: nlsprl. So nlsprl is embedded in naprel, which is embedded in the TORRIX 68 prelude, which is, in turn, embedded in the ALGOL 68 standard prelude. Our primary objective is to use ALGOL 68 as a reference language for unambiguous description of our algorithms. The ALGOL 68 programs have been compiled by the A68 compiler of the CYBER 73 system at the computer center SARA at Amsterdam. We have tested some example programs for some small test problems in order to obtain some faith that these programs are correct. This testing has been performed using an optimized version of TORRIX BASIS, in which system .mode .scal = .real and in which the dyadic operators .max and .min have been defined for arguments of .mode .scal. We have not performed a full testing of the ALGOL 68 programs. Our experiments have been performed in ALGOL 60 (see chapter 7).

Description in ALGOL 68

naprel:

```
   # numerical algebra prelude,
     J.C.P. Bus, update 800103,
     to be compiled by: a68,i=lfn,p=numal3/tormin,n.
     where tormin is an optimized version of the torrix basis prelude
     (see MEULEN en VELDHORST [1978])
     (tormin contains the operators .max  and .min  for .scal  operands)
```

```
  #
.begin

   .mode  .prb  = .struct (.scal  relacc, absacc, reltol, abstol,
                          .int  maxit);
   .mode  .prob  = .ref .prb ;
   .mode  .matprob  = .struct (.mat  mat, .prob  prob),
          .lud  = .struct (.mat  lu, .index  piv, .scal  nrm,
                          .bool  ready),
          .svd  = .struct (.mat  u, v, .vec  sngval, .bool  ready);
   .mode  .dec  = .union (.ref .lud , .ref .svd );
   .mode  .scldmat  = .struct (.mat  mat, .vec  rows, .vec  cols,
                              .bool  scr, .bool  scc, .bool  sng);

   .op  .defprob  = (.mat  m).prob :
      (.heap  .prb  prob:=
       (small scal, small scal, small scal * ten, small scal * ten,
        .size  m * 10); prob
      );

   .op  (.matprob ).lud  .ludec  = .pr  XREF LUDEC .pr  .skip ;
   .op  (.matprob ).svd  .svdec  = .pr  XREF SVDEC .pr  .skip ;
   .op  .ludec  = (.mat  m).lud : (.ludec  .matprob (m, .defprob  m));
   .op  .svdec  = (.mat  m).svd : (.svdec  .matprob (m, .defprob  m));

   .op  .check  = (.lud  lud).bool : ready .of  lud;
   .op  .check  = (.svd  svd).bool : ready .of  svd;

   .op  (.lud , .vec ).vec  .sol  = .pr  XREF LUSOL .pr  .skip ;
   .op  (.svd , .vec ).vec  .sol  = .pr  XREF SVSOL .pr  .skip ;
   .op  .solve  = (.dec  dec, .vec  rhs).vec :
      .case  dec .in
      (.ref .lud  lud): lud .sol  rhs,
      (.ref .svd  svd): svd .sol  rhs
      .esac ;

   .op  .trims  = (.scal  r, .svd  svd).svd :
      .begin  .mat  u = u .of  svd, v = v .of  svd;
         .vec  sv = sngval .of  svd;
         .int  n = 1 .upb  u .min  2 .upb  u; .int  k, rk:= 0, i:= 0;
         .while  (i +:= 1;
                 (i <= n ! (k .max  sv[i:n .at  i]) > r ! .false ))
         .do  rk +:= 1; .if  k /= i
            .then  sv[i] =:= sv[k]; u[,i] =:= u[,k]; v[,i] =:= v[,k]
            .fi
         .od ;
         .svd (u[,1:rk], v[,1:rk], sv[1:rk], ready .of  svd)
      .end  #operator trims #;

   .prio  .sol  = 2, .solve  = 2, .trims  = 3;

   .op  .sqr  = (.vec  x).scal  : x * x;
   .op  .sqr  = (.scal  x).scal  : x * x;

   .op  .nrm  = (.vec  x).scal :
      (.scal  max := .maxabs  x; .if  max <= minscal .then  zero
       .else  .vec  y = x / max; sqrt(y * y) * max .fi
      ) # vector norm with avoiding overflow due to squaring #;

   .prio  .nrm  = 8;                                               #
```

```
#
   .proc  genranvec = (.int  n).vec :
   .begin  .proc  ran = (.int  i).scal : next random(setr);
      .vec  v = ran .into  genvec(n); v /< (.nrm  v)
   .end  # generation of vectors with random elements #;

   .int  setr:= 10;
   .scal  zero = .widen  0,
          one  = .widen  1,
          two  = .widen  2,
          ten  = .widen  10;
   .scal  onetenth = one / ten,
          minscal = two ** (-975)
                    # we choose this value, which performs well.
                      a precise choice of the smallest normalized real
                      number requires a routine in machine language. #,
          max scal = max real,
          small scal = small real;

   .proc  rotvec = (.vec  a, b, .scal  c, s).void :
   .begin  .int  l = .lwb  a, u = .upb  a;
      .for  k .from  l .to  u
      .do  .scal  x = a[k], .ref .scal  y = b[k];
         a[k]:= c * x + s * y; y:= c * y - s * x
      .od
   .end  # rotation of two vectors #;

   .op  .dmul  = (.vec  d, x).vec :
      (((.int  i).scal :(d[i] * x[i])) .into
       genvec(.upb  x .max  .upb  d));

   .op  .dimul  = (.vec  d, x).vec :
      (((.int  i).scal :(x[i] / d[i])) .into
       genvec(.upb  x .max  .upb  d));

   .prio  .dmul  = 6, .dimul  = 6;

   .op  (.mat ).scldmat  .scale  = .pr  XREF SCALE .pr  .skip ;
   .op  (.scldmat ).scldmat  .scale  = .pr  XREF SCALMAT .pr  .skip ;
   .op  (.scldmat ).mat  .bckscale  = .pr  XREF BCKSCLE .pr  .skip ;


   .pr  PROG .pr  .skip

.end  # naprel, numerical algebra prelude#
```

```
nlsprl:

   # prelude for newton-like methods for solving systems of nonlinear
     equations by J.C.P. Bus, update 80013,
     embedded in numerical algebra prelude naprel in nl68lib (id=jbus),
     in this prelude a problem is defined and the various identifiers
     used in the modules of the algorithms are defined and set to
     default. this design is chosen in order to be able to use algol 68
     as a reference language for unambiguous definition of the
     algorithms. to be compiled by the following control cards:
     attach,nl,nl68lib,id=jbus.
     a68,i=lfn,p=nl/naprel,n.
   #

.begin

   .mode  .func  = .struct (.vec  f, .bool  in);

   #******************** problem definition (see definition 7.2) *****#

   .proc  fun:= (.vec  x).func : .skip
                # the procedure defining the problem function has to be
                  assigned to fun,
                  the variables are given in x,
                  on exit either in .of  fun(x) = .true  and f .of
                  fun(x) contains the function vector, or
                  in .of  fun(x) = .false
                #;

   .proc  jacobian:= (.vec  x).mat : .skip
                # the procedure defining the problem jacobian has to be
                  assigned to jacobian,
                  the variables are given in x,
                  jacobian will only be called if x is in the domain
                  of the function (fun(x) is called first),
                  on exit the jacobian matrix at x is delivered
                #;

   # it is assumed that the procedures to be assigned to fun and
     jacobian only use identifiers which have the same scope as
     fun and jacobian #

   .int  n      # the order of the function #;
   .bool  linpart:= .false  # linpart = .true  iff function has linear
                  components, default value is .false  #;
   .vec  x0     # the initial guess to the solution #,
         lb     # the right hand side of the linear part #;
   .mat  la     # the matrix of the linear part #;
   .scal  eprf  # the rel. prec. of the function (see (6.3)) #,
          epaf  # the abs. prec. of the function (see (6.3)) #,
          eprj  # the rel. prec. of the jacobian (see (6.4)) #,
          epaj  # the abs. prec. of the jacobain (see (6.4)) #,
          dlf   # the tolerance of the function norm #,
          dlrx  # the rel. tolerance of the variables #,
          dlax  # the abs. tolerance of the variables #;

   #*************************** end problem definition ***************
```

```
#
   .int  it     # iteration counter #,
         fcnt   # function evaluation counter #,
         jcnt   # jacobian evaluation counter #,
         dcnt   # decomposition counter #,
         maxit  # maximum number of iterations allowed #;
   maxit:= 50 # default value of maxit #;

   .bool  dif   # .true  iff difference approximation is used #,
          anl   # .true  iff analytic jacobian is used #,
          fix   # .true  iff fixed approximation is used #,
          upd   # .true  iff update approximation is used #,
          safe  # .true  iff all failure criteria have to be used #,
          scale # .true  iff scaling allowed #,
          nongener # .true  iff use of nongeneralized method allowed #,
          gener # .true  iff use of generalized method allowed #,
          update # .true  iff conditional updating allowed #;
          dif:= anl:= fix:= upd:= scale:= .false ;
          safe:= nongener:= gener:= update:= .true ;
          # setting default values of these booleans #

   .vec  x      # the current vector of variables #,
         xj     # the last point at which non fixed approximation is
                  used #,
         f      # the current function vector #,
         f0     # the previous function vector if inverse-updating is
                  used, otherwise the function vector at xj #,
         dx     # the current step vector #,
         v      # a vector with random elements and norm 1 #,
         w      # the product bl * v, with bl the previous approximation
                  to the jacobian #,
         sol    # for use in reducenewt, see section 6.11 #;

   .mat  b      # the current approximation to the jacobian (or its
                  inverse in inverse-updating methods) #,
         v2     # for use in reducenewt, see section 6.11 #;

   .dec  decb   # the current decomposition of b #;

   .mode  .metric  = .struct (.int  c, .ref .lud  deca);
   .metric  a    # if c .of  a = 1 then the matrix a in the levelfunct-
                  ion (see (4.23)) is the identity matrix,
                  if c .of  a = 2 then a is the inverse of the jacobian
                  approximation at x0 and the decomposition is given in
                  deca .of  a,
                  if c .of  a = 3 then a is the inverse of b and its
                  decomposition is given in deca .of  a #;
    c .of  a := 1 # default value #;

   .scal  epsh = small scal * .widen  100 #lower bound differ. step#;
   .scal  nrmx, nrmf, nrmdx
                # norms of x, f and dx #,
          slevel# the square root of the value of the levelfunction #,
          omga  # the approximation to omega or gamma (section 6.3) #,
          beta, kappa, eta, labda, e
                # the approximations to these variables (section 6.3) #,
          eta0  # the value of eta in the first iteration step #,
          h     # the difference step length #,
          nrmul, nrmu2
                # nrm(ul) and nrm(u2), see section 6.2.2 #,
          epf   # (eprf + small scal) * nrmf + epaf #,
          ej    # the error in b(xj) #;
   .scldmat  scb# gives scaled matrix b if explicit scaling is used #;
                                                                       #
```

```
#
   [ ].char
   text1 = "no progress, maybe due to too high required precision",
   text2 = "no progress relative to error in function",
   text3 = "stationary point of norm of the function, no solution",
   text4 = "too many function evaluations or iterations required",
   text5 = "numerical singularity in triangular decomposition",
   text6 = "failure of singular value decomposition",
   text7 = "rank of jacob. approx. equal to zero",
   text8 = "error in jacob. approx. yields possible singular jacob.",
   text9 = "nearby singularity of jacobian expected",
   text10= "difference approx. impossible, point on boundary domain",
   text11= "divergence out of domain of function",
   text12= "starting point not in domain of function",
   text13= "eprf set to default (small scal)",
   text14= "epaf set to default (0)",
   text15= "eprj set to default (small scal)",
   text16= "epaj set to default (0)",
   text17= "dlf set to default (epaf)",
   text18= "dlrx set to default (small scal)",
   text19= "dlax set to default (small scal)";

   .proc  .scal  calh = .pr  XREF CALH .pr  .skip ;
   .proc  .scal  calgh = .pr  XREF CALGH .pr  .skip ;
   .proc  (.vec ) .mat  diffjac = .pr  XREF DIFFJAC .pr  .skip ;
   .proc  (.mat ).void  invupd1 = .pr  XREF INVUPD1 .pr  .skip ;
   .proc  (.mat ).void  invupd2 = .pr  XREF INVUPD2 .pr  .skip ;
   .proc  .void  cdatalr = .pr  XREF CDATALR .pr  .skip ;
   .proc  .void  cdatasv = .pr  XREF CDATASV .pr  .skip ;
   .proc  (.ref .vec , .ref .vec ).scal  slefu =
   .pr  XREF SLEFU .pr  .skip ;
   .proc  .void  strict = .pr  XREF STRICT .pr  .skip ;
   .proc  .void  resbis = .pr  XREF RESBIS .pr  .skip ;
   .proc  (.scal ,.scal ,.scal ,.scal ,.scal ,.scal ).scal  quad
      = .pr  XREF QUAD .pr  .skip ;
   .proc  (.ref .scal ,.ref .scal ,.ref .scal ,.ref .scal ,.ref .scal ,
      .proc (.scal ).scal ,.scal ).bool  interp
      = .pr  XREF INTERP .pr  .skip ;
   .proc  .void  resint = .pr  XREF RESINT .pr  .skip ;
   .proc  (.ref .scal ,.ref .scal ,.ref .scal ,.ref .scal ,
      .proc (.scal ).scal ).bool  extrap = .pr  XREF EXTRAP .pr  .skip ;
   .proc  .void  resest = .pr  XREF RESEST .pr  .skip ;
   .proc  .bool  stopful = .pr  XREF STOPFUL .pr  .skip ;
   .proc  .bool  stopspl = .pr  XREF STOPSPL .pr  .skip ;
   .proc  .void  default = .pr  XREF DEFAULT .pr  .skip ;
   .proc (.proc (.vec ).mat ).mat  conupdjac =
      .pr  XREF CONUPD .pr  .skip ;
   .proc (.proc (.vec ).mat ).mat  confixjac =
      .pr  XREF CONFIX .pr  .skip ;
   .proc (.proc (.vec ).mat ).mat  confixjacg =
      .pr  XREF CONFIXG .pr  .skip ;
   .proc (.proc .bool ).bool  reducenewt = .pr  XREF REDNEWT .pr
      .skip ;

   .pr  PROG .pr  .skip

.end  # prelude for nonlinear system solving #
```

6.2. NUMERICAL ALGEBRA ALGORITHMS

6.2.1. Triangular decomposition

Given a nonsingular matrix $B \in L(\mathbb{R}^n)$, we obtain a triangular decomposition by the process of triangularization with partial pivoting (i.e. with row interchanges only) (see WILKINSON [1965, section 4.15 - 4.23]). Hence, we find a permutation matrix P, a unit upper-triangular matrix U and a lower-triangular matrix L such that

$$PB = LU.$$

Of course, round-off errors will occur during this process. In fact, if we solve a linear system Bx = b in such a way, then we obtain the exact solution of the system (see DEKKER [1971] and WILKINSON [1965, section 4.24 - 4.29])

(6.5) $\quad (B + E_2)x = b$

with

(6.6) $\quad \|E_2\| \leq \varepsilon g(n^3+5n^2)$

and g the so-called growth (i.e. the modulus of the in modulus largest element in the matrix during the process of triangularization). Although, one can construct pathological examples for which g becomes as large as $2^{n-1} \max_{i,j} |B_{ij}|$, in practice such a growth is very rare. With reference to DEKKER [1971, section 5] and WILKINSON [1965, section 4.27] we choose g equal to a fixed multiple of some norm of B. We also replace n^3+5n^2, by simply n, as it appears in practice that (6.6) yields a severe overestimate. So, we obtain

(6.7) $\quad \|E_2\| \leq 16\varepsilon n \, \|B\|.$

In order to increase the numerical stability of triangularization one might use complete pivoting (row and column interchanges) which is expensive, or a combination of both complete and partial pivoting (see BUSINGER [1971] and BUS [1972]). Using the last technique one easily obtaines a reasonable upper bound on the growth g in (6.6). It depends on the software library to be used for a specific implementation of a Newton-like program which particular method shall be used for triangularization. In our experiments in ALGOL 60 we use the combined pivoting strategy mentioned above. In our ALGOL 68 descriptions we simply use partial pivoting. The process of triangularization is terminated if in some stage the moduli of the elements in the first column of the remaining submatrix (see WILKINSON [1965, section 4.20]) are all less than some prescribed precision ($\varepsilon_{rk}\|B\|$). We choose $\varepsilon_{rk}=\varepsilon$ and say that B is numerically singular if the process is terminated too early. We choose this value to avoid calamities like arithmetic overflow on a computer. In this stage, this precision is not related to the error in B as an approximation to J(x). Such criteria are discussed in subsection 6.4.8.

Together with the triangular decomposition we give a program for forward and backward substitution to calculate the solution of a linear system if the triangular decomposition is given.

<u>Description in ALGOL 68</u>

```
.op  .ludec  = (.matprob  m).lud :
.pr  XDEF LUDEC .pr
.begin  .mat  lu = .copy  mat .of  m; .int  n = 1 .upb  lu;
  .bool  ready:= .true ; .scal  bnd:= zero;
  .index  p = .subscr  lu[ ,1]; .vec  v = genvec(n);
  .for  i .to  n
  .do  v[i]:= (.scal  vi= .sqr  lu[i, ];
     bnd +:= vi; .if  vi = zero .then
     ready:= .false ; one .else  one / sqrt(vi) .fi )
  .od ;
  .scal  nrm:= sqrt(bnd); bnd := nrm * (relacc .of  prob .of  m);
  .for  k .to  n .while  ready
  .do  .scal  max:= zero; .int  pk:= k; .vec  colk = lu[ , k];
     .for  i .from  k .to  n
     .do .if  .scal  s= (.abs
           (colk[i] -:= lu[i, :k-1] * colk[ :k-1]) * v[i]); s > max
        .then  pk:= i; max:= s .fi
     .od ;
     .if  max < bnd .then  ready:= .false
     .else  p[k]:= pk; v[pk]:= v[k];
        .if  pk /= k .then  lu[k, ] =:= lu[pk, ] .fi ;
        .vec  rowk = lu[k, ];
        .for  i .from  k + 1 .to  n
        .do  rowk[i] -:= rowk[ :k-1] * lu[ :k-1,i] .od ;
        rowk[k+1: ] /< rowk[k]
     .fi
  .od ; .lud (lu, p, nrm, ready)
.end  # triangular (lu) decomposition of a matrix #
.pr  FEDX .pr ;
```

```
  .op  .sol  = (.lud  lud, .vec  rhs).vec :
  .pr  XDEF LUSOL .pr
  .begin  .int  n = .upb  rhs;
    .index  p = piv .of  lud; .vec  x = .copy  rhs;
    .mat  lu = lu .of  lud;
    .for  k .to  n
    .do .int  pk = p[k], .scal  r = x[k];
       x[k]:= (x[pk] - lu[k, :k-1] * x[ :k-1]) / lu[k,k];
       .if  pk /= k .then  x[pk]:= r .fi
    .od ;
    .for  k .from  n - 1 .by  - 1 .to  1
    .do  x[k] -:= lu[k,k+1: ] * x[k+1: ] .od ;
    x
 .end  # forward and backward substitution # .pr  FEDX .pr ;
```

6.2.2. Singular value decomposition

The singular value decomposition as described here is an ALGOL 68 implementation of the ALGOL 60 procedures from the NUMAL library (HEMKER et al. [1979]). These procedures are transcriptions of the procedures given by GOLUB & REINSCH [1971]. The decomposition consists of four parts:

- the transformation of the matrix to bidiagonal form using Householder orthogonalization (routine: hshreabid),
- calculation of the postmultiplying matrix from the Householder matrices used to transform the matrix into bidiagonal form (routine: psttfmmat),
- calculation of the premultiplying matrix from the Householder matrices used to transform the matrix into bidiagonal form (routine: pretfmmat),
- transformation of the bidiagonal matrix to diagonal form by the QR-iteration process (routine: svdecbid).

The routines are combined in one operator (.op .svdec) which deliveres an unordered singular value decomposition (the singular values are not given in non-increasing order). Application of the operator .trims (see numerical algebra prelude) orders the singular values and delivers the significant part of the singular value decomposition (U_1, Σ_r and V_1) according to (1.18). So we obtain for a given matrix $B \in L(\mathbb{R}^n)$, orthonormal matrices U_1, V_1 and a diagonal matrix Σ_r (cf. (1.18)) such that

$$B = U_1 \Sigma_r V_1^T.$$

Let B + E be the matrix which is exactly equal to the product of the numerically computed matrices U_1, Σ_r and V_1, i.e.

$$B + E = fl_{\varepsilon}(U_1) fl_{\varepsilon}(\Sigma_r)(fl_{\varepsilon}(V_1))^T.$$

Then we assume that $\|E\| \le c\varepsilon\|B\|$, where c is not much greater than n. This is a reasonable assumption as orthogonal transformations are used to obtain the singular value decomposition (see WILKINSON [1965, section 6.3]).

Operator .sol gives the solution of a linear system if the singular value decomposition of the matrix is given.

Description in ALGOL 68

```
.op  .svdec  = (.matprob  a).svd :
.pr  XDEF SVDEC .pr
.begin
  .proc  hshreabid = (.mat  a, .vec  d, b, .ref .scal  norm).void :
  .begin  .int  m = 1 .upb  a, n = 2 .upb  a; norm:= zero;
    .for  i .to  m
    .do .scal  w:= .sigmabs  a[i, ];
       .if  w > norm .then  norm := w .fi
    .od ;
    .scal  machtol = small scal * norm;
    .for  i .to  n
    .do  .int  il= i + 1;
       .if  .scal  s:= .sqr  a[il : , i]; s <= machtol
       .then  d[i] := a[i,i]
       .else  .vec  ai = a[i : , i]; .ref .scal  f = ai[1];
          s+:= .sqr  f; .scal  g= ( d[i]:=
          .if  f < zero .then  sqrt (s) .else  - sqrt (s) .fi );
          .scal  h= f * g - s; f -:= g;
          .for  j .from  il .to  n
          .do  a[i : , j] +< (a[i : , j] * ai ) / h * ai .od
       .fi ;
       .if  i < n
       .then  .vec  ai = a[i, il : ];
          .if  .scal  s:= .sqr  ai[2 : ]; s <= machtol
          .then  b[i] := ai[1]
          .else  .ref .scal  f = ai[1];
             s+:= .sqr  f; .scal  g = (b[i]:=
             .if  f < zero .then  sqrt (s) .else  - sqrt (s) .fi );
             .scal  h= f * g - s; f -:= g;
             .for  j .from  il .to  m
             .do  a[j, il : ] +< ( a[j, il : ] * ai ) / h * ai .od
          .fi
       .fi
    .od
  .end  # householder bidiagonalization of real matrix #;
```

```
.proc  psttfmmat = (.mat  a, .mat  v, .vec  b).void  :
.begin  .co  psttfmmat .co
   .int  n = 2 .upb  a; # check : n = (1 and 2) .upb  v #
   v[n,n] := one;
   .for  i .from  n-1 .by  -1 .to  1
   .do  .int  il= i + 1; .vec  ai = a[i, il : ], vi = v[il : , i];
      .int  revn = .upb  ai;
      .if .scal  h= b[i] * ai[1]; h < zero
      .then .for  j .to  revn .do  vi[j]:= ai[j] / h .od ;
         .for  j .from  il .to  n
         .do  v[il : , j] +< ( v[il : , j] * ai ) * vi .od
      .fi ;
      .for  j .to  revn .do  v[i, j + i]:= vi[j] := zero .od ;
      v[i,i] := one
   .od
 .end  # post transformation matrix of householder matrices #;


.proc  pretfmmat = (.mat  a, .vec  d).void  :
.begin  .co  pretfmmat .co
   .int  m = 1 .upb  a, n = 2 .upb  a;
   .for  i .from  n .by  -1 .to  1
   .do  .vec  ai = a[i : m, i]; .int  revm = .upb  ai, il = i + 1,
      .scal  g= d[i]; .scal  h= g * ai[1];
      .for  j .from  il .to  n .do  a[i,j] := zero .od ;
      .if  h < zero
      .then  .for  j .from  il .to  n
         .do  a[i : , j] +< (ai[2 : ] * a[il : , j]) / h * ai .od ;
         .for  j .to  revm .do  ai[j] /:= g .od
      .else  .for  j .to  revm .do  ai[j]:= zero .od
      .fi ;
      ai[1]+:= one
   .od
 .end  # pre transformation matrix of householder matrices #;


.proc  svdecbidqr = (.vec  d, b, .mat  u, v, .real  nrm,
                     .prob  prob).bool :
.begin .scal  c, s,
  .int   n = .upb  d, m = 1 .upb  u;
  .int   nn:= n,
  .scal  eps:= relacc .of  prob, bmax:= zero;
  .int  count:= 0, rnk:= n, imax:= maxit .of  prob;
  .scal  dmin:= reltol .of  prob, tol:= eps * nrm;
  .while  nn > 0
  .do  .int  k:= nn, .int  nl= nn - 1;
  next :
     .if  k-:= 1; k > 0
     .then .if  .abs  b[k] >= tol
        .then .if  .abs  d[k] >= tol .then  next .fi ;
           c:= zero; s:= one;
           .for  i .from  k .to  nl
           .do  .int  il= i + 1, .scal  f= s * b[i]; b[i]*:= c;
              .if  .abs  f < tol .then  neglect .fi ;
              .scal  g= d[il];
              .scal  h = (d[il]:= sqrt(.sqr  f + .sqr  g));
              c:= g / h; s:= - f / h;
              rotvec(u[ , k], u[ , il], c, s)
           .od ;
        neglect : .skip
        .elif  .abs  b[k] > bmax .then  bmax:= .abs  b[k]
        .fi
     .fi ;                                                    #
```

```
          .if  k = nl
          .then .scal  dnn= d[nn]; .if  dnn < zero
             .then  d[nn]:= - dnn;
                .for  i .to  n .do  v[i,nn]:= - v[i,nn] .od
             .fi ;
             .if  d[nn] <= dmin .then  rnk -:= 1 .fi ; nn:= nl
          .else  .if  count+:= 1; count > imax .then  end .fi ;
             .int  kl= k + 1;
             .scal  z= d[nn], .scal  x:= d[kl], y:= d[nl],
             g:= .if  nl = 1 .then  zero .else  b[nl - 1] .fi ,
             h:= b[nl];
             .scal  f:= ((y-z) * (y+z) + (g-h) * (g+h)) / (2*h*y);
             g:= sqrt(.sqr  f + one);
             .if  f < zero .then  f-:= g .else  f+:= g .fi ;
             f:= ((x - z) * (x + z) + h * (y / f - h)) / x;
             c:= s:= one;
             .for  i .from  kl + 1 .to  nn
             .do  .int  il= i - 1; g:= b[il]; y:= d[i];
                h:= s * g; g*:= c; .scal  z=
                   sqrt(.sqr  f + .sqr  h); c:= f / z; s:= h / z;
                .if  il /= kl .then  b[il - 1]:= z .fi ;
                f:= x * c + g * s;
                g:= g * c - x * s; h:= y * s; y*:= c;
                rotvec(v[ , il], v[ , i], c, s);
                .scal  zl= (d[il]:= sqrt(.sqr  f + .sqr  h));
                .if  zl < small scal
                .then  c:= one; s:= zero; f:= g; x:= y
                .else  c:= f / zl; s:= h / zl;
                   f:= c * g + s * y; x:= c * y - s * g;
                   rotvec(u[ , il], u[ , i], c, s)
                .fi
             .od ;
             b[nl]:= f; d[nn]:= x
          .fi
       .od ;
    end : .skip ; nn = 0
    .end  # qr iteration on bidiagonal matrix yielding svd #;

    .mat  u = .copy  mat .of  a; .bool  ready;
    .int  m = 1 .upb  u, n= 2 .upb  u;
    .vec  b = genvec(n), sv = genvec(n);
    .mat  v = gensquare(n); .scal  norm;
    .if  m < n .then  ready:= .false
    .else  hshreabid(u, sv, b, norm);
      psttfmmat(u, v, b); pretfmmat(u, sv);
      ready:= svdecbidqr(sv, b, u, v, norm, prob .of  a)
    .fi ;
    .svd (u, v, sv, ready)
  .end  # singular value decomposition of matrix #
  .pr  FEDX .pr ;


  .op  .sol  = (.svd  svd, .vec  rhs).vec :
  # it is assumed that svd is trimmed at least with zero #
  .pr  XDEF SVSOL .pr
  .begin  .vec  x = rhs * u .of  svd;
    .for  i .to  .upb  x
    .do  x[i] /:= (sngval .of  svd)[i] .od ;
    v .of  svd * x
  .end  # solution of system with sing val decomposition #
  .pr  FEDX .pr ;
```

6.2.3. Scaling of a matrix

We give a description of scaling of a matrix with diagonal matrices with elements equal to powers of two, according to subsection 1.3.5. The pré- and post-multiplying scaling matrices are defined as in lemma 1.31, except for the use of a simpler row and column norm. Instead of using the euclidean norm we use the infinity norm: $\|x\| = \max_i |\xi_i|$, with $x = (\xi_1,\ldots,\xi_n)^T$. We also give here a description of scaling of a matrix if the scaling matrices are known (.op scale) and of backscaling of a matrix (.op .bckscle).

Description in ALGOL 68

```
 .op  .scale  = (.mat  a) .scldmat :
 .pr  XDEF SCALE .pr
 # scale will yield a scldmat whose field mat points to a,the matrix
   a is changed, its rows are multiplied with factors given in rows
   and its columns with factors given in cols #
 .begin  .scal  ln2:= ln(two);
    .int  n = 1 .upb  a, m = 2 .upb  a;
    .scal  max:= .widen  (n * 4); .scal  min:= one / max;
    .bool  reg:= .true , scr:= .false , scc:= .false ;
    .vec  rows = one .into  genvec(n), cols = one .into  genvec(m);
    .for  i .to  n .while  reg
    .do  .scal  norm:= .maxabs  a[i,];
       .if  (reg:= norm >= minscal) .and
            (.bool  bl = norm > max .or  norm < min;
             scr:= scr .or  bl; bl)
       .then  a[i,] *< (rows[i]:= two ** .entier (-ln(norm)/ln2)).fi
    .od ;
    .for  j .to  n .while  reg
    .do  .scal  norm:= .maxabs  a[,j];
       .if  (reg:= norm >= minscal) .and
            (.bool  bl = norm > max .or  norm < min;
             scc:= scc .or  bl; bl)
       .then  a[,j] *< (cols[j]:= two ** .entier (-ln(norm)/ln2)).fi
    .od ;
    .scldmat (a, rows, cols, scr, scc,.not  reg)
 .end  # scaling of rows and columns of a matrix #
 .pr  FEDX .pr ;
```

```
.op  .bckscle  = (.scldmat  sa) .mat :
.pr  XDEF BCKSCLE .pr
.begin  .mat  a = mat .of  sa; .if  scr .of  sa
   .then  .vec  rows = rows .of  sa;
      .for  i .to  .upb  rows .do  a[i,] /< rows[i] .od
   .fi ;
   .if  scc .of  sa
   .then  .vec  cols = cols .of  sa;
      .for  j .to  .upb  cols .do  a[,j] /< cols[j] .od
   .fi ;
   a
.end  # backscaling of scaled matrix #
.pr  FEDX .pr ;

.op  .scale  = (.scldmat  sa).scldmat :
 # this operator scales the matrix in sa with the factorsgiven in
   sa, the matrix in sa is changed,in fact sa = .scale  sa after
   completion #
.pr  XDEF SCALMAT .pr
.begin  .mat  a = mat .of  sa; .if  scr .of  sa
   .then  .vec  rows = rows .of  sa;
      .for  i .to  .upb  rows .do  a[i,] *< rows[i] .od
   .fi ;
   .if  scc .of  sa
   .then  .vec  cols = cols .of  sa;
      .for  j .to  .upb  cols .do  a[,j] *< cols[j] .od
   .fi ; sa
.end  # scaling of matrix with known scaling matrices #
.pr  FEDX .pr ;
```

6.3. CHOICE OF APPROXIMATION TO THE INVERSE JACOBIAN

6.3.1. Introductory remarks

In the next subsections we describe in detail the various choices of $\Psi_2(x,H)$ given in sections 4.2 and 4.4. We restrict attention to five basic choices:

- inverse of analytic jacobian,
- inverse of difference approximation,
- inverse-updating approximation,
- generalized inverse of analytic jacobian,
- generalized inverse of difference approximation.

From remark 3.14 we see that the inverse of an update approximation is equal to an appropriate inverse-update of the inverse of the original matrix. As updating Newton methods require $o(n^3)$ arithmetical operations at each step and inverse-updating Newton methods $o(n^2)$, we prefer inverse-updating as a basic choice. Conditional use of updating as well as fixed approximation are considered as optional features (see sections 6.8 and 6.9). We expect fixed Newton algorithms to be inferior, particularly if initial guesses to the solution are not good. Therefore, fixed approximation does not belong to the set of basic choices given above.

6.3.2. Inverse of analytic jacobian

We assume that a routine is given in which the calculation of the analytic expressions for the elements of the jacobian matrix are programmed (see .proc jacobian in prelude nlsprl, section 6.1). In fact, neither at the initial phase, nor at the iteration steps, the inverse jacobian is calculated. Only a triangular decomposition is made and the linear systems are solved by using this decomposition (see subsection 6.2.1).

6.3.3. Inverse of difference approximation

In order to calculate the difference approximation (see (3.5)) we have to choose values for the difference steps h_i $(i=1,\dots,n)$ in such a way that the approximation is well defined and its error is as small as possible. Theorem 3.9 gives an upper bound on the approximation error $e(x)$. For simplicity we choose h_i $(i=1,\dots,n)$ by

(6.8) $\qquad h_i = h(1+|\xi_i|)$,

for $x = (\xi_1,\ldots,\xi_n)^T$. Then with the notation $u_1 = (1+|\xi_1|,\ldots,1+|\xi_n|)^T$, $u_2 = ((1+|\xi_1|)^{-1},\ldots,(1+|\xi_n|)^{-1})^T$, and assuming that F and x satisfy condition 1.7 on some open neighbourhood U of x containing $x + h_ie_i$ $(i=1,\ldots,n)$ we obtain by (3.13)

(6.9) $\qquad e(x) \le \dfrac{1}{1-\bar{c}_1h}(\bar{c}_2h^{-1}+\bar{c}_1h)$,

with

$$\bar{c}_1 = \tfrac{1}{2}\omega(x)\|u_1\|; \qquad \bar{c}_2 = 2.12\varepsilon_f(x)\eta(x)\|u_2\|,$$

$$\varepsilon_f(x) = (\varepsilon+\varepsilon_{rf})\|F(x)\| + \varepsilon_{af}$$

and $\eta(x)$ given by (6.1). This upper bound on $e(x)$ attains its minimum with respect to h $(0 < h < 1/\bar{c}_1)$ for $h = h_{opt}$, with

$$h_{opt} = \bar{c}_2\left(-1+\sqrt{1+(\bar{c}_1\bar{c}_2)^{-1}}\right),$$

provided $\bar{c}_1\bar{c}_2 \ne 0$. If $\bar{c}_1\bar{c}_2 = 0$ then $\omega(x) = 0$ (F is linear) or $\varepsilon_f(x) = 0$ ($\varepsilon_{af} = 0$ and $F(x) = 0$). We like to avoid that $\|h_ie_i\|$ becomes to large so that possibly $x + h_ie_i$ lies outside the domain of the function.
Therefore we use 1 as an upper bound on h. Furthermore, if h_i is chosen too small then it may happen that $fl_\varepsilon(\xi_i+h_i) = \xi_i$. As $h_i > \varepsilon_h|\xi_i|$ $(\varepsilon_h \ge \varepsilon)$ guarantees that $fl_\varepsilon(\xi_i+h_i) \ne \xi_i$, we demand $h_{opt} \ge \varepsilon_h$, with $\varepsilon_h = 100\varepsilon$. Some calculations show that $1 \ge h_{opt} > \varepsilon_h$ is equivalent to the conditions

$$(2\bar{c}_1-1)\bar{c}_2 + \bar{c}_1 \ge 0, \qquad (2\varepsilon_h\bar{c}_1-1)\bar{c}_2 + \varepsilon_h^2\bar{c}_1 < 0.$$

We use these conditions in the definition of the approximated optimal value for h

$$(6.10)\quad \hat{h} = \begin{cases} 1, & \text{if } (2\hat{c}_1-1)\hat{c}_2 + \hat{c}_1 < 0, \\ \varepsilon_h, & \text{if } (2\varepsilon_h\hat{c}_1-1)\hat{c}_2 + \varepsilon_h^2\hat{c}_1 \ge 0, \\ \hat{c}_2\left(-1+\sqrt{1+(\hat{c}_1\hat{c}_2)^{-1}}\right), & \text{otherwise,} \end{cases}$$

where $\varepsilon_h = 100\varepsilon$,

(6.11) $\hat{c}_1 = \tfrac{1}{2}\hat{\omega}(x)\|u_1\|, \quad \hat{c}_2 = 2\varepsilon_f(x)\hat{\eta}(x)\|u_2\|,$

and $\hat{\omega}(x)$ and $\hat{\eta}(x)$ approximations to $\bar{\omega}$ and $\eta(x)$ (see section 6.4). Note that, at the moment we calculate $\hat{h}$ according to (6.10), we have not yet available the difference approximation. Therefore $\hat{\eta}(x)$ will be based on the approximation to the jacobian in the previous iteration step. Finally, if $x + h_i e_i \notin D$ for some i (i=1,...,n), then we choose $h_i = \varepsilon_h(1+\xi_i)$. If this still yields a point outside D then the iterative process is terminated (error exit of program).

The initial values of $\hat{\omega}(x)$ and $\hat{\eta}(x)$ are chosen equal to 1. This yields an initial value for $\hat{h}$, which is required to calculate $B(x_0)$, using (6.10).

<u>Description in ALGOL 68</u>

```
.proc  calh = .scal :
.pr  XDEF CALH .pr
.begin  nrmul:= nrmu2:= zero; .for  i .to  n
   .do  .scal  aid:= (one + .abs  x[i]) ** 2;
      nrmul +:= aid; nrmu2 +:= (one / aid)
   .od ; nrmul:= sqrt(nrmul); nrmu2:= sqrt(nrmu2);
   .scal  cl:= nrmul * omga / two, c2:= nrmu2 * epf * eta * two;
   h:= (.if  (cl * two - one) * c2 + cl < zero .then  one
        .elif  (cl * epsh * two - one) * c2 + cl * epsh ** 2 >= zero
        .then  epsh
        .else  (sqrt(one + one / (cl * c2)) - one) * c2
        .fi )
.end  #calculation of difference step in non-generalized case #
.pr  FEDX .pr ;

.proc  diffjac = (.vec  x).mat :
.pr  XDEF DIFFJAC .pr
.begin  .mat  jac = gensquare(n);
   fcnt +:= n; .for  j .to  n
   .do  .vec  xl = .copy  x;
      .scal  hj:= .abs  x[j] * h + h; xl[j] +:= hj;
      .func  fu:= fun(xl); .bool  in:= in .of  fu;
      .if  (in ! .true
           ! hj:= small scal * ten * ten;
             hj +:= (.abs  x[j] * hj); xl[j]:= x[j] + hj;
             fu:= fun(xl);
             (in .of  fu ! .true  ! torrix(warning, text10); .false )
           )
      .then  jac[,j]:= (f .of  fu - f) / hj .fi
   .od ; jac
.end  # difference approximation of jacobian for given step #
.pr  FEDX .pr ;
```

6.3.4. Inverse-updating approximation

We distinguish two inverse-jacobian update functions (see remark 3.17). These are $V(u(x,H))$ (see (3.18)) with

(6.12) $\quad u(x,H) = H(F(\Psi_1(x,H))-F(x))$

and

(6.13) $\quad u(x,H) = \Psi_1(x,H) - x.$

We combine these methods with fixed approximations as follows

(6.14) $$\bar{\Psi}_2(x,H) = \begin{cases} V(u(x,H))(H,x,\Psi_1(x,H)), & \text{if } (x,H) \in D_\Psi, \\ H, & \text{if } (x,H) \notin D_\Psi, \end{cases}$$

where Ψ defines the inverse-updating process, $\bar{\Psi}$ the resulting combined process ($\bar{\Psi}_1 \equiv \Psi_1$) and D_Ψ the domain of Ψ (see (4.15)).

Description in ALGOL 68

```
.proc  invupdl = (.mat  b).void :
.pr  XDEF INVUPDl .pr
.begin  .vec   u = b * (f - f0);
   .if  .abs  (dx * u) > small scal * nrmdx * .nrm  u
   .then  .vec  v = dx - u; u:= (u / .sqr (u)) * b;
      .for  j .to  n .do  b[,j] +< (u[j] * v) .od
   .fi
.end  # inverse updating with u = h \ (f - f0) #
.pr  FEDX .pr ;

.proc  invupd2 = (.mat  b).void :
.pr  XDEF INVUPD2 .pr
.begin  .vec   v = b * (f - f0);
   .scal  dxv:= dx * v;
   .if  .abs  dxv > small scal * nrmdx * .nrm  v
   .then  .vec  u = (dx * b) / dxv; v -< dx;
      .for  j .to  n .do  b[,j] -< (u[j] * v) .od
   .fi
.end  # inverse updating with u = dx #
.pr  FEDX .pr ;
```

6.3.5. Generalized inverse of analytic jacobian

The analytic jacobian is given by .proc jacobian (see prelude nlsprl in section 6.1). We do not calculate the generalized inverse but only a singular value decomposition (see subsection 6.2.2.).

6.3.6. Generalized inverse of difference approximation

As in subsection 6.3.3 we have to choose values for the difference steps h_i $(i=1,\dots,n)$ in such a way that the approximation is well defined and its error is as small as possible. Theorem 3.10 gives an upper bound on the approximation error $e^+(x)$ (cf. (6.2)). We obtain, with h_i as in (6.8)

(6.15) $$e^+(x) \le \bar{c}_1^+ h + \bar{c}_2^+ h^{-1},$$

where

$$\bar{c}_1^+ = \tfrac{1}{2}\gamma(x)\eta^+(x)\|u_1\|, \quad \bar{c}_2^+ = 2.12\varepsilon_f(x)\eta^+(x)\|u_2\|$$

and $\gamma(x)$ the Lipschitz constant of condition 1.6 for some open neighbourhood of x. Using the same lower and upper bounds as in subsection 6.3.3 we define the approximated optimal step $\hat{h}^+$ by

(6.16) $$\hat{h}^+ = \begin{cases} 1, & \text{if } \hat{c}_1^+ < \hat{c}_2^+, \\ \varepsilon_h, & \text{if } \hat{c}_1^+ \times \varepsilon_h^2 \ge \hat{c}_2^+, \\ \sqrt{\hat{c}_2^+ / \hat{c}_1^+}, & \text{otherwise,} \end{cases}$$

where

(6.17) $$\hat{c}_1^+ = \tfrac{1}{2}\hat{\gamma}(x)\|u_1\|, \quad \hat{c}_2^+ = 2\varepsilon_f(x)\|u_2\|$$

and $\hat{\gamma}(x)$ is an approximation to $\gamma(x)$. Note that $\bar{c}_1^+ / \bar{c}_2^+$ does not depend on $\eta^+(x)$ so that we can choose $\bar{c}_1^+$ and $\bar{c}_2^+$ independent of $\eta^+(x)$. The initial approximation to $\gamma(x)$ is: $\hat{\gamma}(x_0) = 1$. We only give a formal description of the computation of the difference step. Computation of the difference approximation with this difference step has to be done with the routine diffjac (see subsection 6.3.3).

Description in ALGOL 68

```
.proc  calgh = .scal :
.pr  XDEF CALGH .pr
.begin  nrmul:= nrmu2:= zero; .for  i .to  n
   .do  .scal  aid:= (one + .abs  x[i]) ** 2;
      nrmul +:= aid; nrmu2 +:= (one / aid)
   .od ; nrmul:= sqrt(nrmul); nrmu2:=sqrt(nrmu2);
   .scal  cl:= nrmul * omga / two, c2:= nrmu2 * epf * two;
   h:= (.if  cl < c2 .then  one
        .elif  cl * epsh ** 2 >= c2 .then  epsh
        .else  sqrt(c2 / cl)
        .fi )
.end  # calculation of difference step in generalized case #
.pr  FEDX .pr ;
```

6.4. APPROXIMATION OF REQUIRED DATA

6.4.1. Introduction

Most algorithms described in this chapter use approximate values of quantities depending on function and method ($\bar{\omega}$, $\bar{\gamma}$, $\beta(x)$, $\kappa(x)$, $e(x)$ etc. (see notation 5.1)). The method of approximating these quantities is described in this section. We assume that F, $A \in L(\mathbb{R}^n)$ and $x_k (k=0,1,\dots)$ satisfy condition 1.10 or, if generalized methods are considered, condition 1.9. Moreover, we assume that $\bar{\omega}$, $\bar{\gamma} \neq 0$ and $F(x_k) \neq 0$ for $k = 0,1,\dots$.

We restrict attention to Newton-like algorithms with an approximation to the jacobian as described in section 6.3. So, only the two strict inverse-updating algorithms generate H_k explicitly. In these inverse-updating algorithms we do not use any of the approximate values described in this section, for reasons that will become clear in subsection 6.5.1. So we may restrict attention to algorithms using approximations to the jacobian without explicitly inverting these approximations. For simplicity we say that the processes considered in this section generate a sequence $\{(x_k,B_k)\}$ for a given starting point (x_0,B_0), instead of $\{x_k,H_k)\}$ for (x_0H_0). We have the relation

$$H_k = B_k^{-1} \quad \text{or} \quad H_k = B_k^{+}, \qquad k = 0,1,\dots,$$

depending whether classical or generalized inversion is used and assuming that in the first case the inverse exists.

In the remaining part of this section we describe the actual approximation to the values of the relevant quantities. We assume that for some $k \geq 0$ the sequence $\{(x_i,B_i)\}_{i=0}^{k}$ is generated by a given Newton-like process for starting point (x_0,B_0). We distinguish two cases

1. *Classical Newton-like methods.* We assume that condition 5.3 is satisfied and that approximations $\hat{\omega}_i$, $\hat{\beta}_i$, $\hat{\kappa}_i$, $\hat{e}_i$, $\hat{\eta}_i$ and $\|\hat{B}_i\|$ to $\omega(x_i)$ (and $\bar{\omega}$), β_i, κ_i, e_i, η_i and $\|B_i\|$ are given for $i < k$. Note that we use an approximation to $\bar{\omega}$ which depends on the iteration index. We shall pay attention to this point in subsection 6.4.2. We want to calculate $\hat{\omega}_k$, $\hat{\beta}_k$, $\hat{\kappa}_k$, $\hat{e}_k$, $\hat{\eta}_k$ and $\|\hat{B}_k\|$.
2. *Generalized Newton-like methods.* We assume that F, $A = I$ (see remark 4.33) and x_i $(i=1,\dots,k)$ satisfy condition 1.9 and that approximations $\hat{\gamma}_i$, $\hat{e}_i^{+}$,

$\hat{r}_i^+$, $\hat{\eta}_i^+$ and $\|\hat{B}_i\|$ to $\gamma(x_i)$ (and $\bar{\gamma}$), e_i^+, rank(B_i), η_i^+ and $\|B_i\|$ are given for $i < k$. We want to calculate $\hat{\gamma}_k$, $\hat{e}_k^+$, $\hat{r}_k^+$, $\hat{\eta}_k^+$ and $\|\hat{B}_k\|$.

The description in ALGOL 68 of these calculations has been given at the end of this section.

6.4.2. Approximation of ω_k

Let condition 1.5 be satisfied and $J(x)$ be nonsingular for some $x \in D$. Then $\omega(x)$ defined by

$$(6.18) \qquad \omega(x) = \sup_{\substack{y\in D\\ y\neq x}} \|J(x)^{-1}J(y) - I\| \;/\; \|y-x\|$$

can be used in condition 1.7 and if condition 1.8 is satisfied for F, x and A then $\bar{\omega}$ defined by

$$(6.19) \qquad \bar{\omega} = \sup_{y\in S_F(x,A)} \omega(y),$$

with $\omega(y)$ given by (6.18) can be used in condition 1.10. However, using (6.18) leads to a rather elaborate computation to obtain $\omega(x_k)$ and careful examination of the various applications yields easier and more efficient suggestions. We consider four possible situations in which an approximation to $\omega(x_k)$ or $\bar{\omega}$ is used.

1. Application of remark 5.9 to obtain an a-priori estimate of the step length factor. This application is based on theorem 5.4. In fact, the formulation of the results is such that in the k-th step we can approximate $\bar{\omega}$ by (6.19) with x replaced by x_k. However, from remark 5.6 we see that this would yield a too strong condition and that we can apply (5.28) and (5.29) to obtain a reasonable estimate to $\bar{\omega}$ at the k-th iteration step. First note that, with $x = x_{k-1}$, we have for $s = 1$ and small t (see (5.16)) $w(t,s) = z(t)$ and

$$p(t) - x_{k-1} = tJ_{k-1}^{-1}F_{k-1} + o(t) \approx tB_{k-1}^{-1}F_{k-1} + o(t).$$

Furthermore,

$$z(t) = x_{k-1} - tB_{k-1}^{-1}F_{k-1},$$

$$x_k = z(\lambda_{k-1}).$$

As $J_k^{-1}F_k$ is the direction in which a new iterate is searched for, we expect it to be a reasonable approximation to the direction $z(\lambda_{k-1}) - p(\lambda_{k-1})$. Assuming that e_k is small we approximate this direction by $B_k^{-1}F_k$. Now substituting $s = 1$ and $t = \lambda_{k-1}$ in (5.28) and (5.29) we obtain two lower bounds on the value of $\omega(x_k)$ to be computed

$$\hat{\omega}_k^0 = \frac{\| (B_{k-1}^{-1}B_k - I)B_k^{-1}F_k\|}{\|x_k-x_{k-1}\| \; \|B_k^{-1}F_k\|} = \frac{\|B_{k-1}\backslash F_k - B_k\backslash F_k\|}{\|x_k-x_{k-1}\| \; \|B_k\backslash F_k\|},$$

$$\hat{\omega}_k^1 = \frac{\| (B_k^{-1}B_{k-1} - I)B_{k-1}^{-1}F_{k-1}\|}{\|x_k-x_{k-1}\| \; \|B_{k-1}^{-1}F_{k-1}\|} = \frac{\|B_k\backslash F_{k-1} - B_{k-1}\backslash F_{k-1}\|}{\|x_k-x_{k-1}\| \; \|B_{k-1}\backslash F_{k-1}\|}.$$

We shall approximate $\hat{\omega}_k$ by

$$\hat{\omega}_k = \begin{cases} \max(\hat{\omega}_k^0, \hat{\omega}_k^1), & \text{if } B_k \neq B_{k-1}, \\ \hat{\omega}_{k-1}, & \text{if } B_k = B_{k-1}. \end{cases} \tag{6.20}$$

Notice that computing $\hat{\omega}_k$ only requires computation of $B_{k-1}\backslash F_k$ and $B_k\backslash F_{k-1}$ in addition to what has to be done anyhow. As decompositions of B_k and B_{k-1} are available, the number of basic arithmetical operations to compute $\hat{\omega}_k$ is of order n^2. Furthermore, if $B_k = B_{k-1}$ (fixed approximation) then $\hat{\omega}_k^0 = \hat{\omega}_k^1 = 0$.

2. Application of theorem 3.16 to obtain an a-priori upper bound on the error in the jacobian approximation if update approximation is used (see corollary 5.15). As follows from the theorem we need in the k-th iteration step (k=1,2,...) an approximation to $\omega(x_k)$ on the set $D_0 = \{z | z = x_k - tB_k^{-1}F_k,\ t \in [0,1]\}$. As B_{k+1} is not available the approximation given by (6.20) seems to be a reasonable alternative.

3. Application of theorem 3.19 to obtain an a-priori upper bound on the error in the fixed jacobian approximation (corollary 5.17). For this case we can make the same observations as for the second case above.

4. Application of theorem 3.5, in order to obtain an a-priori upper bound on the error in the difference approximation to the jacobian, and (6.10) to compute the optimal difference step. For these cases (6.19) can only be simplified to

$$\omega(x_k) = \sup_{\substack{y\in U(x_k,\delta)\\ y\neq x_k}} \|J_k^{-1}J(y) - I\|/\|y-x_k\|,$$

for $\delta > 0$ such that $(\sum_{i=1}^{n} h_i^2)^{\frac{1}{2}} < \delta$. If $x_{k-1} \in U(x,\delta)$, $B_{k-1} = J_{k-1}$ and $B_k = J_k$ then $\hat{\omega}_k^1$ is a lower bound on the desired value.

The above cases suggest to use $\hat{\omega}_k$ given by (6.20) except for application 4. Anyhow it is about the best we can with the information available. For the same reason, although we may expect $\hat{\omega}_k$ to be too small in the fourth case, we also use $\hat{\omega}_k$ in this case. Moreover, as calculation of $\hat{\omega}_k$ is cheap, its use is very attractive.

As is shown in section 6.6 we shall also use $\hat{\omega}_k$ in two other applications. One, a failure criterion, is based on theorem 5.7 and therefore the same arguments as in case 1 hold for use of $\hat{\omega}_k$. The other is a convergence criterion based on theorem 5.18. The usefulness of $\hat{\omega}_k$ in this case is discussed in section 6.6.

6.4.3. Approximation of γ_k

The value of $\hat{\gamma}_k$ is used for obtaining an upper bound on the error e_k^+ in the difference approximation in generalized difference Newton methods and for computing an optimal difference step according to (6.16). Furthermore, it is used in the condition for fixed approximation (corollary 5.17). Except for the last application we need an estimate to

$$\gamma_k = \sup_{\substack{x\in U(x_k,\delta)\\ y\neq x_k}} \|J(y) - J(x_k)\|/\|y-x_k\|.$$

We like to avoid computation of additional jacobian approximations. Therefore we use B_k and B_{k-1} yielding

$$\|B_{k-1}-B_k\|/\|x_{k-1}-x_k\|$$

as an approximation. To compute a norm of $B_{k-1}-B_k$ requires, however, storing of both B_{k-1} and B_k. Therefore, we simplify this expression even more yielding

$$(6.21)\qquad \hat{\gamma}_k = \|B_{k-1}v-B_kv\|/\|x_{k-1}-x_k\|,$$

where $v = v_1/\|v_1\|$, with elements of v_1 randomly chosen in $[-1,+1]$.

6.4.4. Approximation of β_k

To obtain an approximation to β_k one may choose several $x \in S_F(x_k,A)$ and evaluate $\|B(x)\backslash F_k\|$, where $B(x)$ is an approximation to $J(x)$. We obtain an approximation to β_k by taking the maximum of the values obtained. However, this requires evaluation of an approximation to $J(x)$ at other points than the iteration points, which is highly unattractive. Therefore, we use only B_k, the approximation to $J(x_k)$. So

$$(6.22)\qquad \hat{\beta}_k = \|B_k\backslash F_k\|.$$

If $J(x)$ is reasonably smooth on $S_F(x_k,A)$ and its condition number is not large relative 1, then $\hat{\beta}_k$ is a good approximation to β_k.

6.4.5. Approximation of κ_k

By notation 5.1 we have

$$(6.23)\qquad \kappa_k = \|AJ_k\|\ \|J_k^{-1}F_k\|/\|AF_k\|.$$

An approximation to κ_k is used for estimating an a-priori step length factor using remark 5.9 and for some stopping criteria. We distinguish three typical choices for the matrix A in the level function (see 4.23). In fact, this defines three typical ways of implicit scaling (see remark 4.18).

1. $A = I$.
   No implicit scaling is performed and restrained methods with such a choice are not affine invariant. Replacing J_k by B_k in (6.23) yields as an approximation to κ_k

$$(6.24)\qquad \hat{\kappa}_k = \|B_k\|\ \|B_k\backslash F_k\|/\|F_k\|.$$

Using the perturbation lemma 1.13 and the definition of e_k it is easy to show that

$$(6.25) \qquad \frac{1-e_k}{1+e_k}\kappa_k \le \hat{\kappa}_k \le \frac{1+e_k}{1-e_k}\kappa_k.$$

Hence, if e_k is small then $\hat{\kappa}_k$ is a good approximation to κ_k.

2. $A = B_0^{-1}$.

   Implicit scaling with the inverse of the jacobian approximation at the initial guess. This choice satisfies condition (4.24) for all nonsingular matrices $T \in L(\mathbb{R}^n)$. Hence affine invariant restrained methods can be constructed with this choice. Replacing J_k by its approximation B_k in (6.23) yields an estimate $\bar{\kappa}_k$ to κ_k:

$$\bar{\kappa}_k = \|B_0^{-1}B_k\| \ \|B_k^{-1}F_k\|/\|B_0^{-1}F_k\|.$$

   For this estimate (6.25) holds with $\hat{\kappa}_k$ replaced by $\bar{\kappa}_k$. Hence $\bar{\kappa}_k$ is a good approximation to κ_k for small e_k. However, direct computation of $\|B_0^{-1}B_k\|$ requires $o(n^3)$ basic arithmetical operations. Therefore we approximate this norm yielding

$$(6.26) \qquad \hat{\kappa}_k = \|B_0\backslash(B_k v)\| \ \|B_k\backslash F_k\|/\|B_0\backslash F_k\|,$$

   where $v = v_1/\|v_1\|$ and v_1 a vector with elements randomly chosen in $[-1,+1]$.

3. $A = B_k^{-1}$.

   With this choice we can also construct affine invariant restrained algorithms. Replacing J_k by B_k in (6.23) yields as an approximation to κ_k:

$$(6.27) \qquad \hat{\kappa}_k = 1.$$

   Note however, that the global convergence result 5.8 is not applicable to methods with variable A. Finally, $\hat{\kappa}_k$ satisfies (6.25).

6.4.6. Approximation of $\|B_k\|$, η_k and η_k^+

For several matrix norms which are compatible with a vector norm, it requires only $o(n^2)$ arithmetical operations to calculate that norm of a matrix. E.g. the infinity-, one- or Frobenius norms. One might choose $\|\hat{B}_k\|$ equal to one of these norms of B_k. If a singular value decomposition of B_k is available, hence in all generalized algorithms, we simply choose

$$\|\hat{B}_k\| = \sigma_1,$$

where σ_1 is the largest singular value of B_k. For simplicity we choose in the other algorithms

$$\|\hat{B}_k\| = \max_{1\le i\le n} \left(\sum_{j=1}^{n} B_{ij}^2\right)^{\frac{1}{2}},$$

because the operator .ludec as well as the procedure used for triangular decomposition in our ALGOL 60 experiments yield this value as an auxiliary result. For other programming systems other choices may be more attractive.

The approximation of η_k is more complicated. In fact a computation of $\|B_k^{-1}\|$ requires explicit computation of B_k^{-1}, also for other norms than the spectral norm. This is very unattractive. Therefore, we use the approximation

$$(6.28) \qquad \hat{\eta}_k = \|B(x_k)\backslash v\|,$$

where $v = v_1/\|v_1\|$ with v_1 a vector with elements chosen randomly in $[-1,+1]$.

Finally, approximation of $\hat{\eta}_k^+$ is at hand:

$$(6.29) \qquad \hat{\eta}_k^+ = 1/\sigma_{\hat{r}_k^+},$$

with $\hat{r}_k^+$ the approximated rank of B_k and, therefore, $\sigma_{\hat{r}_k^+}$ the smallest nonzero singular value.

6.4.7. Approximation of e_k and e_k^+

Approximation of e_k and e_k^+ depend on the choice of the approximation of the jacobian. We distinguish the basic choices of section 6.3.

Inverse of analytic jacobian.
Backward analysis of the triangular decomposition process and considering numerical errors in the computation of $J(x)$ (see (6.4)) yields that we, in fact, approximate $J(x)$ by

(6.30) $\quad B = J(x) + E_1 + E_2$,

where, with use of the approximate upper bound on $\|E_2\|$ given in (6.7),

(6.31) $\quad \|E_1\| \le \varepsilon_{rj} \|J(x)\|+\varepsilon_{aj}; \ \|E_2\| \le 16\varepsilon n\|B\|$.

So, e_k as defined in (5.4) is approximated by its approximate upper bound

(6.32) $\quad \hat{e}_k = \hat{\eta}_k((\varepsilon_{rj}+16\varepsilon_n)\|\hat{B}_k\|+\varepsilon_{aj})$.

Inverse of difference approximation.

We can use formula (6.9) to approximate the error due to difference approximation and (6.7) to approximate the error due to triangular decomposition. Including both errors, the inverse jacobian is in fact approximated by $(B_k+E_2)^{-1}$ with E_2 satisfying (6.31). Using lemma 1.13 (formula (1.6), with $A = B_k+E_2$, $B = B_k$) we obtain for the error, including round off

(6.33) $$\|(B_k+E_2)^{-1}J_k - I\| \le e_k' + \frac{\|B_k^{-1}\| \ \|E_2\|}{1-\|B_k^{-1}\|\|E_2\|}(1+e_k'),$$

where e_k' is the upper boud on the error due to difference approximation (see(6.9)). Therefore, an approximation to the error may be

(6.34) $$\hat{e}_k = \bar{e}_k + \frac{16n\varepsilon\hat{\eta}_k\|\hat{B}_k\|}{1-16n\varepsilon\hat{\eta}_k\|\hat{B}_k\|}(1+\bar{e}_k)$$

with
$$\bar{e}_k = \frac{\hat{c}_2\hat{h}^{-1} + \hat{c}_1\hat{h}}{1 - \hat{c}_1\hat{h}}$$

and $\hat{c}_1$ and $\hat{c}_2$ given by (6.11). In our ALGOL 60 experiments we neglected the second term in the right hand side of (6.34). Only in very exceptional cases ((almost) linear functions and (almost) exact computation of the function, so that $\hat{\omega}_k$ and $\varepsilon_f(x_k)$ are (almost) equal to zero) this term is not negligible relative to the first one. Moreover, only large values of $\hat{e}_k$ (relative to ε) influence the Newton-like processes.

Inverse-updating approximation.

If we use inverse-updating in every step, then the error bound based on theorem 3.16 increases in every step. In fact, this bound may become so large that it is useless for the applications we have in mind (for restraining strategies (see section 6.5) or failure criteria). Therefore, in the basic inverse-updating algorithms we do not use an approximation to e_k. Only if updating is performed conditionally we calculate such an approximation.

Generalized inverse of analytic jacobian

Using (6.4) and the bound on the error due to round-off during the singular value decomposition (see subsection 6.2.2) we take as an approximation to e_k^+ (cf. (6.2))

$$\hat{e}_k^+ = \hat{\eta}_k((\varepsilon_{rj}+n\varepsilon)\|\hat{B}_k\|+\varepsilon_{aj}). \tag{6.35}$$

Generalized inverse of difference approximation.

We can use (6.15) to approximate the error due to difference approximation and the bound given in subsection 6.2.2 due to singular value decomposition. So we take as an approximation to e_k^+:

$$\hat{e}_k^+ = (\hat{c}_1^+\hat{h}_k^+ + \hat{c}_2^+(\hat{h}_k^+)^{-1} + n\varepsilon\|\hat{B}_k\|)\hat{\eta}_k^+, \tag{6.36}$$

with $\hat{c}_1^+$ and $\hat{c}_2^+$ given by (6.17). In our ALGOL 60 experiments we neglected $n\varepsilon\|\hat{B}_k\|$ relative to the other terms, as only in very exceptional cases ($\hat{c}_1^+$ and $\hat{c}_2^+$ are (almost) equal to zero) this term is not negligible.

In all cases we use $1-\varepsilon$ as an upper bound on the approximated error in order to avoid arithmetic overflow in some parts of the algorithms.

6.4.8. Approximation of the rank of B_k

We distinguish between algorithms using triangular decomposition and algorithms using singular value decomposition. In the first case we are only interested in a possible breakdown of the process of triangular decomposition, in the latter we need an approximation to the rank of B_k.

Triangular decomposition.

We say that B_k is singular if the process of triangularization breaks down with $\varepsilon_{rk} = \varepsilon$ (see subsection 6.2.1.), otherwise B_k is nonsingular (rank equals n). In section 6.6 we describe some failure criteria which relate singularity of B_k to the error in B_k as an approximation to J_k.

Singular value decomposition

If B_k is nonsingular then $e_k^+ = \|J_k B_k^{-1} - I\|$ and $e_k^+ < 1$ guarantees that J_k is nonsingular. We use this condition to approximate the rank r_k of B_k. In fact, we say that B_k is singular if the error bound is so large that nonsingularity of J_k cannot be guaranteed. Let $\Sigma = \mathrm{diag}(\sigma_1,\ldots,\sigma_n)$ be the diagonal matrix of singular values of B_k. Notice that we have defined $\hat{\eta}_k^+ = 1/\sigma_{\hat{r}_k^+}$ (see (6.29)). Hence, the condition $\hat{e}_k^+ < 1$, with $\hat{e}_k^+$ given by (6.35) or (6.36), yields the following definition of $\hat{r}_k^+$.

<u>6.2. DEFINITION</u>. The approximated rank $\hat{r}_k^+$ of B_k is the largest integer less than or equal to n such that

(6.37) $$\sigma_{\hat{r}_k^+} > C_k,$$
where
(6.38) $$C_k = (\varepsilon_{rj} + n\varepsilon)\|\hat{B}_k\| + \varepsilon_{aj}$$

if the analytic jacobian is used and

(6.39) $$C_k = \hat{c}_1^+ \hat{h}_k^+ + \hat{c}_2^+ (\hat{h}_k^+)^{-1} + n\varepsilon\|\hat{B}_k\|$$

if the difference approximation is used.

6.4.9. Description in ALGOL 68

We distinguish between calculation of required data in nongeneralized algorithms (.proc cdatalr) and in generalized algorithms (.proc cdatasv). In these routines we also perform the decomposition of the matrix and compute the direction of search. So, in those algorithms in which no data as described in this section is required (e.g. inverse-updating algorithms (see section 6.7)) we replace a call of one of these routines by the statements:

```
dx := -b * f; nrmdx := .nrm dx; f0 := f;
```

```
.proc  cdatalr = .void :
.pr  XDEF CDATALR .pr
.begin  .if  it = 1 .then  decb:= .heap .lud := .ludec  b .fi ;
   .case  decb .in  (.ref .lud  blu):
   .begin  .if  fix .then  dx:= -(blu .sol  f); beta:= .nrm  dx
      .elif  it = 1
      .then  dcnt +:= 1; .if  .not  .check  blu
         .then  torrix(warning,text5); ready .fi ;
         dx:= -(blu .sol  f); beta:= .nrm  dx; omga:= one;
         .if  c .of  a > 1
         .then  deca .of  a := blu; slevel:= beta
         .else  slevel:= nrmf .fi ; xj:= x; f0:= f
      .else  .vec  d01 = blu .sol  f; blu := .ludec  b;
         dcnt +:= 1; .if  .not  .check  blu
         .then  torrix(warning, text5); ready .fi ;
         .if  c .of  a = 2 .then  deca .of  a := blu .fi ;
         .scal  oml:= .nrm ((blu .sol  f0) * labda + dx);
         dx:= -(blu .sol  f); beta:= .nrm  dx;
         .scal  om:= .nrm (d01 + dx);
         oml /:= (nrmdx ** 2); om /:= (nrmdx * beta);
         omga:= om .max  oml; xj:= x; f0:= f
      .fi ; nrmdx:= beta;
      kappa:= .case  c .of  a .in
         nrm .of  blu * nrmdx / slevel,
         .nrm (deca .of  a .sol  (b * v)) * nrmdx / slevel,
         1 .esac ;
      eta:= .nrm (blu .sol  v); .if  it = 1 .then  eta0:= eta .fi ;
      .scal  aid:= eta * nrm .of  blu * n * .widen  16 * small scal;
      .if  anl
      .then  e:= ej:= (eprj * nrm .of  blu + epaj) * eta + aid
      .elif  dif
      .then  e:= ej:=
         (.scal  c1:= nrmu1 * omga / two,
                 c2:= nrmu2 * eta * epf * two;
          c2:= c2 / h + c1 * h; c1:= one - c1 * h;
          c2:= (c1 < c2 ! one ! c2 / c1);
          (aid * two > one ! one !
           c2 + (one + c2) * aid / (one - aid)))
      .fi ; e:= e .min  (one - small scal);
   ready: .skip
   .end  .esac
.end  #computing data and step direction in case of lu decom #
.pr  FEDX .pr ;
```

```
.proc  cdatasv = .void :
.pr  XDEF CDATASV .pr
.begin  .if  it = 1 .then  decb:= .heap .svd := .svdec  b .fi ;
   .case  decb .in  (.ref .svd  bsv):
   .begin  .if  .not  fix
      .then  omga:= (it = 1 ! w:= b * v; one
                   ! .nrm (w - (w:= b * v)) / nrmdx);
         .if  it > 1 .then  bsv:= .svdec  b .fi ;
         dcnt +:= 1; xj:= x; .if  .not  .check  bsv
         .then  torrix(warning, text6); ready .fi
      .fi ;
      .scal  maxval:= .max  sngval .of  bsv;
      .scal  aid:= maxval * n * small scal;
      .scal  ck:= (anl ! maxval * eprj + epaj + aid
                 !: dif ! nrmul * omga * h / two
                    + nrmu2 * epf * two / h + aid);
      .if  it = 1 .then  slevel:= nrmf .fi ;
      .if  anl .or  dif
      .then  bsv:= ck .trims  bsv;
         .int  rk = .upb  sngval .of  bsv;
         eta:= one / (sngval .of  bsv)[rk];
         e:= ej:= (ck * eta) .min  (one - small scal)
      .fi ;
      nrmdx:= .nrm  (dx:= -(bsv .sol  f));
      kappa:= maxval * nrmdx / slevel;
      .if  .nrm  (u .of  bsv * f) < epf
      .then  torrix(warning, text3) .fi ;
   ready: .skip
   .end  .esac
.end  # computing data and step direction in case of sv dec #
.pr  FEDX .pr ;
```

6.5. RESTRAINING STRATEGY

6.5.1. Introduction

In this section we shall describe three possible restraining strategies together with the module used in strict algorithms to calculate a new point and its function value. Define the level function (cf. (4.27)):

(6.40) $$\phi(t) = \|AF(z(t))\|^2, \quad t \in [0,1],$$

where $z(t) = x-tHF(x)$ and $A \in L(\mathbb{R}^n)$ nonsingular. We can only guarantee the existence of $t \in [0,1]$ such that $\phi(t) < \phi(0)$ (cf. (4.23)), if the step direction $(-HF(x))$ is a descent direction for $\|AF(y)\|$ at $y = x$. Assuming that F and x satisfy condition 1.5, $F(x) \neq 0$ and $J(x)$ is nonsingular, lemma 4.19 states a sufficient condition for $-HF(x)$ being a descent direction (see also condition 5.3 (vi)), viz.

(6.41) $$e(x)\kappa(x) < 1.$$

Therefore, in our non-generalized restrained Newton-like algorithms we use such a condition (see section 6.6) in order to guarantee the existence of a step length factor.As is already mentioned in subsection 6.4.7 the approximate error in inverse-updating algorithms is increasing in every step and may easily become too large. Therefore, we have no good way to quarantee existence of a step length factor in inverse-updating algorithms. As a consequence, we shall not consider restrained inverse-updating algorithms. In the generalized algorithms nonsingularity of the jacobian is not quaranteed. In these algorithms we simply try to find an appropriate step length factor and terminate the process if we can not find one. Note that we use approximations to κ_k and e_k which may be bad. Therefore, condition (6.41) is not sufficient to guarantee existence of a step length factor. We shall use an additional failure criterion in order to avoid looping during restraining, which particularly may occur if numerical errors in the function are large relative to the required precision. This criterion is based on (6.3). Let t_1, t_2 and $t_3 \in [0,1]$ be three successive attempted values for the step length factor, obtained by one of the strategies described in subsections 6.5.3, 6.5.4 and 6.5.5), then the algorithm is terminated if

(6.42) $$|\sqrt{\phi(t_i)} - \sqrt{\phi(t_{i+1})}| < \varepsilon_f(x), \quad i = 1,2.$$

We choose $\varepsilon_f(x)$ (see (6.9)) instead of $\varepsilon_{rf}\|F(x)\|+\varepsilon_{af}$, just for ease of programming. This value is available in most algorithms and the essential effect is the same. Inequality (6.42) has to hold for three successive values, as for two values it might occasionally hold if $z(t_1)$ and $z(t_2)$ lie on different sides of a valley of $\phi(t)$. Of course, there is still a chance that a reasonable p can be found although (6.42) is satisfied, but we think that this chance is small enough to be negligible. Due to (6.42) and the fact that $fl_\varepsilon(\phi(t)) = fl_\varepsilon(\phi(0))$ for t small enough, the restraining process is always terminating if successive values for the step length factor are decreasing to zero.

Finally we give an ALGOL 68 description of a routine, which is used in all algorithms. This routine calculates the function value and the square root of the value of the level function. If the argument vector is out of the domain of the function then max scal is delivered.

Description in ALGOL 68

```
.proc  slefu = (.ref .vec  x, f).scal :
.pr  XDEF SLEFU .pr
.begin  .func  fu = fun(x); fcnt +:= 1; .if  in .of  fu
   .then  f:= f .of  fu;
      (c .of  a = 1 ! .nrm  f ! .nrm (deca .of  a .sol  f))
   .else  max scal .fi
.end  #computing function and level function #
.pr  FEDX .pr ;
```

6.5.2. No restraining.

We choose $\lambda(x,H) = 1$, for all $(x,H) \in D \times L(\mathbb{R}^n)$.

Description in ALGOL 68

```
.proc  strict = .void :
.pr  XDEF STRICT .pr
.begin  x:= x + dx; slevel:= slefu(x, f);
   .if  slevel = max scal .then  torrix(warning, text11)
   .else  nrmf:= .nrm  f .fi
.end  # strict step with function evaluation at new point #
.pr  FEDX .pr ;
```

6.5.3. Bisection.

Let $(x,H) \in D \times L(\mathbb{R}^n)$. Suppose there exists an integer $p \geq 0$ such that

$$(6.43) \qquad p = \min(i \mid i \in N,\ \phi(2^{-i}) < \phi(0)),$$

where N denotes the set of nonnegative integral numbers and ϕ is defined by (6.40). Then we choose

$$\lambda(x,H) = 2^{-p}.$$

p is calculated by trying subsequently $p = 0,1,2,\ldots$.
Note that definition of the level function by .proc slefu is such that, if $z(t)$ lies outside the domain D of F for some $t = 2^{-i} \in [0,1]$, then the process is not terminated, but bisection is continued until a value is found which yields a point in the domain. Such a value exists as $x \in D$ and D is open.

Description in ALGOL 68

```
.proc  resbis = .void :
.pr  XDEF RESBIS .pr
.begin  .scal  nrmfl:= zero, slevell:= slevel;
   .int  i:= 0; labda:= two;
   .vec  xl, fl;
   .while
      slevell >= slevel .and
      (.abs (nrmf - nrmfl) <= epf
      ! (i = 2 ! .false  ! i:= 2; .true )
      ! i:= 0; .true ) .or  nrmfl  = max scal
   .do  labda /:= two; xl:= x + labda * dx;
      slevell:= slefu(xl, fl);
      nrmfl:= (slevell = max scal ! max scal ! .nrm  fl)
   .od ;
   x:= xl; dx:= labda * dx; f:= fl; nrmdx *:= labda;
   nrmf:= nrmfl; slevel:= slevell; .if  i = 2
   .then  .if  anl .or  dif
      .then  torrix(warning, text2)
      .else  e:= ej:= one - small scal; reset number of warnings
   .fi  .fi
.end  # restraining with bisection and evaluation of function at
        new point #
.pr  FEDX .pr ;
```

N.B. If (6.42) is satisfied and conditional updating or fixed approximation is used (.not(anl .or dif)) then the process is not terminated. We first try an iteration step with calculation of the analytic or difference jacobian, which is automatically induced in such algorithms if e and e_j are set to $1 - \varepsilon$ (see section 6.8 and 6.9).

6.5.4. Interpolation.

Let be given $(x,H) \in D \times L(\mathbb{R}^n)$. By successive quadratic interpolation we may determine $t_m \in [0,1]$ such that $\phi(t_m) < \phi(0)$ and set

$$\lambda(x,H) = t_m.$$

In our algorithms we first test whether the condition $\phi(1) < \phi(0)$ is satisfied. If this is true then $\lambda(x,H) = 1$; otherwise we use quadratic interpolation starting from the points $t = 0$, $t = 0.5$ and $t = 1$, to find a value $t_m \in (0,1)$ satisfying $\phi(t_m) < \phi(0)$. In the interpolation algorithm, which is defined by .proc quad and .proc interp, we use the fact that, for t small enough, $\phi(t) < \phi(0)$ has to be satisfied. Interpolation is performed subsequently, with $t = 0$ and the two smallest approximations to t_m found so far. As soon as a value is found for which the value of the levelfunction is less than $\phi(0)$ we choose t_m equal to this value. We use $(\varepsilon_{rx}\|x\|+\varepsilon_{ax})/\|HF(x)\|$ as a lower bound on the value of $\lambda(x,H)$. Two successive values differ at least with this value. If the lower bound is reached without finding an appropriate value, then the process is terminated.

Description in ALGOL 68

```
.proc  quad = (.scal  v, fv, w, fw, z, fz).scal :
.pr  XDEF QUAD .pr
# it is assumed that fz is the smallest function value #
.begin  .scal  r:= (z - w) * (fz - fv), q:= (z - v) * (fz - fw);
   .scal  p:= (z - v) * q - (z - w) * r; q:= (q - r) * two;
   .if  q > zero .then  p:= -p .else  q:= -q .fi ;
   (p >= (one - z) * q ! one !: p <= -z * q ! zero ! p / q)
.end  # one quadratic interpolation step #
.pr  FEDX .pr ;
```

```
.proc  interp = (.ref .scal  f0, x, fx, b, fb,
                  .proc (.scal ).scal  fun, .scal  tol).bool :
.pr  XDEF INTERP .pr
# it is assumed that f0 = fun(0) and if interp delivers .true  then
 x is such that fun(x) < fun(0) #
.begin  .scal  u; .bool  bl;
   .while  .if  b > tol * two .then  .not   (bl:= fx < f0)
      .else  bl:= .false  .fi
   .do  u:= quad(b, fb, x, fx, zero, f0);
      .if  u < tol .then  u:= tol
      .elif  u > x - tol
      .then  u:= x / two
      .fi ;
      b:= x; fb:= fx; x:= u; fx:= fun(u)
   .od ; bl
.end  #interpolation with quadratic formula until function value
        less than fun(0) is obtained or interval becomes too small #
.pr  FEDX .pr ;

.proc  resint = .void :
.pr  XDEF RESINT .pr
.begin
   .proc  fu = (.scal  labda).scal :
   .begin  .vec  xl:= x + labda * dx; slevell:= slefu(xl, fl);
      nrmfl:= (slevell = max scal ! max scal ! .nrm  fl);
      (.abs (nrmf - nrmfl) > epf .or  nrmfl = max scal
      ! i:= 0 !: i = 2 ! ready ! i:= 2);
      (smr .min  slevell) ** 2
   .end  # function to be interpolated #;

   .int  i:= 0; .scal  slevell, nrmfl, smr:= sqrt(max scal);
   .vec  fl; labda:= one;
   .scal  level:= (smr .min  slevel) ** 2, levell:= fu(one);
   .if  levell >= level
   .then  labda /:= two;
      .scal  b:= one, fll:= levell, flb:= fu(labda);
      .if  .not  interp(level, labda, flb, b, fll, fu,
                        (nrmx * dlrx + dlax) / nrmdx)
      .then  torrix(warning, textl)
      .fi
   .fi ;
ready: dx *< labda; x +:= dx; nrmdx *:= labda; f:= fl;
   nrmf:= nrmfl; slevel:= slevell; .if  i = 2
   .then .if  anl .or  dif
      .then  torrix(warning, text2)
      .else  e:= ej:= one - small scal; reset number of warnings
      .fi
   .fi
.end  # restraining with interpolation and evaluation of function at
         new point #
.pr  FEDX .pr ;
```

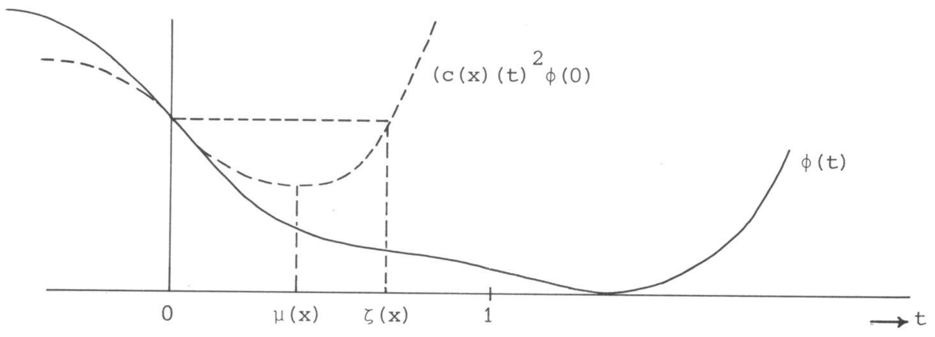
6.5.5. <u>A-priori estimation and control</u>.

Let be given $(x,H) \in D \times L(\mathbb{R}^n)$. Assume that the conditions of theorem 5.4 are satisfied. Then, with ϕ defined by (6.40) and use of notation 5.1, we have

$$\phi(t) \le (c(x)(t))^2\phi(0) \qquad t \in [0,\min(1,\zeta(x))], \tag{6.44}$$

Note that $c(x)(\zeta(x)) = 1$ and $c(x)(t)$ is minimal on $[0,\zeta(x)]$ for $t = \mu(x)$. The situation is illustrated by figure 6.1.

figure 6.1.



Denote

$$\phi_{opt} = (c(x)(\mu(x)))^2\phi(0).$$

Then

$$\phi(\mu(x)) \leq \phi_{opt}.$$

A possible strategy for choosing $\lambda(x,H)$ would be to choose it equal to $\min(1,\mu(x))$ as it gives the best upper bound on the function $\phi(t)$. However, $(c(x)(t))^2\phi(0)$ is locally a good approximation of $\phi(t)$ but little is known when t is large. As figure 6.1 suggests, situations are imaginable in which $\mu(x)$ is a rather pessimistic choice for the step length factor and practical experience shows that repeated underestimation of the step length factor may result in slow convergence. Therefore, we will choose an initial estimate to $\lambda(x,H)$ based on the estimation of $\phi(t)$ by $(c(x)(t))^2\phi(0)$, but we use the difference between the $\phi(t)$ and its approximation $(c(x)(t))\phi(0)$ to decide whether we accept this value or choose a larger or smaller value. Then, no extra function evaluations are required if the first estimate to $\lambda(x,H)$ is accepted. Before describing the proposed strategy notice that

$$\phi'(0) = -2[AJ(x)HF(x),AF(x)].$$

Therefore, $\phi'(0)$ may be approximated by

$$\hat{\phi}' = -2\|AF(x)\|^2 = -2\phi(0).$$

Quadratic interpolation with $(0,\phi(0),\phi'(0) \approx \hat{\phi}')$ and $(\zeta,\phi(\zeta))$ yields an optimal value

$$q_0 = \frac{\zeta^2\phi(0)}{(\phi(\zeta)-\phi(0)+2\phi(0))},$$

where we have a minimum at q_0 if the denominator is positive. Denote $\hat{\xi} = \min(1,\hat{\zeta}(x))$ and $\hat{\zeta}(x)$ and ϕ_{opt} are the approximated values of $\zeta(x)$ and ϕ_{opt} obtained by using the approximations from section 6.4. We use $\hat{\xi}$ as a first estimate to $\lambda(x,H)$. If $\phi(\hat{\xi})$ is not small enough relative to $\phi(0)$ and $\hat{\phi}_{opt}$ (i.e. what can be obtained approximately) then we search for a smaller value using interpolation. If, however, $\phi(\hat{\xi})$ is much less than expected (if $\phi(\hat{\xi}) < \hat{\phi}_{opt}$; we expect $\phi(\hat{\xi}) = \phi(0)$), then we search for a larger step length factor by extrapolation. We define

$$(6.45)\qquad \lambda(x,H) = \begin{cases} \hat{\xi}, & \text{if } \hat{\phi}_{opt} \le \phi(\hat{\xi}) < \tfrac{1}{2}(\hat{\phi}_{opt}+\phi(0)) \\ & \text{or } \hat{\xi} = 1 \text{ and } \hat{\phi}_{opt} > \phi(\hat{\xi}) \\ & \text{or } q_0 \in (0,\hat{\xi}), \\ q_i(0,\hat{\xi}/2,\hat{\xi}), & \text{if } q(\hat{\xi}) \ge \tfrac{1}{2}(\hat{\phi}_{opt}+\phi(0)), \\ q_e(0,\hat{\xi}, \max(2\hat{\xi},\min(q_0,1))), & \\ & \text{if } \hat{\xi} \ne 1 \text{ and } q_0 \notin (0,\hat{\xi}) \text{ and } \hat{\phi}_{opt} > \phi(\hat{\xi}), \end{cases}$$

where $q_i(0,\hat{\zeta}/2,\hat{\xi})$ is the first value less than or equal to $\hat{\xi}/2$, obtained by quadratic interpolation as described by .proc interp in subsection 6.5.3, in which the value of the level function is less than $\phi(0)$ and $q_e(0,\hat{\xi},q_0)$ is a value obtained by quadratic extrapolation, on $(0,1]$. In fact, quadratic extrapolation is performed as long as the level function in the extrapolated values decreases and the distance between successive extrapolated values is not decreasing, q_e is precisely described by .proc extrap.

Note that we always have $\lambda(x,H) \le 1$. Although it may be sensible to choose $\lambda(x,H) > 1$ for some (x,H), we will not do so. At least asymptotically $\lambda(x,H) = 1$ is a good value (see theorems 5.8 and 5.23) as convergence of such a generated sequence of iterates will be quadratic for appropriate jacobian approximations. It is not clear how to choose $\lambda(x,H) \ge 1$ in such a way that this asymptotic convergence behaviour is preserved.

Description in ALGOL 68

```
 .proc  resest = .void :
 .pr  XDEF RESEST .pr
 .begin
    .proc  fu = (.scal  labda).scal :
    .begin  .vec  xl:= x + labda * dx; slevell:= slefu(xl, fl);
       nrmfl:= (slevell = max scal ! max scal ! .nrm  fl);
       (.abs (nrmf - nrmfl) > epf .or  nrmfl = max scal
       ! i:= 0 !: i = 2 ! ready ! i:= 2);
       (smr .min  slevell) ** 2
    .end  # function to be interpolated #;

    .int  i:= 0; .scal  slevell, nrmfl, smr:= sqrt(max scal);
    .vec  fl;
    .scal  level:= (smr .min  slevel) ** 2, fiopt;
    .scal  el:= e + one, t:= (dlrx * nrmx + dlax) / nrmdx;
    .scal  nu0:= kappa * el / two, nul:= (el * e * two + one) / el;
    .scal  nu2:= (one - kappa * e) / nu0;
    .if  nu2 < 0 .then  labda:= t; fiopt:= level
    .else  .scal  aidl:= nul * nul, alfa:= omga * beta;
       .scal  three = .widen  3;
       .scal  aid2 = (sqrt(aidl + nu2 * three) - nul) / three,
              aid3 = (sqrt(aidl + nu2 * two * two) - nul) / two;
       .scal  labdaopt = (aid2 >= alfa ! one ! aid2 / alfa);
       labda:= (aid3 >= alfa ! one ! aid3 / alfa);
       aidl:= alfa * labdaopt;
       fiopt:= (it = 1 ! one
               ! one + nu0 * labdaopt * ((aidl + nul) * aidl - nu2))
                 ** 2 * level
    .fi ;
    .scal  levele:= fu(labda), labdae:= labda;
    .if  levele >= (fiopt + level) / two
    .then  labda /:= two; .scal  flb:= fu(labda);
       .if  .not  interp(level, labda, flb, labdae, levele, fu, t)
       .then  torrix(warning, textl)
       .fi
    .elif  levele < fiopt .and  labda /= one
    .then  .vec  f2 = genvec(n);
       .proc  fud = (.scal  labda).scal :
       .begin  f2:= fl; slevel2:= slevell; fu(labda) .end ;

       .scal  slevel2, aid:= labda * level * two;
       .scal  p:= aid * labda, q:= (levele - level + aid) * two;
       aid:= q * labdae; .scal  labda2:= (labda * two) .min  one;
       .if  q >= zero
       .then  .if  p > q .then  labda2:= one
          .elif  p < aid .then  labda2:= labdae
          .elif  p >= aid * two .then  labda2:= p / q .fi
       .fi ;
       .if  .not  extrap(level, labda, levele, labda2, fud)
       .then  fl:= f2; slevell:= slevel2 .fi
    .fi ;
 ready: dx *< labda; x +:= dx; nrmdx *:= labda; f:= fl;
    nrmf:= nrmfl; slevel:= slevell; .if  i = 2
    .then  torrix(warning, text2) .fi ;
    .if  (fix .or  upd) .and  number of warnings > 0
    .then  e:= ej:= one - small scal; reset number of warnings
    .fi
 .end  # restraining with a priori estimation and control #
 .pr  FEDX .pr ;
```

```
.proc  extrap = (.ref .scal  f0, x, fx, y, .proc (.scal ).scal  fun)
.bool :
.pr  XDEF EXTRAP .pr
# it is assumed that fx < f0, f0 = fun(0) and .true  is delivered if
  we stop due to a too small step #
.begin  .bool  b:= .true ;
   .if  y - x > x .or  y = one
   .then  .scal  fy:= fun(y), u;
      .while  b:= fy < fx
      .do  u:= quad(zero, f0, x, fx, y, fy); x:= y; fx:= fy;
         .if  u = zero .then  u:= one .fi ;
         .if  u - y <= y .then  end .fi ;
         y:= u; fy:= fun(u)
      .od
   .fi ;
end: b
.end  # extrapolation until minimum is passed #
.pr  FEDX .pr ;
```

6.6. STOPPING CRITERIA

6.6.1. Criteria for convergence

An important aspect of a Newton-like program is the stopping rule. For a converging sequence $\{x_k\}_{k=0}^{\infty}$ we need some criterion to determine whether the iterate obtained can be considered a good approximation to the solution. We can think of two possible criteria:

$$(6.46) \qquad \|F(x_k)\| \le \delta_f, \quad \text{for some } k,$$

where δ_f is some prescribed tolerance and

$$(6.47) \qquad \|x_k - x^*\| \le \delta_{rx}\|x_k\| + \delta_{ax}, \quad \text{for some } k,$$

where δ_{rx} and δ_{ax} are prescribed tolerance values. If $J(x)$ satisfies condition 1.6 or 1.7 and $x^* + \theta(x_k - x^*) \in D$ for $\theta \in [0,1]$, x^* is a solution of $F(x) = 0$ and x_k is the approximation to x^* in the k-th step, then by lemma 1.15 or 1.16 we have

$$(6.48) \qquad \|F(x_k) - J(x^*)(x_k - x^*)\| = O(\|x_k - x^*\|^2).$$

So, if $\|x_k - x^*\|$ is small and $\|J(x^*)(x_k - x^*)\|$ large (hence $\|J(x^*)\|$ large) then $\|F(x_k)\|$ will be large and in such cases (6.46) will only be satisfied for x_k relatively close to x^*. However, if $\|J(x^*)(x_k - x^*)\|$ is small relative to $\|x_k - x^*\|$, then (6.46) will be satisfied if $\|x_k - x^*\|$ is still relatively large. In practice it depends on the problem to be solved which criterion is most desirable. Sometimes, only an argument vector x^* is required for which $\|F(x^*)\|$ is small, sometimes the error in the argument vector itself has to be small. Therefore, we like to use both criteria and the user may adapt them to his problem by choosing appropriate values for δ_f, δ_{rx} and δ_{ax}.

Clearly, (6.46) can easily be used, however, (6.47) cannot be evaluated as x^* is not known in advance. To obtain a reasonable upper bound we use the nonlinear majorizing sequence defined in the proof of theorem 5.18 (see also MIEL [1977]). For short we delete the subscript k in x_k and denote x_{k+1} by $\bar{x}$. Then we can apply theorem 5.18 assuming that the iterative process is started with x and the conditions of the theorem are satisfied.

Then

$$\|x^*-\bar{x}\| \le \bar{\zeta} - \bar{\beta}_0.$$

Hence, since $\bar{\beta}_0 = \|\bar{x}-x\|$ for strict Newton-like methods we have

$$\|x^*-\bar{x}\| \le (\bar{\zeta}/\bar{\beta}_0-1)\|\bar{x}-x\|.$$

From (5.55) we see that

$$\bar{\zeta}/\bar{\beta}_0 = \frac{1}{\bar{\alpha}}\left(\chi_2 - \sqrt{\chi_2^2-2\bar{\alpha}}\right) = \frac{2}{\chi_2 + \sqrt{\chi_2^2-2\bar{\alpha}}}$$

and therefore we have

$$1 < 1/\chi_2 \le \bar{\zeta}/\bar{\beta}_0 \le 2/\chi_2 < \infty, \qquad \lim_{\beta_0\to 0} \bar{\zeta}/\bar{\beta}_0 = 1.$$

Thus $\bar{x}$ satisfies the convergence criterion if

(6.49) $$\|\bar{x}-x\| \le (\delta_{rx}\|\bar{x}\| + \delta_{ax})/(\bar{\zeta}/\bar{\beta}_0-1).$$

6.3. REMARK. If $\|F(x)\|$ is small and the condition number of the jacobian approximation is not large relative to 1, then $\bar{\beta}_0$ is small. As $\lim_{\beta_0\to 0} \bar{\zeta}/\bar{\beta}_0 = 1$ we see that the closer x is to x^*, the smaller is $\bar{\beta}_0$ and the larger is $(\bar{\zeta}/\bar{\beta}_0-1)^{-1}$. In fact if x is close to x^* then $(\bar{\zeta}/\bar{\beta}_0-1)^{-1}$ will be greater than 1 and criterion (6.49) is easier satisfied than e.g. $\|\bar{x}-x\| \le \delta_{rx}\|\bar{x}\| + \delta_{ax}$.

Based on (6.49) and using the approximations to various quantities as given in section 6.4 we obtain the following convergence criterion on the variables

(6.50)
$$\hat{e}_k < -1+\sqrt{2}; \qquad \hat{\omega}_k\hat{\beta}_k\hat{\chi}_{1,k} < \tfrac{1}{2}\hat{\chi}_{2,k}^2,$$
$$\|x_{k+1} - x_k\| \le (\delta_{rx}\|x_{k+1}\| - \delta_{ax})/\hat{\psi}_k,$$

where

$$\hat{\chi}_{1,k} = \frac{1+\hat{e}_k}{1-\hat{e}_k}, \qquad \hat{\chi}_{2,k} = \frac{1-2\hat{e}_k-\hat{e}_k^2}{1-\hat{e}_k},$$

$$\hat{\psi}_k = 2\left(\hat{\chi}_{2,k} + \sqrt{\hat{\chi}_{2,k}^2-2\hat{\omega}_k\hat{\beta}_k\hat{\chi}_{1,k}}\right)^{-1} - 1.$$

Note that $\hat{\beta}_k = \|H_k F_k\|$ which is the exact value to be used. Furthermore we need an approximation to $\omega(x_k)$ on a neighbourhood of x_k. Clearly, $\hat{\omega}_k$ is an approximate lower bound to this value as it is only based on approximating the Lipschitz constant in certain directions. Thus, it is not quaranteed that (6.47) is satisfied if (6.50) is satisfied. The usefulness of (6.50) has to be established experimentally.

To summarize, in the nongeneralized Newton algorithms we use (6.46) and (6.50). For the restrained algorithms we add the condition $\lambda_k = 1$. We make an exception for the inverse-updating Newton algorithms. For these algorithms and for the generalized algorithms we use condition (6.46) together with

(6.51) $$\|x_{k+1} - x_k\| \le \delta_{rx}\|x_{k+1}\| + \delta_{ax}.$$

6.6.2. Failure criteria

We distinguish two kinds of failures.

1. *Failure resulting in termination of the algorithm without completing the iteration step.*
   These are:
   a. divergence out of the domain of the function in strict algorithms,
   b. numerical singularity of the jacobian approximation if triangular decomposition is used (see subsection 6.2.2),
   c. trespassing the boundary of the domain D during computation of a difference approximation,
   d. finding a stationary point of the levelfunction which is no solution.
2. *Failure resulting in termination after completion of the iteration step.*
   During computation situations might occur which make termination after completing the iteration step sensible or necessary. Examples are
   a. no better value for the levelfunction can be found due to numerical errors in the function values (condition (6.42)),
   b. the approximate error bound for the jacobian approximation is so large that nonsingularity of the jacobian can not be quaranteed.

We shall now define precisely what is meant by 1.d and 2.b.

Finding a stationary point (ad 1.d)
Let for some $k, B_k = U_1 \Sigma_{\hat{F}_k^+} V_1^T$ be the corresponding singular value decomposition.

Then we say that x_k is a stationary point of $\|F(x)\|^2$ if (cf. theorem 4.31)

$$(6.52) \qquad [U_1U_1^TF_k,F_k] = \|U_1^TF_k\|^2 < \varepsilon_f^2(x).$$

This condition is given in .proc cdatasv (subsection 6.4.9).

singular jacobian expected (ad 2.b)
We want to develop a criterion for non-generalized algorithms, which indicates at an early stage that a (nearly) singular jacobian is expected. If such a criterion is available, then we may use a non-generalized algorithm until the criterion is satisfied and then switch to a generalized method. For non-generalized algorithms, this would avoid slow convergence to points with a singular jacobian which are no solution. Of course the criterion should not disturb the behaviour of the algorithms if the jacobian (approximation) is well-conditioned.

Consider non-generalized algorithms using approximations to required data as in section 6.4 and assume that the functions considered satisfy the conditions of theorem 5.7 for some $(x,H) \in D \times L(\mathbb{R}^n)$, with $\theta = 2$ (we choose this value and not a value close to 1 in order to avoid numerical problems). Then conditions (5.11) and (5.30) become

$$2e(x)\kappa(x) \leq 1$$

and a real number τ has to exist such that

$$\tau \leq \min(1,(4\alpha(x)(1+2\kappa(x)))^{-1}).$$

By theorem 5.7 we have, provided that the step length factor is chosen appropriately,

$$\|AF(\Psi_1(x,H))\| \leq (1 - \frac{\tau^2}{8})\|AF(x)\|.$$

In order to guarantee that the decrease in $\|AF(x)\|$ is not of the order of the numerical errors in $fl_\varepsilon(F(x))$ we demand (using (6.3))

$$\tau^2\|AF(x)\| \geq 16\|A\|\varepsilon_f(x).$$

So, using the upper bound on τ, we obtain as a failure criterion

(6.53) $\|AF_k\| \le 16\|\hat{A}\|\varepsilon_f(x_k)\ \max(1, 16\hat{\alpha}_k^2(1+2\hat{\kappa}_k)^2)$,

where $\|\hat{A}\|$ equals 1 if $A = I$, $\hat{\eta}_0$ if $A = H_0$ and $\hat{\eta}_k$ if $A = H_k$. In the derivation of this inequality we used several upper bounds which may be somewhat rough. Therefore, we relax this failure condition with a factor. For easiness of programming we dropped the factor 16 and changed $16\hat{\alpha}_k^2$ into $4\hat{\alpha}_k^2$. In addition to this failure criterion we use

(6.54) $2\hat{e}_k\hat{\kappa}_k \ge 1$.

The condition $2e_k\kappa_k \le 1$ was a premise for the above reasoning. Moreover, it guarantees that the jacobian matrix is nonsingular and that a descent direction can be found (see (6.41)).

The usefulness of the relaxed criterion (6.53) and (6.54) as criteria for switching from non-generalized algorithms to generalized algorithms has to be established experimentally.

6.6.3. Description in ALGOL 68

We give one routine for generalized algorithms and inverse updating algorithms (.proc stopspl) and one for the other Newton-like algorithms (.proc stopful). Note that the failure criteria (6.53) and (6.54) are used only if save = .true.

```
.proc  stopspl = .bool :
.pr  XDEF STOPSPL .pr
.if  nrmf = zero .then  reset number of warnings; .true
.elif  nrmdx <= dlrx * nrmx + dlax .and  nrmf <= dlf
   .then  reset number of warnings; .true
.elif  it >= maxit
   .then  torrix(warning, text4); .true
.else  number of warnings > 0
.fi  # simple convergence and failure criteria #
.pr  FEDX .pr ;
```

```
.proc  stopful = .bool :
.pr  XDEF STOPFUL .pr
.if  nrmf = zero .then  reset number of warnings; .true
.else  .scal  ksi1:= (one + e) / (one - e),
              ksi2:= (one - (e + two) * e)/(one - e),
              alfa2:= omga * beta * two;
   .if  (labda < one .or  e >= sqrt(two) - one ! .false
        ! nrmf <= dlf .and
          (.scal  a1 = alfa2 * ksi1, k2 = ksi2 * ksi2;
           (a1 < k2 ! nrmdx * (two / (ksi2 + sqrt(k2 - a1)) - one)
            <= (dlrx * nrmx + dlax) ! .false )))
   .then  reset number of warnings; .true
   .elif  it = 1 .and  (e <= one - small scal .or  .not  safe)
     .then  .false
   .elif  safe .and  e * kappa * two > one .and  (dif .or  anl)
     .then  torrix(warning, text8); .true
   .elif  safe .and
      (.scal  tau2i = (((one + kappa * two) * alfa2)**2) .max  one;
       slevel <= epf * tau2i *
                  (.case  c .of  a .in  one, eta0, eta .esac ))
     .then  torrix(warning, text9); .true
   .elif  it >= maxit .then  torrix(warning, text4); .true
   .else  number of warnings > 0
   .fi
.fi  # full convergence and failure criteria #
.pr  FEDX .pr ;
```

6.7. SYNTHESIS OF BASIC NEWTON-LIKE ALGORITHMS

In this section we synthesize the basic Newton-like algorithms from the basic modules given in section 6.2 up to 6.6. The structure of these algorithms is already described in section 6.1. We are still free to choose the restraining strategy and the method of approximating the inverse jacobian from the various possibilities given in sections 6.3 and 6.5. The other modules are determined by these choices, except for the initial choice of the jacobian approximation. However we shall choose the initial approximation in the same way as is done in the iteration step, except for the inverse-updating methods. In the last methods we choose H_0 equal to the inverse of the difference approximation to the jacobian. As far as restraining is concerned we are free to choose the matrix A defining the level function (6.40). However, such a choice induces implicit scaling of the function, which is to be considered together with explicit scaling as an optional feature that can be built in one or more basic algorithms. In the basic algorithms we choose $A = I$.

We summarize the various choices for restraining and approximation of the inverse jacobian which have been described in former sections.

Restraining

(S) No restraining (strict) (subsection 6.5.2).

(B) Bisection (subsection 6.5.3).

(I) Interpolation (subsection 6.5.4).

(E) A-priori estimation and control (subsection 6.5.5).

Approximation of inverse jacobian

(A) Inverse of analytic jacobian (subsection 6.3.2).

(D) Inverse of difference approximation (subsection 6.3.3).

(U1,U2) Inverse-updating by the two methods described in subsection 6.3.4, respectively.

(GA) Generalized inverse of analytic jacobian (subsection 6.3.5).

(GD) Generalized inverse of difference approximation to the jacobian (subsection 6.3.6).

As is already discussed in subsection 6.5.1 we do not consider restrained inverse-updating Newton methods (B, I or E).

A-priori estimation and control of the step length factor (E) is not used in generalized algorithms. The theory on which this strategy is based demands a nonsingular jacobian matrix.

In order to show the effect of the special failure criteria (6.53) and (6.54) we have added to our basic set a strict Newton and difference Newton method which do not use these criteria (we added a W to the name to distinguish them from the others).

The names of the basic Newton-like algorithms are composed from the capitals between the parentheses before the particular choices above, starting with the capital(s) denoting approximation of the inverse jacobian.
We obtain the following 18 basic Newton-like algorithms.

1. ASW, AS, AB, AI, AE,
2. DSW, DS, DB, DI, DE,
3. U1S, U2S,
4. GAS, GAB, GAI,
5. GDS, GDB, GDI.

Description in ALGOL 68

We shall not define all 18 basic programs. We only give 5 examples: AB, DB, U2S, GAS and GDS. The definition of the other programs can easily be derived from these example programs.

We shall give first a procedure which causes resetting of some input parameters if these are apparently wrong.

```
.proc  default = .void :
.pr  XDEF DEFAULT .pr
.begin  .if  eprf < small scal
   .then  torrix(warning, text13); eprf:= small scal .fi ;
   .if  epaf < zero
   .then  torrix(warning, text14); epaf:= zero .fi ;
   .if  eprj < small scal
   .then  torrix(warning, text15); eprj:= small scal .fi ;
   .if  epaj < zero
   .then  torrix(warning, text16); epaj:= zero .fi ;
   .if  dlf < epaf
   .then  torrix(warning, text17); dlf:= epaf .fi ;
   .if  dlrx < small scal
   .then  torrix(warning, text18); dlrx:= small scal .fi ;
   .if  dlax < small scal
   .then  torrix (warning, text19); dlax:= small scal .fi
.end  # resetting to default of wrongly given precisions #
.pr  FEDX .pr ;
```

```
.proc  ab = .bool :
# restrained newton algorithm (with bisection)#
.begin  .bool  bl; anl:= .true ;
   it:= dcnt:= 0; fcnt:= 1;
   e:= zero; omga:= beta:= kappa:= eta:= labda:= one;
   v:= genranvec(n); default;
   .if  number of warnings /= 0 .then  copyerrorfile .fi ;
   x:= x0; nrmx:= .nrm  x; .func  fu = fun(x);
   .if  .if  .not  in .of  fu
      .then  torrix(warning, text12); .false
      .else  f:= f .of  fu; nrmf:= .nrm  f;
          epf:= (eprf + small scal) * nrmf + epaf; nrmf > minscal
      .fi
   .then  b:= jacobian(x); jcnt:= 1;

      .while  it +:= 1; cdatalr;
         .if  number of warnings > 0 .then  .false
         .else  resbis; nrmx:= .nrm  x;
            epf:= (eprf + small scal) * nrmf + epaf;
            .not  stopful
         .fi
      .do  b:= jacobian(x); jcnt +:= 1 .od
   .fi ; .if  .not  (bl:= number of warnings = 0)
   .then  copyerrorfile .fi ; bl
.end  # ab #;



.proc  db = .bool :
# restrained difference newton algorithm (with bisection)#
.begin  .bool  bl; dif:= .true ;
   it:= dcnt:= 0; fcnt:= 1;
   e:= zero; omga:= beta:= kappa:= eta:= labda:= one;
   v:= genranvec(n); default;
   .if  number of warnings /= 0 .then  copyerrorfile .fi ;
   x:= x0; nrmx:= .nrm  x; .func  fu = fun(x);
   .if  .if  .not  in .of  fu
      .then  torrix(warning, text12); .false
      .else  f:= f .of  fu; nrmf:= .nrm  f;
          epf:= (eprf + small scal) * nrmf + epaf; nrmf > minscal
      .fi
   .then  calh; b:= diffjac(x);

      .while  it +:= 1; cdatalr;
         .if  number of warnings > 0 .then  .false
         .else  resbis; nrmx:= .nrm  x;
            epf:= (eprf + small scal) * nrmf + epaf;
            .not  stopful
         .fi
      .do  calh; b:= diffjac(x) .od
   .fi ; .if  .not  (bl:= number of warnings = 0)
   .then  copyerrorfile .fi ; bl
.end  # db #;
```

```
.proc  gas = .bool :
# strict generalized newton algorithm #
.begin  .bool  bl; anl:= .true ; safe:= .false ;
   it:= dcnt:= 0; fcnt:= 1;
   e:= zero; omga:= beta:= kappa:= eta:= labda:= one;
   v:= genranvec(n); default;
   .if  number of warnings /= 0 .then  copyerrorfile .fi ;
   x:= x0; nrmx:= .nrm  x; .func  fu = fun(x);
   .if  .if  .not  in .of  fu
      .then  torrix(warning, textl2); .false
      .else  f:= f .of  fu; nrmf:= .nrm  f;
          epf:= (eprf + small scal) * nrmf + epaf; nrmf > minscal
      .fi
   .then  b:= jacobian(x); jcnt:= 1;

      .while  it +:= 1; cdatasv;
         .if  number of warnings > 0 .then  .false
         .elif  strict; number of warnings > 0 .then  .false
         .else  nrmx:= .nrm  x;
            epf:= (eprf + small scal) * nrmf + epaf;
            .not  stopspl
         .fi
      .do  b:= jacobian(x); jcnt +:= 1 .od
   .fi ; .if  .not  (bl:= number of warnings = 0)
   .then  copyerrorfile .fi ; bl
.end  # gas #;


.proc  gds = .bool :
# strict generalized difference newton algorithm #
.begin  .bool  bl; dif:= .true ; safe:= .false ;
   it:= dcnt:= 0; fcnt:= 1;
   e:= zero; omga:= beta:= kappa:= eta:= labda:= one;
   v:= genranvec(n); default;
   .if  number of warnings /= 0 .then  copyerrorfile .fi ;
   x:= x0; nrmx:= .nrm  x; .func  fu = fun(x);
   .if  .if  .not  in .of  fu
      .then  torrix(warning, textl2); .false
      .else  f:= f .of  fu; nrmf:= .nrm  f;
          epf:= (eprf + small scal) * nrmf + epaf; nrmf > minscal
      .fi
   .then  calgh; b:= diffjac(x);

      .while  it +:= 1; cdatasv;
         .if  number of warnings > 0 .then  .false
         .elif  strict; number of warnings > 0 .then  .false
         .else  nrmx:= .nrm  x;
            epf:= (eprf + small scal) * nrmf + epaf;
            .not  stopspl
         .fi
      .do  calgh; b:= diffjac(x) .od
   .fi ; .if  .not  (bl:= number of warnings = 0)
   .then  co`yerrorfile .fi ; bl
.end  # gds #;
```

```
.proc  u2s = .bool :
# inverse-updating newton #
.begin  .bool  bl; it:= dcnt:= 0; fcnt:= 1;
   omga:= beta:= kappa:= eta:= labda:= one;
   default;
   .if  number of warnings /= 0 .then  copyerrorfile .fi ;
   x:= x0; nrmx:= .nrm  x; .func  fu = fun(x);
   .if  .if  .not  in .of  fu
      .then  torrix(warning, textl2); .false
      .else  f:= f .of  fu; nrmf:= .nrm  f;
          epf:= (eprf + small scal) * nrmf + epaf; nrmf > minscal
      .fi
   .then  calh; b:= diffjac(x);
      .lud  lub:= .ludec  b; .vec  e = zero .into  genvec(n);
      .for  j .to  n
      .do  e[j]:= 1; b[,j]:= lub .sol  e; e[j]:= 0 .od ;

      .while  it +:= 1; dx:= - (b * f); nrmdx:= .nrm  dx;
         f0:= f; strict;
         .if  number of warnings >0 .then  .false
         .else  nrmx:= .nrm  x;
            .not  stopspl
         .fi
      .do  invupd2(b) .od
   .fi ; .if  .not  (bl:= number of warnings = 0)
   .then  copyerrorfile .fi ; bl
.end  # u2s #;
```

6.8. CONDITIONAL USE OF APPROXIMATION BY UPDATING

This feature is described in corollary 5.15 and remark 5.16. It is applicable to all basic Newton-like methods which compute approximations to e_k, ω_k and κ_k in every iteration step. These are A(S, B, I or E) and D(S, B, I or E). As approximation by updating requires no extra function evaluations, it can be used more economically than difference approximation or possibly even evaluation of the analytic expressions for the jacobian. Note that conditional use of updating might increase the number of iteration steps required to solve a problem within a certain precision. Experiments have to establish the usefulness of this strategy.

We use update approximation and no inverse-update approximation, as we have no explicit inverse of the jacobian approximation in the algorithms under consideration and calculation of an explicit inverse is, for large order, about as expensive as two iteration steps.

In the definition as given in corollary 5.15 we have to replace $\mathcal{V}(u)(H,x,\Psi_1(x,H))$ by $(\mathcal{U}(u)(B,x,\Psi_1(x,B^{-1})))^{-1}$ $(H = B^{-1})$ and the constants by their approximations as given in section 6.4. Furthermore, we choose $\theta = 1$ and $\kappa(\bar{\Psi}_1(x,B^{-1}))$ is approximated by the approximation of $\kappa(x)$. We add one extra condition for practical reasons. This is

$$\bar{e}(\bar{\Psi}_1(x,H)) < 0.1. \tag{6.55}$$

This is done to avoid slow convergence if, due to updating, the error becomes almost 1, which is possible if the condition number κ_k is about 1. We also refer to the condition $e_k < -1+\sqrt{2}$ in theorem 5.18. The value 0.1 is chosen after some experimental tests.

If during the restraining no appropriate value of $\lambda(x,H)$ can be found, then this might be the result of bad approximation of the required quantities. Therefore, the algorithm is not terminated if in this iteration step an update approximation was used, but it is forced to perform a normal iteration first (with analytic jacobian or difference approximation).

We do not use updating in the first two iteration steps. The approximation $\hat{\omega}_k$ is expected to be reasonable only after computation of data in the second iteration step. Note that, if conditional updating is used,

$\hat{e}_{k+1}$ is calculated in .proc conupdjac instead of in .proc cdatalr.

Description in ALGOL 68

```
.proc  conupdjac = (.proc (.vec ).mat  jacapp).mat :
.pr  XDEF CONUPD .pr
.begin  .scal  pu; .vec  u;
   .if  (.if  it = 1 .or  e >= onetenth .or  .not  update
         .then  .false
         .else  u := decb .solve  (f - f0);
            pu := dx * u; .scal  nrmu = .nrm  u;
            e:= (e / (one - e) + (one + nrmdx * .widen  3 /
                (nrmu * two)) * nrmdx * omga) * (one + e);
            kappa * e < one .and  e < onetenth
        .fi )
   .then  upd:= .true ;
      .vec  q = f + f0 * (labda - one); .for  j .to  n
      .do  b[,j] +< ((u[j] / pu) * q) .od ;
      b
   .else  upd:= .false ; jacapp(x) .fi
.end  # conditional updating with jacapp #
.pr  FEDX .pr ;
```

Modification of e.g. algorithm AB is obtained by adding after the statement : " jcnt:= 1;" the statement: "upd:= .false;". Furthermore, replace "b:= jacobian(x); jcnt +:= 1" by:
" b:= conupdjac(jacobian);
  .if anl:= .not upd .then jcnt +:= 1 .fi ".

6.9. CONDITIONAL USE OF FIXED APPROXIMATION

Conditional use of fixed jacobian approximation is based on corollary 5.17. It is applicable to all Newton-like algorithms described in section 6.7 except for inverse-updating algorithms. We need different conditions for algorithms using triangular decomposition and those using singular value decomposition. These conditions follow from the theorems 3.19 and 3.20 respectively. To obtain a fixed approximation as well as its decomposition does not require additional computation. Hence it may be very attractive to incorporate this feature. The negative effects of greater errors in the jacobian approximation, probably yielding slower convergence, can hardly be estimated theoretically. Experiments have to establish the practical usefulness of this feature.

Let $\Psi = (\Psi_1, \Psi_2)$ be an appropriate Newton-like process (associated with a basic Newton-like algorithm except for U1S or U2S). Then, the k-th iteration step $(k>2)$ of the modified process $\bar{\Psi} = (\bar{\Psi}_1, \bar{\Psi}_2)$ say, generating $\{(x_k, H_k)\}$ for given (x_0, H_0), is defined by

$$\bar{\Psi}_1(x_k,H_k) = \Psi_1(x_k,H_k) \quad (= \mathbf{x}_{k+1}, \text{ by definition}),$$

$$\bar{\Psi}_1(x_k,H_k) = \begin{cases} H_k, & \text{if } \hat{\hat{e}}_{k+1} < \min\,(1/\hat{\kappa}_k, 0.1) \text{ and } \hat{r}_k^+ = n, \\ \Psi_2\,(x_k,H_k), & \text{otherwise,} \end{cases}$$

where $\hat{\hat{e}}_{k+1}$ is the a-priori estimation to e_{k+1} if fixed approximation would have been used:

$$\hat{\hat{e}}_{k+1} = \hat{e}_j + \hat{\omega}_k(1+\hat{e}_j)\|x_{k+1}-x_j\|, \text{ if triangular decomposition is used,}$$

$$= \hat{e}_j^+ + \hat{\gamma}_k\hat{\eta}_k^+ \,\|x_{k+1} + x_j\|, \text{ if singular value decomposition is used,}$$

and where j is the greatest index less than or equal to k such that $\bar{\Psi}(x_{j-1},H_{j-1}) = \Psi(x_{j-1},H_{j-1})$. Note that we also use condition (6.55) in this case. Furthermore, note that if fixed approximation is used then $\hat{r}_k^+ = n$, hence if singular value decomposition is used it follows from subsection 6.4.8 that $\hat{r}_{k+1}^+ = n$.

Moreover, for $j \le i \le k$, we have $\hat{\omega}_i = \hat{\omega}_j$, $\hat{\gamma}_i = \hat{\gamma}_j$, $\hat{\eta}_i = \hat{\eta}_j$ and $\hat{\eta}_i^+ = \hat{\eta}_j^+$.

Description in ALGOL 68

```
.proc  confixjac = (.proc (.vec ).mat  jacapp).mat :
.pr  XDEF CONFIX .pr
.if  (.if  it = 1 .then  .false
      .else  e:= ej + .nrm (x - xj) * omga * (one + ej);
         kappa * e < one .and  e < onetenth
      .fi )
.then  fix:= .true ; b
.else  fix:= .false ; jacapp(x) .fi
# conditional fixed approximation in case of lu decomp. #
.pr  FEDX .pr ;

.proc  confixjacg = (.proc (.vec ).mat  jacapp).mat :
.pr  XDEF CONFIXG .pr
.if  (.if  it = 1 .then  .false
      .else  e:= ej + .nrm (x - xj) * omga * eta;
      kappa * e < one .and  e < onetenth .and
     .case  decb .in  (.ref .svd  bsv): .upb  sngval .of  bsv = n
     .esac  .fi )
.then  fix:= .true ; b
.else  fix:= .false ; jacapp(x) .fi
# conditional fixed approximation in case of sv decomp. #
.pr  FEDX .pr ;
```

Modification of the basic algorithms is as for conditional updating with "upd" replaced by "fix".

6.10. IMPLICIT AND EXPLICIT SCALING

Consider the restrained basic Newton-like algorithms of section 6.7 which use triangular decomposition. A choice of the matrix A in the definition of the level function (6.40) is equivalent with scaling of the function with A (we say implicit scaling) (see remark 4.18). We have restricted attention to three ways of implicit scaling (see subsection 6.4.5): (1) $A = I$, (2) $A = H_0$, (3) $A = H_k$. The last two choices yield affine invariant methods. (Note that the convergence theory of chapter 5 does not hold for variable A). Although the last two choices yield affine invariant methods, use of finite arithmetic can spoil this property. For instance, if we compute $TJ_k \backslash TF_k$ then we may obtain another result than if we compute $J_k \backslash F_k$, as the errors depend on the condition number of TJ_k and J_k, respectively. Implicit scaling with choice (2) or (3), only influences the restraining strategy and not the solution of the linear system.

Explicit scaling of the function and variables is described in subsection 1.3.6. It is in fact based on scaling of the jacobian approximation at the starting guess with diagonal matrices D_1 and D_2 satisfying (1.33) and (1.34) (with A replaced by B_0). A description, in ALGOL 68, of the matrix scaling is given in subsection 6.2.3.

We shall consider implicit scaling for the restrained algorithms A(B, I, E) and D(B, I, E) and explicit scaling for all algorithms, as far as these algorithms appear to be useful from the general testing.

Description in ALGOL 68

Implicit scaling is obtained by assigning an appropriate value to c .of a (see prelude nlsprl) in the initial phase of the iterative process.

Explicit scaling is illustrated by the following example program, which defines algorithm AB with explicit scaling.

```
.proc  scab = .bool :
# restrained newton with explicit scaling #
.begin  .bool  bl; anl:= .true ;
   it:= dcnt:= 0; fcnt:= 1;
   e:= zero; omga:= beta:= kappa:= eta:= labda:= one;
   v:= genranvec(n); default;
   .if  number of warnings /= 0 .then  copyerrorfile .fi ;
   x:= x0; nrmx:= .nrm  x; .func  fu:= fun(x);
   .proc (.vec ).func  oldfun; .proc (.vec ).mat  oldjacob;

   .ref .bool  scr = scr .of  scb, scc = scc .of  scb;
   .ref .vec  rows = rows .of  scb, cols = cols .of  scb;
   .if  .if  .not  in .of  fu
      .then  torrix(warning, text12); .false
      .else  f:= f .of  fu; nrmf:= .nrm  f;
         epf:= (eprf + small scal) * nrmf + epaf; nrmf > minscal
      .fi
   .then  jcnt:= 1; b:= jacobian(x); scb:= .scale  b;
      .if  sng .of  scb .then  torrix(warning, text5)
      .elif  scr .or  scc
      .then  oldfun:= fun; oldjacob:= jacobian;
         fun:= (.vec  x).func :
               (.func  fu:= oldfun((scc ! x .dmul  cols ! x));
                .if  scr .then  f .of  fu:= f .of  fu .dmul  rows
                .fi ; fu);
         jacobian:= (.vec  x).mat :
               (mat .of  scb:= oldjacob((scc ! x .dmul  cols ! x));
                scb:= .scale  scb; mat .of  scb);
         .if  scc .then  x:= cols .dimul  x .fi ;
         .if  scr .then  f:= rows .dmul  f .fi ;
         nrmf:= .nrm  f; epf:= (eprf + small scal) * nrmf + epaf
      .fi ;

      .while  it +:= 1; cdatalr;
         .if  number of warnings > 0 .then  .false
         .else  resbis; nrmx:= .nrm  x;
            epf:= (eprf + small scal) * nrmf + epaf;
            .not  stopful
         .fi
      .do  b:= jacobian(x); jcnt +:= 1 .od
   .fi ;
   .if  scc .then  x:= cols .dmul  x .fi ;
   .if  scr .then  f:= rows .dimul  f .fi ;
   .if  scc .or  scr
   .then  b:= .bckscale  scb; fun:= oldfun; jacobian:= oldjacob
   .fi ;
   .if  .not  (bl:= number of warnings = 0)
   .then  copyerrorfile .fi ; bl
.end  # scab #;
```

6.11. REDUCTION OF PROBLEMS WITH LINEAR FUNCTION COMPONENTS

Suppose that the problem $F(x) = 0$ can be splitted in a linear problem $Ax-b = 0$ and and a nonlinear problem $\bar{F}(x) = 0$, with $\bar{F} : D \to \mathbb{R}^{n-p}$ and A a $p \times n$ matrix with rank equal to p. Then, according to theorem 1.27 we may reduce the nonlinear problem of order n to a nonlinear problem of order n-p, yielding

$$G(z) = \bar{F}(A^+b+V_2z),$$

where V_2 is given by (1.15) and (1.17). For the jacobian of G we have (cf. remark 1.28)

$$G'(z) = \bar{F}'(A^+b+V_2z)V_2.$$

Description in ALGOL 68

```
 .proc  reducenewt = (.proc .bool  newt).bool :
 # newt gives the newton-like program with which the reduced problem
   has to be solved #
 .pr  XDEF REDNEWT .pr
 .begin  .proc (.vec ).func  oldfun; .proc (.vec ).mat  oldjacob;
    .int  m = .upb  x0, p = 1 .upb  la;
    .mat  a0 = zero .into  gensquare(m);
    a0[1:p,1:m]:= la; .svd  asv0:= .svdec  a0;
    .scal  eps = .max (sngval .of  asv0) * small scal * ten;
    .svd  asv:= eps .trims  asv0;
    .int  r = .upb (sngval .of  asv);
    .if  r /= p
    .then  torrix(warning, "matrix of linear part not full rank");
       copyerrorfile; x:= x0; .false
    .else  .bool  bl;  sol := asv .sol  lb;
       v2:= (v .of  asv0)[1:m, r+1:m];

       oldfun:= fun; oldjacob:= jacobian;
       fun:= (.vec  x).func : oldfun(sol + v2 * x);
       jacobian:= (.vec  x).mat : oldjacob(sol + v2 * x) * v2;
       n:= m - r; x0:= .trnsp  v2 * (x0 - sol);
       bl:= newt; x:= sol + v2 * x;
       fun:= oldfun; jacobian:= oldjacob; bl
   .fi
.end  # reduction of problem with linear components #
.pr  FEDX .pr ;
```

CHAPTER 7

EXPERIMENTAL EVALUATION OF METHODS

7.1. INTRODUCTION

In this chapter we compare the methods described in chapter 6. Furthermore, three algorithms of componentwise approximation, based on the method given by BROWN [1969], are compared with the Newton-like algorithms. These algorithms are described in MORÉ & COSNARD [1979] with names BROWN, BRENT and BRENTM. We use the abbreviations BW, BT and BTM, respectively.

Our ultimate goal is
- to obtain insight in the behaviour of the methods and the effect of particular features,
- to select the "best" methods or combinations of methods for implementation as part of a software library.

We distinguish three classes of criteria for evaluating algorithms for solving systems of nonlinear equations and software based on these algorithms.
1. Criteria about availability of theoretical results for algorithms. Such theoretical results may be theorems about global and/or (semi-) local convergence or asymptotic convergence behaviour.
2. Criteria about the behaviour of algorithms based on experimental results. Such criteria may concern:
   - efficiency (time and storage required to solve problems),
   - robustness (capability of solving relatively hard problems),
   - reliability (is the computed solution accurate within prescribed precision, or is an appropriate error message given).
3. Criteria about the programs and program descriptions, such as
   - structure of the program,
   - ease of choosing values for the program parameters,
   - availability of a users manual.

In the foregoing chapters we paid attention to theoretical results for the Newton-like algorithms. Moreover, program structure is induced by the modular set up of the programs in chapter 6. In this chapter we focus attention to the criteria from the second class. First we describe more precisely what is meant by the notions efficiency, robustness and reliability (sections 7.2 up to 7.4). Then, we describe the experimental design (section 7.5) and, finally, we describe the results of the evaluation of the basic algorithms and the features (section 7.6 and 7.7). In section 7.8 we state conclusions based on these results.

7.2. STANDARD TIME

The notion "time" is essential for defining efficiency of algorithms or programs. It seems to be obvious to measure the CPU-time required to solve a problem with a program on a computer and to say that one program is less efficient than another for solving a certain problem, if this program uses more CPU-time than the other. The crucial point however, is that such a measure yields the efficiency of a program on that moment, but not of the underlying algorithm. The CPU-time required to solve a problem depends highly on (1) programming language, (2) compiler, (3) number of users on the machine in multi-user running systems, (4) memory asked for, etc. As we like to obtain conclusions about the performance of an algorithm and not only about some particular program implementing it, we cannot use simply CPU-time. Another suggestion may be to use the number of basic arithmetical operations (additions plus multiplications) as a measure for the "time" required by an algorithm. However, this also has several disadvantages. (1) Neither the CPU-time required for multiplication and addition nor the ratio of these times is the same for all machines or even for central processors of one machine (e.g. the CYBER 73). As certain computations can be rewritten such that less multiplications are used at the cost of more additions (e.g. matrix multiplication) it is not quite clear what has to be counted. (2) Memory access, particularly array accesses, are relatively expensive and neglected here. (3) It is difficult to obtain any idea about the actual CPU-time required by a program when implemented for a certain programming system.

We like to introduce a time unit which depends on
(1) the computer and its running system,
(2) the programming system used (language, compiler, software library),
(3) the kind of algorithms to be compared (Newton-like algorithms),
(4) the order of the problem to be solved.
This notion should be such, that two programs implementing the same algorithm use about the same amount of standard time units. Moreover, a standard time unit should be easily expressable in CPU-seconds for a given computer and programming system. We define a standard time unit dependent on the order n, as for large order the matrix-vector computations are the bulk of the work, but for small order other things, e.g. routine calls, may also play an important role.

7.1. DEFINITION. Let be given $x \in \mathbb{R}^n$, $f = F(x) \in \mathbb{R}^n$ and $B = J(x) \in L(\mathbb{R}^n)$. A *standard time unit*, U_n, of order n, dependent on a certain computer with given running system, on a certain programming system (language, compiler, software library) and on other machine- and time dependent quantities (e.g. the number of users at a certain moment) is the CPU-time required to compute $y = x - B\backslash f$ (cf. (1.13)) on that computer under the given conditions.

Note that the computation $y = x - B\backslash f$ (by use of triangular decomposition and forward and backward substitution) is the most time consuming part of an iteration step of a strict Newton method, besides from the evaluation of the function and its jacobian. It should be stressed that this computation has to be done with use of the appropriate software library programs which are available in the given programming system and which would have been used in the strict Newton program if it would belong to this library. The conclusions about efficiency, based on standard time are easily transformed to CPU-time, if it is known how much CPU-time is equivalent to one standard time unit for a given computer system. This knowledge can be obtained by running a program in the given programming system which performs computation of $y = x - B\backslash f$ for given x, f and B and measuring the CPU-time required for this computation. We give an example of such a program for the ALGOL 68 programming system introduced in chapter 6.

7.2. EXAMPLE. The following ALGOL 68 program prints U_n(n=2,13,24,35,46) in CPU seconds for the ALGOL 68 programming system from chapter 6, when it is run on the CYBER 73 with the NOS/BE running system.

```
.begin
   .proc comp = (.int n).void :
   .begin .lud dec = .ludec b;
      .vec y = x - (dec .sol f); .skip
   .end # computation to get standard time unit #;

   .index ind = genindex(5);
   ind[1]:= 100; ind[2]:= 8; ind[3]:= 4; ind[4]:= ind[5]:= 1;
   .for n .from 2 .by 11 .to 46
   .do x:= genranvec(n); f:= genranvec(n);
      .proc ran = (.int i, j).scal : next random(setr);
      b:= ran .into gensquare(n);
      .int k = ind[(n-2) .over 11 + 1]; .real t:= clock;
      .for i .to k .do comp(n) .od ; t:= clock - t;
      print(("standard time unit of order ", n,
             "is in cpu secs: ", t / k, newline))
   .od
.end
```

As an illustration, we give the results of this example program in table 7.1, together with the amount of CPU-seconds in a standard time unit for the ALGOL 60 system with the NUMAL software library and for the FORTRAN IV system with the NAG software library. Moreover, we give the amount of standard time units required if $y = x - B^{+}f$ is computed using singular value decomposition for these three programming systems (obtained in ALGOL 68 by replacing the first statement in .proc comp by: ".svd dec = .svdec b;"). The discrepancies between these standard times may be due to (1) algorithmic differences of the matrix decomposition routines in the three programming systems, (2) the fact that the programs have been run in different nights. However, note that there is agreement within at most 50% from the mean and CPU times sometimes differ up to a factor 6. Moreover, note that the differences between the standard times for ALGOL 60 and FORTRAN IV are at most 15% from the mean. These versions use software libraries with optimized code for vector and matrix operations (in NUMAL most of these operations are written in machine language). For the ALGOL 68 system of chapter 6 we did not use optimized code for vector and matrix operations. Table 7.1 shows that standard times may be more appropriate for comparing efficiency of algorithms, but results based on the use of standard times should be interpreted rather carefully. Differences in efficiency of algorithms which are less than 20% say are, in fact, negligible.

<u>table 7.1.</u>

U_n in CPU seconds for three programming systems and standard times for implementations in these systems of a step of the generalized Newton method.

n	U_n in CPU-sec.			standard times of generalized Newton step			mean	largest deviation
	A68	A60	FIV	A68	A60	FIV		
2	0.019	0.013	0.0030	1.5	1.4	1.5	1.5	7%
13	0.25	0.14	0.060	9.6	7.9	5.8	7.8	26%
24	0.80	0.45	0.25	15	9.6	7.6	11	31%
35	1.9	0.99	0.64	17	11	8.3	12	42%
46	3.2	1.8	1.3	21	11	9.2	14	50%

7.3. PROBLEM INDICATORS

The probability that a problem can be solved with a certain algorithm and, if it can be solved, how efficient solving will be, depends on the characteristics of the problem. In this section we shall give a set of *problem indicators*, i.e. quantities which give indications about such characteristics. First, we define the problem of solving a system of nonlinear equations on a given computer.

7.3. DEFINITION. A problem of solving a system of nonlinear equations on a computer with machine precision ε is defined by

(1) a function F of order n with domain $D \subset \mathbb{R}^n$,

(2) an initial guess $x_0 \in D$ to the solution of $F(x) = 0$,

(3) constants ε_{rf} and ε_{af} satisfying (6.3),

(4) tolerance values δ_f, δ_{rx} and δ_{ax} defining the stopping criteria (6.46) and (6.50) or (6.51).

In the sequel we restrict attention to problems which, moreover, satisfy

(i) F and x satisfy condition 1.6 for all $x \in D$,

(ii) $fl_\varepsilon(F(x))$ is a $(\|x\|\delta_{rx} + \delta_{ax})$-unimodal approximation to F on D (see definition 1.12).

A class of problems which are defined by definition 7.3 and satisfy the above conditions is denoted by $\mathcal{C}$. We say that the analytic jacobian of the problem is available if explicit expressions for the elements of the jacobian matrix are given, as well as constants ε_{rj} and ε_{aj} satisfying (6.4).

7.4. PROPOSITION. *The following set of problem indicators provide a good characterization of the problems of a class $\mathcal{C}$ as far as properties are concerned influencing the behaviour of Newton-like algorithms or Brown's algorithm.*

1. *The defining quantities (see definition 7.3.).*
2. *The availability of an analytic jacobian (with ε_{rj} and ε_{aj}).*
3. *The set of indices of those function components which are linear in all variables.*

4.a. *The value of $\kappa(x)$ (see notation 5.1) for $A = I$ and $x \in D$. If $J(x)$ is singular then we define $\kappa(x) = \infty$ and if $F(x) = 0$ then $\kappa(x) = \lim_{k\to\infty} \kappa(y_k)$, for some sequence $\{y_k\} \subset D$, provided that this limit exists;*

otherwise $\kappa(x)$ *is undefined.*

b. *The value of* $\omega(x)$ *on* D, *provided that condition 1.7 is satisfied on* D; *otherwise the value of* $\gamma(x)$ *on* D, *for all* $x \in D$.

c. *The value of* $\|(J(x_0))^{-1}F(x_0)\|$.

5.a. *The standard time* $t_F(x)$ *required to evaluate the value of* $F(x)$ $(x \in D)$.

b. *If the analytic jacobian is available, then the standard time* $t_J(x)$ *required to evaluate the value of* $J(x)$ $(x \in D)$.

6. *The scaling of the function and variables. I.e. the ratio of the largest and smallest row and column norm, respectively, of* $J(x)$ $(x \in D)$.

We want to select test problems based on the values of their problem indicators. This requires simplification of the set of problem indicators.

<u>7.5. REMARK</u>. In the sequel we shall use the following simplified set of problem indicators, where $x^* \in D$ satisfies $F(x^*) = 0$.

1, 2 and 3 as in proposition 7.4.

4.a. $\bar{\kappa}_0 = \bar{\kappa}(x_0)$; $\bar{\kappa}_* = \bar{\kappa}(x^*)$,

where $\bar{\kappa}(x) = \|J(x)\|\|(J(x))^{-1}\|$.

b. $\bar{\omega}_0 = \bar{\omega}(x_0)$ and $\bar{\omega}_* = \bar{\omega}(x^*)$, where

$$\bar{\omega}(x) = \max_{i=1,\dots,p} (\max(w(x,u_i), w(u_i,x))),$$

$$w(x,y) = \begin{cases} \dfrac{\|J(x)\backslash F(y) - J(y)\backslash F(y)\|}{\|x-y\|\|J(y)\backslash F(y)\|} & \text{, if } F(y) \neq 0,\ x \neq y \text{ and } J(x), J(y) \text{ nonsingular,} \\ \infty & \text{, if } J(x) \text{ or } J(y) \text{ are singular,} \\ 0 & \text{, otherwise,} \end{cases}$$

$$u_i = x + hv_i/\|v_i\|, \quad i = 1,\dots,p,$$

with v_i having randomly chosen elements in $[-1,1]$, $h > 0$ is some small real number, depending on x and the machine precision, and p is some integer $1 \leq p \leq n$.

c. $\bar{\beta}_0 = \|(J(x_0))^{-1}F(x_0)\|$.

5.a. t_F, where it is assumed that we have roughly $t_F = t_F(x)$, for all $x \in D$. t_F is called the *function evaluation time*.

b. If the analytic jacobian is available: t_J, where it is assumed that we have roughly $t_J = t_J(x)$ for all $x \in D$. t_J is called the *jacobian evaluation time*.

6.a. $\|D_1\|$, $\|D_2\|$, with D_1 and D_2 given by (1.33) and (1.34) for $A = J(x_0)$.

b. $\bar{\kappa}_0^s = \bar{\kappa}^s(x_0)$, $\bar{\kappa}_*^s = \bar{\kappa}^s(x^*)$, with

$$\bar{\kappa}^s(x) = \|D_1 J(x) D_2\| \, \|(D_1 J(x) D_2)^{-1}\|.$$

The problem indicators play an important role in the experimental evaluation of the algorithms, as will be explained in the next sections.

7.4. ROBUSTNESS, RELIABILITY AND EFFICIENCY

Most of us have some intuitive idea about the notions robustness, reliability and efficiency. We will try to give more precise descriptions or definitions of our interpretation of these notions. To this end we need the following descriptions.

7.6. DESCRIPTION. Let be given a parametrized test problem, such that variation of the value of one or more parameters of this problem effectuates a variation of the value(s) of one or more problem indicators (remark 7.5). Suppose we have obtained the values of the problem indicators for a number of parameter values. Then a table, which gives performance results of an algorithm tested on this problem for these parameter values, is called a *performance table*.

We can imagine many different kinds of performance results, such as standard time required to solve the problem, number of function and/or jacobian evaluations, number of iteration steps, accuracy obtained, etc.

7.7. DESCRIPTION. A set of test problems $T \subset C$ is called a *representative test set* for C, if the values of the problem indicators from remark 7.5, except for t_F and t_J, are reasonably spread over the ranges of values that can be obtained by problems in C. (The exception of t_F and t_J will be explained in remark 7.11).

This description is intuitive and not at all exact. Moreover, it is not based on the problem indicators as described in proposition 7.4, which are claimed to give a good characterization of problems in C, as far as behaviour of our algorithms is concerned, but only on the simplified set of remark 7.5. Nevertheless, we think that this description is useful in practice.

Intuitively, robustness of an algorithm should reflect the highest degree of difficulty of problems in a certain class, that can be solved adequately by that algorithm. Such a definition requires quantification of the degree of difficulty of a problem when solved by a certain algorithm. By proposition 7.4, such a definition depends on the problem indicators. However, we do not know how. In fact, one of the goals of the evaluation is to get insight in this dependency. Therefore, we think that it is not sensible to quantify neither the degree of difficulty of a problem (relative to an algorithm), nor the notion robustness.

7.8. REMARK. We consider *robustness* of an algorithm, relative to $\mathcal{C}$, in the above intuitive sense. Discussion of this property will be based on:

1. a number of performance tables of a program implementing this algorithm,
2. the number of problems from a representative test set for $\mathcal{C}$, which are solved by the algorithm.

The notion reliable can be described somewhat more precise.

7.9. DESCRIPTION. We say that a program for solving problems from $\mathcal{C}$ is *reliable,* if the program, when it is applied to any problem of a representative test set for $\mathcal{C}$, yields either a solution within the required accuracy or an *informative error message,* i.e. a message giving information about the (bad) characteristics of the problem which might have caused the failure.

From the error messages from the ALGOL 68 programming system we consider the following messages not being informative:

- "too many function evaluations or iterations required",
- "difference approximation impossible, point on boundary domain",
- "divergence out of domain of function".

Hence, termination of a program with such a message shows unreliability of this program and is therefore called an *unreliable failure.*

Reliability is a desirable property of programs. A reliable program provides the tools for interpreting the results of the program. The user can rely upon the approximate solution which is obtained in a successful run and in case of failure he obtains an indication for what reason the program fails.

To describe the notion efficiency we need the following definition.

7.10. DEFINITION. Let P be a given program for solving problems from $\mathcal{C}$ and let $p \in \mathcal{C}$. Let t_F and t_J be given (see remark 7.5) and let

t_s: be the standard time required to perform an iteration step (exclusive the time needed for evaluation of the function or its jacobian),

t_I: be the standard time required to perform the initialization (without function or jacobian evaluation),

m_F: be the number of function evaluations required to solve p by P,

m_J: be the number of jacobian evaluations required to solve p by P ($m_J = 0$ if the analytic jacobian is not available),

m_S: be the number of iteration steps required to solve p by P.

Then, the *time* T(P,p), required to solve p by P is defined by

$$T(P,p) = t_I + t_S m_S + t_F m_F + t_J m_J. \tag{7.1}$$

Note that, if J(x) is approximated using, say q, additional function evaluations, then q is added to m_F.

<u>7.11. REMARK</u>. Use of T(P,p) has many advantages above just measuring the total time by experiments. It enables us to present test results for variable t_F and t_J and to disregard t_F and t_J in the description of a representative test set (see description 7.7). This can be seen by the following reasoning. Assume that for P the values of t_I and t_S are known (can be measured once by experiments). Then, the values of m_S, m_F and m_J for solving p by P can be obtained easily by experiment. Therefore, for given t_F and t_J, T(P,p) can be calculated according to (7.1). If $p' \in \mathcal{C}$ differs from p only in the values of t_F and t_J, then T(P,p') can be calculated from the experimental results of p.

<u>7.12. REMARK</u>. We assume that m_S, m_F and m_J, obtained for solving p by P, are representative for the algorithms of which P is an implementation. Hence, we assume that these values are independent of the computer and programming system. In particular, we assume that these values are not (substantially) affected by round-off. This assumption holds as long as the relative errors due to round-off are small relative to 1. If, for instance, the condition number of a matrix in a linear system is of order $1/\varepsilon$, then the numerical solution may contain large relative errors. Solving such a system with double precision might yield a far better solution. Therefore, particularly the machine precision may influence the values of m_S, m_F and m_J if the condition numbers of the successive jacobian approximations are large. In general, one may expect that program P_1 is more robust than program P_2 if they implement the same algorithm and if the machine precision in P_1 is smaller than in P_2.

7.13. DESCRIPTION. Let $\{p_j\}_{j=1}^{\ell}$ be a representative test set for $\mathcal{C}_n = \{p \mid p \in \mathcal{C}$, p has order n$\}$. Let algorithms $A_1,\ldots,A_k$ be implemented for a given computer and programming system by programs $P_1,\ldots,P_k$, respectively. Then we define the *relative efficiency* of $A_j (j=1,\ldots,k)$ with respect to $\{A_i\}_{i=1}^{k}$ by:

$$(7.2) \qquad \mathcal{E}_j(n,t_F,t_J) = \frac{1}{\ell'} \sum_{i \in I} T(P_j,p_i),$$

where T is defined by (7.1), $I = \{i \mid 1 \le i \le \ell$, p_i is solved by all P_j $(j=1,\ldots,k)\}$ and ℓ' is the number of integers in I. We say that A_i is more *efficient* than A_j for solving problems of $\mathcal{C}_n$ with given values for t_F and t_J, if $\mathcal{E}_i(n,t_F,t_J) < \mathcal{E}_j(n,t_F,t_J)$.

7.14. REMARK. We assume independency of $\mathcal{E}_j(n,t_F,t_J)$ $(j=1,\ldots,k)$ of $\{p_j\}_{j=1}^{\ell}$, as we chose this set to be representative for $\mathcal{C}_n$. We also assume that the relative efficiencies of the algorithms, relative to each other, are not dependent on the set of programs $\{P_j\}_{j=1}^{k}$. In general, this assumption does not hold, due to the fact that I changes if programs are added or deleted from the given set. We base this assumption on the representativity of $\{p_j\}_{j=1}^{\ell}$ and the fact that we compare algorithms with the same structure, designed for the same class of problems.

7.15. REMARK. The notions described in this section are all vague and intuitive. They are based on proposition 7.4, which is, most likely, not completely true. Proposition 7.4 is based on the theory of chapter 5 and our practical experience. We expect it to be reasonable. We emphasize that care must be taken in using the notions from this section. For instance, conclusions like algorithm A_1 is 10% more efficient than algorithm A_2, are not relevant, but conclusions like A_1 is twice as efficient as A_2 certainly are relevant (see also the comment on table 7.1).

7.5. DESIGN OF EXPERIMENTS

7.5.1. General design of comparison of algorithms

Let k Newton-like algorithms and/or algorithms of component wise approximation be given, denoted by A_j (j=1,...,k), which have to be compared. Then, we write programs P_j (j=1,...,k) in ALGOL 60, implementing A_j (j=1,...,k), respectively, on a CYBER 73 computer with NOS-BE running system and machine precision $\varepsilon = 2^{-47}$. We use the current ALGOL 3.435 compiler and the NUMAL software library (see HEMKER et al. [1979]). Then the programs are run for the following specific set of test problems.

Test problems for performance tables (cf. appendix I).

1. problem 1, n = 2,3,...,9,10,13,24,35.
   When n increases the values of $\bar{\kappa}_0$ and $\bar{\beta}_0$ increase rapidly; the value of $\bar{\omega}_0$ increases only very slowly. The values $\bar{\kappa}_*$, $\bar{\omega}_*$ at the solutions also increase only slowly.
2. problem 2, n = 3, c = 10^p (p=1,2,...,11).
   When the value of c increases the values of $\bar{\kappa}_0$ and $\bar{\kappa}_*$ increase while $\bar{\omega}_0$ and $\bar{\omega}_*$ remain small and almost constant.
3. problem 4, n = 2,13, c = 10^p (p=1,2,...,7).
   When c increases the values of $\bar{\kappa}_0$, $\bar{\omega}_0$ and $\bar{\beta}_0$ are constant or decreasing and the values of $\bar{\kappa}_*$ and $\bar{\omega}_*$ increase.
4. problem 5, n = 2,13,24,35,46.
   The values of the problem indicators are small and almost independent of n. These problems are expected to be easily solvable and dependence of the behaviour of algorithms on the order can be tested.
5. problem 5a, n = 35,
   (p,q) = $(10^{-12},10^{-12})$, $(10^{-12},10^{-10})$, $(10^{-12},10^{-8})$, $(10^{-12},10^{-6})$, $(10^{-12},10^{-4})$, $(10^{-12},10^{-2})$, $(10^{-10},10^{-12})$, $(10^{-8},10^{-12})$, $(10^{-6},10^{-12})$, $(10^{-4},10^{-12})$ and $(10^{-2},10^{-12})$.
   For this problems we choose $\delta_f = \delta_{rx} = \delta_{ax} = 10^{-3}$. The test problems show the dependence of the algorithm on errors in function and jacobian.
6. problem 7, n = 2, c = 0.1,1,5,10,50,10^2,10^3,10^4,10^5,10^8.
   When c increases, $\bar{\omega}_*$ increases slowly and the other values remain

constant. The solutions (four for $c \geq 10$) come very close to the origin for large values of c.

Based on the results for these test problems which are given in performance tables (see description 7.6), we state conclusions about specific behaviour of algorithms and we also give preliminary conclusions about robustness and reliability. Based on these conclusions we select programs P_{j_i}, $j_i \in \{1,\dots,k\}$ which appear to be worthwhile for testing on a representative test set. In fact we choose test sets $\mathcal{T}_n$, which are representative for $\mathcal{C}_n = \{p \mid p \in \mathcal{C}, \text{ p has order n}\}$. These sets consist of the following problems.

Test problems in representative test set $\mathcal{T}_n$ (cf. appendix I).

problem 1;
problem 2, c = 10;
problem 3;
problem 4, c = 10, 10^4, 10^7;
problem 5;
problem 6;
problem 7, c = 10, 10^4;
problem 8;
problem 9;
problem 10, for (s_r,s_c) = (1,1), (10^{-3},1), (10^{-6},1), (10^{-9},1), (10^{-14},1),
(1,10^{-3}), (1,10^{-6}), (1,10^{-9}), (1,10^{-14});
problem 11, (s_r,s_c) = (1,1);
problem 12;
problem 13;
problem 14;

So, $\mathcal{T}_n$ consists of 25 problems for each n. By collecting the values of the problem indicators, one can see that these values are reasonably spread, although, for large order these are larger on the average than for small order. We are a bit sloppy with this criterion in order to be able to use the same set of problems in $\mathcal{T}_n$, for each $n \geq 2$.

Due to storage requirements and limitations on the CPU-time that can be used for testing, we only choose the following five different values of n (we shall speak about *selected orders*):

$$n = 2, 13, 24, 35, 46.$$

We choose odd and even values in order to avoid conclusions which are influenced by the fact that orders are only even or odd.

Based on the results given in the performance tables and those for the sets $\mathcal{T}_n$, for the selected orders, we shall derive conclusions about the robustness and reliability of the algorithms. The standard times $\mathcal{T}(P_j,p_k)$ (see (7.1)) or efficiencies $E_{j_i}(n,t_F,t_J)$, where $j_i \in \{1,\dots,k\}$ are the indices of the selected programs, are given for the following *selected values* of t_F and t_J:

$$\begin{aligned}(t_F,t_J) = {} & (n^{-2},n^{-1}),\ (n^{-1},1),\ (1,n),\ (n,n^2),\\ & (n^{-1},n^{-2}),\ (1,1),\quad (n,n),\\ & (n,1).\end{aligned}$$

Note that the dependence of t_F and t_j on n is at hand since $F(x) \in \mathbb{R}^n$, $J(x) \in L(\mathbb{R}^n)$. Note that $t_F = 1$ means that one evaluation of a function value is about as expensive as the solution of a linear system of order n, for large n. Finally, note that, for algorithms which do not use analytic expressions for the jacobian, only four choices are left because $m_J = 0$ in that case. We derive final conclusions about the usefulness of P_{j_i}, and hence of A_{j_i}, using the efficiency results, which completes the comparison of a given set of algorithms.

7.5.2. Design of comparison of basic algorithms

In section 6.7 we described 18 basic Newton-like algorithms. Furthermore, we have two basic algorithms of component wise approximation: BW and BT (see section 7.1). The third algorithms of component wise approximation, BTM, is in fact a modification of BT, which reuses old jacobian information at the end of each iteration step. Essentially, this is an additional feature in BT, which may be compared with conditional use of fixed approximation of the jacobian in Newton-like algorithms. Evaluation of this feature is discussed in subsection 7.5.3.

The 20 basic algorithms are grouped into 6 groups, such that for each group we can take the same values for t_S and t_I. This makes comparison relatively easy within a group. For this grouping we assumed that computation of the step length factor is small relative to the time required for solving the linear system. This is true for large n. For small n, we have very small values for t_S and t_I, which are relatively negligible. The various groups with the a-priori knowledge about relevant quantities are listed below.

1. ASW, AS, AB, AI, AE;
   $t_I = 0$, $t_S = 1$, $m_S = m_J$.
2. GAS, GAB, GAI;
   $t_I = 0$, t_S given in table 7.1 (ALGOL 60 results, column 6), $m_S = m_J$.
3. DSW, DS, DB, DI, DE;
   $t_I = 0$, $t_S = 1$, $m_J = 0$.
4. GDS, GDB, GDI;
   $t_I = 0$, t_S given in table 7.1 (column 6), $m_J = 0$.
5. U1S, U2S;
   t_I and t_S given in table 7.2, $m_J = 0$.
6. BW, BT;
   $t_I = 0$, t_S given in table 7.2 (differs for both algorithms),
   $m_J = 0$.

Note that group 6 is an exception to the rule in that t_I and t_S are equal for algorithms within a group. BW and BT are taken together as both are implementations of the same method. Note also that we choose $t_I = 0$ in groups 1, 2, 3 and 4 which agrees with the structure of these algorithms described in section 6.7, where the matrix decomposition is performed together with the computation of data in each iteration step (see also section 6.4). In U1S and U2S we have to calculate an approximation to the inverse jacobian (inversion of difference approximation) in the initialization phase.

table 7.2.

t_I and t_S (in standard time) for some algorithms (based on ALGOL 60 programs).

n	t_I(U1S, U2S)	t_S(U1S, U2S)	t_S(BW)	t_S(BT)
2	1	1	2	2
13	2	0.3	4	4
24	2	0.1	4	5
35	2	0.1	4	5
46	2	0.1	4	6

Comparison of the basic algorithms is performed in the following steps.

1. Comparison of the algorithms of the six groups separately in the way as described in subsection 7.5.1.
2. Comparison of algorithms using the analytic jacobian.
3. Comparison of algorithms using difference approximation.

Some conclusions will be stated based on these comparisons. However, final conclusions can only be made after testing the additional features, which are not all applicable to all basic algorithms.

7.5.3. Design of evaluation of additional features and special properties

For the additional features and special properties, given in section 6.8 up to 6.11, we choose specific sets of test problems, to test their effectiveness and usefulness.

A. *Test problems for testing the convergence criteria.*

The testproblems are chosen such that the solutions are known, or can be computed, in almost full machine precision.

1. problem 1, $n = 5,6,7,8,9,10,13,24$;

   $\delta_f = \delta_{rx} = \delta_{ax} = 10^{-3},\ 10^{-7},\ 10^{-11}.$
2. problem 7, $n = 2$, $c = 10^p$, $p = 1,3,5,8$;

   $\delta_f = \delta_{rx} = \delta_{ax} = 10^{-3},\ 10^{-7},\ 10^{-11}.$
3. problem 15, $n = 4$;

   $\delta_f = \delta_{rx} = \delta_{ax} = 10^{-p}$ $(p=1,2,\ldots,10)$.

Note that problem 1 has reasonably small values for $\bar{\kappa}_*$ and $\bar{\omega}_*$, problem 7 has slowly increasing values for $\bar{\omega}_*$ but small $\bar{\kappa}_*$. For problem 15 the jacobian has rank 2 at the solution $(n=4)$.

B. *Test problems for testing conditional use of approximation by updating or fixed approximation.*

It is expected that these features work well if the jacobian varies only slowly around the solution ($\omega(x^*)$ small), but will have no effects if this is not true.

1. problem 4, $n = 13$, $c = 10$ and 100;

   $\bar{\omega}_* = 4$ and 300 respectively.

2. problem 7, $n = 2$, $c = 0.1,1,5,10,50,100,10^3,10^4,10^5$ and 10^8;

   $\bar{\omega}_*$ grows slowly from 0.09 up to $7_{10}3$, for increasing c.

3. problem 8, $n = 2,13,24,35$ and 46;

   $\bar{\omega}_* \leq 0.2$.

4. problem 1, $n = 5,6,7,8,9,10,13$;

   $\delta_f = \delta_{rx} = \delta_{ax} = 10^{-3}, 10^{-7}, 10^{-11}$, in order to check the effect of these features on the accuracy obtained.

C. *Test problems for testing scaling.*

1. problem 1, $n = 24$;

   scaling based on the jacobian at x_0 spoils the small condition number at x^*.

2. problem 2, $n = 3$, $c = 10^p$ $(p=1,2,\ldots,11)$;

   scaling based on the jacobian at x_0 does not have very much effect on the large condition number at x^*.

3. problem 10, $n = 2,13$, with s_r and s_c chosen as in table I.10;

   scaling decreases the condition number on the whole domain, if s_r is small.

4. problem 11, $n = 2,13$, with s_r and s_c chosen as in table I.11;

   scaling decreases the condition number on the whole domain if s_r is small.

5. problem 16, $n = 2$, $c = 10^p$ $(p=0,1,\ldots,5)$;

   scaling based on $J(x_0)$ hardly has any effect on the scaling at x^*.

D. *Testproblems for testing reduction of problems with linear components.*

problem 1, for $n = 2,3,\ldots,9,10,13,24$;

   this function has n-1 linear function components.

These special properties and additional features are tested only for the basic algorithms selected after evaluation. Based on a qualitative discussion of the experimental results obtained in this way we select useful combinations of basic algorithms and features.

7.5.4. Design of final experiments

Based on the comparisons designed in subsection 7.5.2 and 7.5.3 we shall give final conclusions about robustness, reliability and efficiency of combinations of algorithms (basic algorithms with certain additional features). These conclusions may depend on
- availability of an analytic jacobian,
- the order,
- the values of t_F and t_J.

7.6. EXPERIMENTAL EVALUATION OF BASIC ALGORITHMS

7.6.1. Algorithms ASW, AS, AB, AI and AE

The experimental results are given in appendix II.1. For these algorithms we have: $t_I = 0$, $t_S = 1$, $m_S = m_J$.

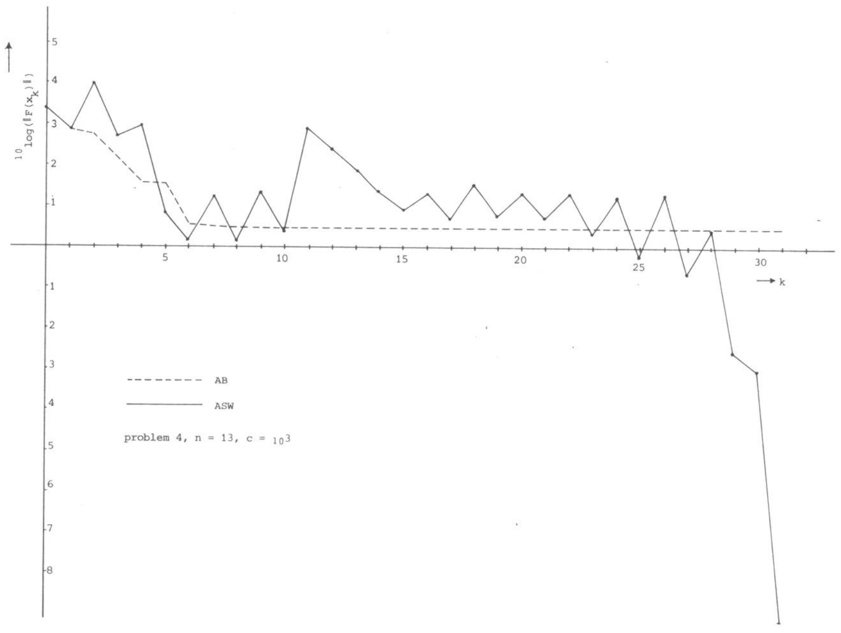
Performance tables

The results for problem 1 show the effect of an increasing condition number of the jacobian matrix at the initial point. Restrained algorithms appear to be favourable, as these solve most problems and efficiently so. The performance of AI is the best. AB fails to solve the problem for $n = 24$ and 35 and AE fails three times ($n = 8,10,13$). AS fails for $n \geq 5$ due to the special failure criterion (6.53) for $n = 5,6,7$ and to numerical singularity for $n \geq 8$. ASW solves the problem for $n = 5,6,7$ but at the cost of relatively many iteration steps.

The results for problem 2 show the effect of an increasing condition number at the starting guess as well as at the solution. Particularly at the solution the condition number increases fast ($\bar{\kappa}_* = 10^7$ for $c = 10^6$). This causes the failure of AS, AB, AI and AE for $c \geq 10^6$, due to failure criteria (6.53) and (6.54). ASW is apparently superior to the other algorithms. It does not use these failure criteria and the bad condition number at the solution does not cause failure, although the number of iteration steps required increases. AE performs badly. The step length factor obtained by a-priori estimation is pessimistically small, so that slow linear convergence occurs. In fact, it fails four times with the non-informative error message 4 ("too many function evaluations or iterations required").

The results for problem 4 show a somewhat similar effect as for problem 2. For this problem $\bar{\kappa}_*$ as well as $\bar{\omega}_*$ increase rapidly with increasing parameter value, while the other problem indicators remain almost constant. Failure of AS, AB, AI and AE is due to failure criteria (6.53) and (6.54) (with one exception for AI). ASW, which does not use these criteria, fails only three times. Note that the restrained algorithms also perform worse than AS. This suggests that the bad behaviour is due to the restraining and that the restrained algorithms would not perform better if the failure criteria (6.53) and (6.54) would not have been used. In fact, these criteria induce earlier detection of failure. This conjecture is confirmed by the results of GAS, GAB and GAI for this problem (see appendix II.2).

figure 7.1.



Here the restrained algorithms GAB and GAI also solve the problem only for c = 1. The fact that restraining performs so bad, is that it forces the algorithms to search for a solution in the valley of the levelfunction in which the starting point lies. There appears to be no solution in this valley. The strict algorithms jump from one valley to another until an appropriate point is found. This difference in behaviour of restrained and strict algorithms is illustrated in figure 7.1. Clearly, the value of $\| F(x_k)\|$ jumps up and down during all but the last three iteration steps of algorithm ASW. In fact, the behaviour of ASW for n = 13 is rather unpredictable and the algorithm fails three times with non-informative messages. It seems that solving problem 4 for n = 13 with ASW is rather occasional.

Based on the values of the problem indicators in table I.5, we expect problem 5 to be easily solvable. This is confirmed by the results. The number of iteration steps and function evaluations is small and practically the same for all selected orders and all programs.

The results for problem 5a show the effect of random errors in the function and jacobian. All algorithms perform well as long as the errors are not so large that the convergence criteria cannot be satisfied (see subsection 7.5.1). This happens for $p = 10^{-2}$. In that case all algorithms fail and only AB, AI and AE give an informative message.

The results for problem 7 are the same for all algorithms. The number of iteration steps and function evaluations required to solve the problem increases with increasing parameter values (which induce increasing values of $\bar{\omega}_*$, see table I.7).

Based on these results we state some conclusions and make some choices.

- AE performs equally well or worse than AB or AI. The a-priori estimation does not work very well in practice. As, moreover, AE is considerably more complicated than AI and AB, we reject AE.
- AS is never more efficient that ASW. Moreover, AS is considerably less robust than ASW and less reliable than AB or AI. Therefore, AS is rejected also.
- We expect that the performance of AB and AI is almost the same. The small differences that occur seem to be very occasional. Therefore, there is little sense in testing both algorithms on the whole set of test problems. We discard AI because of its more complicated program structure.
- ASW seems to be robust, but not very reliable (6 non-informative failure messages). Moreover, its unpredictable behaviour for problem 4 is unsatisfactory.
- AB seems to be reliable (all failures with informative messages) but not very robust. For some problems AB performs considerably better than ASW (e.g. problem 1).

We shall consider ASW and AB for further testing.

Representative test sets

In table 7.3 we give the relative efficiences $E(n,t_F,t_J)$ (see description 7.13) for the selected orders and values of t_F and t_J. We also give the percentages of problems solved and the numbers of unreliable failures (see section 7.4). Note that all results for AB indicate that the number of steps required is almost always very reasonable. Only once 77 steps are required (problem 4, n = 13, c = $_{10}2$) and once 43 (problem 4, n = 13, c = $_{10}6$). In all other cases at most 17 iteration steps are required. On the contrary, ASW reaches 17 times the upper bound on the number of function evaluations.

table 7.3. $E(n,t_F,t_J)$

t_F	t_J	n=2		n=13		n=24		n=35		n=46	
		ASW	AB	ASW	AB	ASW	AB	ASW	AB	ASW	AB
n^{-2}	n-1	7	8	5	5	5	5	4	4	5	5
n^{-1}	1	11	12	9	9	9	9	9	8	10	10
1	n	16	19	65	68	120	110	160	150	230	230
n	n^2	30	34	800	830	2700	2600	5300	5200	10000	10000
n^{-1}	n^{-1}	9	10	5	5	5	5	4	4	5	5
1	1	13	15	14	15	14	14	14	13	15	15
n	n	22	26	130	140	240	240	330	330	490	490
n	1	18	21	77	83	140	140	190	190	280	280
% solved		92	76	84	68	76	68	72	60	72	60
unrel.fail		1	0	3	0	5	0	6	0	6	0

Conclusions

Efficiency: ASW and AB are equally efficient.

Robustness: ASW is more robust than AB, in the sense that it solves more test problems. However, the behaviour of ASW is sometimes unsatisfactory.

Reliability: AB is very reliable and ASW is not.

7.6.2. Algorithms GAS, GAB and GAI

The experimental results are given in appendix II.2. For these algorithms $t_I = 0$, $m_S = m_J$ and t_S is given in table 7.1 (column 6). Notice that these algorithms use singular value decomposition, so that failure criteria (6.53) and (6.54) as well as the special convergence criteria are not used (see section 6.6).

Performance tables:

The results for problem 1 show that the restrained algorithms GAI and GAB perform best. Failure of these algorithms for n = 24 is due to forcing termination as the number of function evaluations exceeds its upper bound (25 for n = 24). The first step direction vector is very large and a large number of bisection or interpolation steps is required to obtain a reasonable step length factor. GAS obtains an iterate outside the domain of the function.

The results for problem 2 are almost the same for GAB and GAI and there is a slight difference with GAS. Only for very large values of the parameter (very large $\bar{\kappa}_*$, see table I.2) restraining causes a less efficient behaviour.

The results for problem 4 show almost the same picture as for algorithms with triangular decomposition. Restraining causes bad behaviour for these problems.

The results for problem 5 are practically the same for GAS, GAB and GAI and for all selected orders.

The effect of random errors is shown by the results of problem 5a. GAS and GAB are terminated due to too many function evaluations; GAI reports that no progress is obtained due to too high required precision. Note that the number of iteration steps in GAB is less than half of that of GAS. Restraining in GAB requires more function evaluations per step.

Finally, the results for problem 7 show no difference between GAS, GAB and GAI.

We come to the following conclusions and choices.
- The difference between performances of GAB and GAI is very small. Therefore we discard GAI as GAB has a simpler program structure.
- GAB performs best for problem 1 (large $\bar{\kappa}_0$) and GAS performs best for problem 4 (large $\bar{\kappa}_*$ and $\bar{\omega}_*$). It is difficult to draw conclusions about robustness and reliability of GAS and GAB based on these results.

GAS and GAB are considered for further testing.

Representative testsets

In table 7.4 we give the relative efficiencies $E(n,t_F,t_J)$ for the selected orders and values of t_F and t_J, the percentages of problems solved and the numbers of unreliable failures.

table 7.4. ($E(n,t_F,t_J)$)

		n=2		n=13		n=24		n=35		n=46	
t_F	t_J	GAS	GAB	GAS	GAB	GAS	GAB	GAS	GAB	GAS	GAB
n^{-2}	n^{-1}	8	9	38	40	50	50	48	48	54	52
n^{-1}	1	12	13	43	45	55	55	53	53	59	57
1	n	19	21	110	110	180	170	220	220	310	290
n	n^2	33	37	910	950	3100	3000	6200	6100	12000	11000
n^{-1}	n^{-1}	9	10	39	40	50	50	49	48	54	52
1	1	14	16	48	50	61	60	59	58	65	63
n	n	24	27	170	180	310	310	420	420	590	580
n	1	20	23	120	120	200	200	260	250	350	350
% solved		96	92	88	84	80	80	80	76	84	84
unrel. fail		1	1	2	4	3	4	4	6	4	4

Conclusions

Efficiency: GAS and GAB are equally efficient.

Robustness: GAS is slightly more robust than GAB.

Reliability: GAS and GAB are not reliable.

7.6.3. Comparison of algorithms requiring an analytic jacobian

We compare ASW, AB, GAS and GAB. We use tables 7.3 and 7.4, although these tables do not give precisely the efficiency of these algorithms relative to each other (note that I and ℓ' in (7.2) are based on ASW and AB for table 7.3, and on GAS and GAB for table 7.4). Precise computation of these efficiencies will not lead to other conclusions. We obtain the following conclusions.

Conclusions

Efficiency: For very small order ASW, AB, GAS and GAB are almost equally efficient. Otherwise ASW and AB are significantly more efficient than GAS and GAB for problems with cheap function and jacobian evaluations (up to a factor 10, due to the expensive singular value decomposition relative to triangular decomposition). For problems with expensive function and jacobian evaluations (e.g. $(t_F,t_J) = (n,n^2)$), ASW, AB, GAS and GAB are almost equally efficient.

Robustness: GAS is the most robust algorithm (solves 86% of the problems of the representative test sets, GAB 83%, ASW 79% and AB 66%). Algorithms using singular value decomposition are more robust than algorithms using triangular decomposition. Strict algorithms are more robust than restrained algorithms.

Reliability: AB is very reliable (no unreliable failures for all test problems). ASW, GAS and GAB are not reliable.

Other properties: Some kind of problems (e.g. problem 1) can better be solved by restrained algorithms, others (e.g. problem 4) can better be solved by strict algorithms.

Based on these conclusions we suggest to combine the reliable and efficient algorithm AB with the robust and less efficient algorithm GAS. That means that, when AB fails, we subsequently use GAS. Note that failure of AB is usually detected after a few iteration steps. Furthermore, the error messages given in case of failure of the reliable algorithm AB, may help to interpret possible unreliable results of GAS. We combine AB with GAS and not with GAB because (1) GAS is somewhat more robust than GAB and (2) based on the last conclusion above, it is preferable to combine a restrained and a strict algorithm. We shall talk about the *poly-algorithm* AB+GAS. Based on the results of the testing of special features we possibly modify this poly-algorithm.

7.6.4. Algorithms DSW, DS, DB, DI and DE

The experimental results are given in appendix II.3. For these algorithms we have $t_I = 0$, $t_S = 1$ and $m_J = 0$.

Performance tables

The results show a remarkable similarity with the results for ASW, AS, AB, AI and AE, respectively, (see appendix II.1), besides the fact that the difference approximation requires n extra function evaluations in each step. There is a small difference in the results for problem 2, where DS, DB, DI and DE fail for $c \geq {}_{10}5$ and AS, AB, AI and AE for $c \geq {}_{10}6$. This happens because difference approximation yields a greater value for $\hat{e}_k$, so that failure criterion (6.54) is satisfied for smaller values of $\hat{\kappa}_k$ in the difference algorithms. Furthermore, notice the difference in performance of ASW and DSW for problem 4 ($c \geq {}_{10}6$). The large values of $\bar{\kappa}_*$ and $\bar{\omega}_*$ clearly indicate difficulties with computing the difference approximation. The performance of DSW is considerably worse than of ASW for these test problems.

Finally, comparison of the results for problem 5a indicate that numerical errors in the function cause more problems for algorithms with difference approximations than for algorithms using the analytic jacobian. Apart from these observations we obtain similar conclusions as in subsection 7.6.1.

- DE and DS are rejected.
- DI is discarded.
- DSW seems more robust but less reliable than DB.

DSW and DB are considered for further testing.

Representative test sets

In table 7.5 we give the relative efficiencies $E(n,t_F,t_J)$ for the selected orders and values of t_F (note that $m_J = 0$, so that the efficiencies are independent of t_J). Furthermore this table gives the percentages of problems solved and the numbers of unreliable failures. Note that DB has no unreliable failures, while DSW has 22.

Conclusions

<u>Efficiency</u>: DSW and DB are equally efficient.

<u>Robustness</u>: DSW is more robust than DB.

<u>Reliability</u>: DB is very reliable and DSW is not.

<u>table 7.5. ($E(n,t_F,t_J)$)</u>

	n=2		n=13		n=24		n=35		n=46	
t_F	DSW	DB	DSW	DB	DSW	DB	DSW	DB	DSW	DB
n^{-2}	7	9	5	5	5	5	5	5	5	5
n^{-1}	11	12	9	10	9	9	9	9	10	10
1	17	20	68	71	120	120	160	160	240	240
n	31	36	830	870	2700	2700	5600	5600	11000	11000
% solved	88	72	76	64	68	60	60	54	60	54
unr.fail	0	0	3	0	6	0	7	0	6	0

7.6.5. Algorithms GDS, GDB and GDI

The experimental results are given in appendix II.4. For these algorithms $t_I = 0$, $m_J = 0$ and t_S is given in table 7.1 (column 6). These algorithms use singular value decomposition, so that failure criteria (6.53) and (6.54) as well as convergence criterion (6.50) are not used.

Performance tables

Comparing the results of GDS, GDB and GDI with those of GAS, GAB and GAI (appendix II.2), we must realize that a greater error in the approximation to the jacobian may cause another value for the approximate rank, as the approximate rank depends on the approximate error $\hat{e}_k$. Therefore, we have a somewhat different performance of GDS, GDB and GDI as compared with GAS, GAB and GAI. Such an effect does not occur in algorithms using triangular decomposition.

The results for problem 1 show the best performance for GDB and GDI. There is almost no difference between the performance of GDB and GDI for this problem. Notice that GDS performs better than GAS for $n = 6,8,24$, which may be due to a lower approximate rank in GDS.

The results for problem 2 show that the bad condition number of the jacobian, if c is large, causes problems for the algorithms. GDB and GDI terminate for $c \geq 10^8$ with error messages: no progress. This may be due to a too low approximated rank which forces a search direction in a subspace in which the level function can not be decreased.

The results for problem 4 are similar to other results as far as the restrained algorithms GDB and GDI are concerned. GDS performs worse than GAS for $n = 2$, but better for $n = 13$. It strengthens an earlier remark that this problem is hard to solve and causes unpredictable performance of strict algorithms.

The results for problem 5 and 7 show practically no differences between GDS, GDB and GDI.

The results for problem 5a show the effect of numerical errors in the function values. These errors have a strong effect on the difference approximation.

We come to the following conclusions and choices.

- The difference between the performances of GDB and GDI is very small. Therefore, GDI is discarded because of the simpler program structure of GDB.
- GDS performs best for problem 4 and GDB for problem 1. It is difficult to

draw conclusions about robustness and reliability based on these results.

Representative test sets

In table 7.6 we give the relative efficiencies $E(n,t_F,t_J)$ for the selected orders and values of t_F, the percentages of problems solved and the numbers of unreliable failures.

table 7.6. ($E(n,t_F,t_J)$)

	n=2		n=13		n=24		n=35		n=46	
t_F	GDS	GDB	GDS	GDB	GDS	GDB	GDS	GDB	GDS	GDB
n^{-2}	8	9	38	39	50	50	46	46	53	51
n^{-1}	12	13	43	44	55	55	51	51	58	56
1	19	21	100	110	170	170	210	210	300	290
n	33	37	910	930	3000	3000	5900	5900	12000	11000
% solved	92	92	84	76	84	80	84	76	88	84
unr.fail	0	1	1	3	2	3	2	3	1	2

Conclusions

Efficiency: GDS and GDB are equally efficient.

Robustness: GDS is slightly more robust than GDB.

Reliability: GDS and GDB are not very reliable.

7.6.6. Algorithms using difference approximations.

We compare DSW, DB, GDS and GDB using tables 7.5 and 7.6. (Note that these tables do not give precisely the relative efficiencies of these four algorithms.) We can state the following conclusions.

Efficiency: For very small order DSW, DB, GDS and GDB are almost equally efficient. Otherwise DSW and DB are significantly more efficient than GDS and GDB for functions with cheap function evaluations (up to a factor 10 for $t_F = n^{-2}$) and about equally efficient for expensive function evaluations ($t_F = n$).

Robustness: GDS is the most robust algorithm (GDS solves 86% of the problems from the representative test sets, GDB 82%, DSW 70% and DB 61%). Algorithms using singular value decomposition are more robust than those using triangular decomposition. Strict algorithms are more robust than restrained algorithms.

Reliability: DB is very reliable. GDB, GDS and DSW are not very reliable.

Other properties: Some kind of problems can better be solved by restrained algorithms, others by strict algorithms.

Based on these conclusions we suggest, similar to the conclusions in subsection 7.6.3, to use a poly-algorithm: DB + GDS. That means that, when DB fails, we subsequently use GDS. We shall possibly modify this poly-algorithm based on the results of the testing of special features.

7.6.7. Algorithms U1S and U2S

The experimental results are given in appendix II.5. For these algorithms we have $m_J = 0$, t_I and t_S given by table 7.2.

Performance tables

No significant difference is shown by the results for problems 1, 2, 4, 5, 5a and 7. Furthermore, both U1S and U2S do not seem to be very robust or reliable.

Representative test sets

In table 7.7 we give the relative efficiencies $E(n,t_F,t_J)$ for the selected orders and values of t_F, the percentages of problems solved and the numbers of unreliable failures.

table 7.7. ($E(n,t_F,t_J)$)

	n=2		n=13		n=24		n=35		n=46	
t_F	U1S	U2S	U1S	U2S	U1S	U2S	U1S	U2S	U1S	U2S
n^{-2}	10	10	4	4	4	4	3	3	3	3
n^{-1}	12	13	6	6	5	5	4	4	4	4
1	17	18	27	27	38	38	49	49	60	60
n	27	27	300	300	840	840	1600	1600	2600	2600
% solved	80	84	72	72	72	72	72	68	64	68
unr.fail	5	4	7	7	7	7	7	8	9	8

Conclusions

Efficiency: U1S and U2S are equally efficient. Both are considerably more efficient than DB and GDS.

Robustness: U1S and U2S are not very robust (solve both 72% of the problems of the representative test sets (GDS 86%)).

Reliability: U1S and U2S are not reliable.

As U2S uses the simplest and most well known update formula (Broyden's formula) we discard U1S.

7.6.8. Algorithms BW and BT

The experimental results are given in appendix II.6. For these algorithms $t_F = 0$, $m_J = 0$ and t_S is given in table 7.2 (different for BW and BT). In these algorithms a value has to be chosen for a parameter controlling the length of the difference steps in the formulas for approximating the derivatives. We have created some performance tables for two different values of this parameter. These values are $\sqrt[3]{\varepsilon}$ and $\sqrt{\varepsilon}$. However, most tests are performed with the value $\sqrt[3]{\varepsilon}$. Furthermore, in these programs we use a simple convergence criterion in order to avoid additional function evaluations. This criterion is

$$|F^{(j)}(y_{k,j})| < \delta_f, \quad j = 1,\dots,n,$$

for $F(x) = (F^{(1)}(x),\dots,F^{(n)}(x))^T$, $y_{k,j}$ is the iterate in the j-th subiteration step of the k-th iteration step. Finally these programs require evaluation of function components separately. For a given vector of variables only one function component has to be evaluated. In the reported results MF denotes the number of component evaluations divided by the order n and rounded to below.

Performance tables

For problem 1, the smaller value ($\sqrt{\varepsilon}$) for the difference steps yields the best results. The reason for these big differences in behaviour is difficult to explain. It might be due to the fact that the only nonlinear function component is highly nonlinear and therefore its derivative is best approximated with small difference steps. Moreover, the fact that n-1 components are linear might be of influence to this performance too. Note that there is not much difference between the performances of BW and BT for $n \leq 8$, if we choose the same value for the difference step.

The results for problem 2 do not show much difference between the two choices for the value of the difference step. Here BT solves more problems. Note that BT performs worse, and BW hardly any better than the Newton-like algorithms AB and DB.

For problem 4 we see that the smallest value for the difference step performs best. This is understandable as a large Lipschitz constant ($\bar{\omega}_*$ is large for large c) yields a small optimal value for the difference step (see subsection 6.3.3).

The results for problem 5 show no difference between BW and BT.

The results for problem 5a show that the convergence criterion used does not imply that $\|F(\bar{x})\| < \delta_f$ for the computed solution x. BW and BT satisfy the convergence criterion without satisfying this condition on the norm of the function.

The results for problem 7 do not show significant differences between the performance of the algorithms.

We come to the following conclusions.
- Both algorithms are sensitive to the choice of the value for the difference step. A choice, which is appropriate for all problems, cannot be given.
- BW and BT are not reliable.
- BW and BT seem to be not very robust.

Representative test sets

In table 7.8 we give the relative efficiencies $E(n,t_F,t_J)$ for the selected orders and values of t_F, the percentages of problems solved and the numbers of unreliable failures.

table 7.8. $E(n,t_F,t_J)$

	n=2		n=13		n=24		n=35		n=46	
t_F	BW	BT	BW	BT	BW	BT	BW	BT	BW	BT
n^{-2}	14	15	21	24	22	28	23	28	23	32
n^{-1}	18	18	24	27	25	31	26	31	26	35
1	25	26	63	71	96	100	130	140	160	160
n	39	40	570	640	1800	1900	3800	3800	6400	6000
% solved	96	88	92	80	92	76	80	68	84	72
unr.fail	1	1	1	3	2	6	5	7	4	5

Conclusions

Efficiency: BW and BT are almost equally efficient. For small values of t_F, BW and BT are less efficient than DB (up to a factor 4). For large values of t_F, however, BW and BT are more efficient than DB (up to a factor 2). These conclusions are based on the assumption that the computation of n function components for different argument vectors is as expensive as computation of one function vector for a given argument vector. Very often this assumption does not hold, however. In that case BW and BT might become considerably less efficient than DB (up to a factor n if calculating one component is as expensive as calculating the whole vector).

Robustness: BW is more robust than BT. The performance tables show no big difference but BW solves 89% and BT 77% of the test problems from the representative test sets. Note that the robustness of BW is comparable with the robustness of GDS (considering all test results and depending on the choice of the value of the difference step in BW).

Reliability: BW and BT are not reliable.

Conclusions here indicate that BW is the best choice. As BT can be modified as to use old jacobian information and BW can not, a definite choice between BW and BT has to be postponed until this modification of BT is tested.

7.6.9. Summary of conclusions about basic algorithms

1. The use of an analytic jacobian, if available, leads to more robust algorithms (see discussion of performance tables in subsection 7.6.4 and 7.6.5).
2. Generalized algorithms are more robust than other Newton-like algorithms.
3. Generalized algorithms are less efficient than other Newton-like algorithms, particularly for cheap function and jacobian evaluations.
4. Strict algorithms are, on the average, more robust than restrained algorithms, although restrained algorithms perform better for some classes of problems.
5. Restraining with a-priori estimation of the step length factor performs badly.
6. Restraining with bisection and with interpolation yields almost the same performance.
7. The poly-algorithms AB+GAS and DB+GDS seem to be reliable, robust and reasonably effioient algorithms.

8. For efficiency reasons U2S and, in some particular cases, BW and/or BT may be interesting; these algorithms are not reliable; U2S and BT are not robust.

The following algorithms are left for testing the special properties and features:

AB, GAS, DB, GDS, U2S, BW and BT.

7.7. EVALUATION OF SPECIAL PROPERTIES AND FEATURES

7.7.1. Convergence criterion

In appendix II.7.1 we show some experimental results for evaluating the performance of the convergence criteria. Algorithm ASWC is the same as ASW except for the fact that the special convergence criterion (6.50) is not used in ASWC, but (6.51) is used instead. The other algorithms are those selected in section 7.6. The selected test problems have been mentioned in subsection 7.5.3.A.

The results show a typical difference between ASW and ASWC. ASW has always reached the precision required if no error message is given, however, ASWC sometimes delivers a solution in a lower precision than required, without giving an error message (problem 15 with singular jacobian at the solution). Furthermore, for easier problems ASWC often performs one more iteration step than ASW. In fact ASWC terminates too late and obtains higher precision than necessary. We conclude that the convergence criterion (6.50) performs as was expected. It is safer and more accurate than (6.51). This performance is not disturbed by the use of restraining or difference approximation as is shown by the results of AB and ASW for problem 7 and the results of AB and DB. The results for problem 15 show that criterion (6.50) sometimes is too pessimistic, as sometimes the precision is obtained and a failure is reported. However, in our opinion, this is preferable above having not reached the precision without giving any message. Furthermore, notice that the error messages of AB and DB are adequate (not for ASWC as no special failure criteria are used in this algorithm).

Another observation to be made from the results is that ASW, ASWC, AB, GAS, DB and GDS require only one or two more iteration steps to satisfy the convergence criteria with $\delta_f = \delta_{rx} = \delta_{ax} = 10^{-11}$ instead of 10^{-3}. (notice that $\varepsilon = 10^{-14}$, approximately). This shows the (almost) quadratic order of convergence for these algorithms (see theorem 5.20). If the jacobian is singular at the solution then the order of convergence may become linear, which is confirmed by the results for function 15. Furthermore, the asymptotic convergence behaviour of BW and BT is almost as for the Newton-like algorithms above. For U2S we see a lower order of convergence.

Finally we want to point out some remarkable results:

- U2S sometimes performs well for low precision but fails for high precision (problem 1, n = 6,7).
- U2S sometimes performs many iteration steps without gaining any better results (problem 15, precision $\leq 10^{-7}$).
- Asking a higher precision for problem 1 (n = 24) with BT results in one more iteration step and loss of all precision. This suggests a very unreliable behaviour of BT. It is not quite clear whether we may expect similar unreliable behaviour for BW.

Conclusions:

- The convergence criterion (6.50), used in ASW, AB and DB, is safe and accurate. It yields somewhat more efficient algorithms, particularly for easy problems. However, it might lead to unnecessary failure sometimes.
- The convergence criterion (6.51), used in ASWC, GAS, GDS and U2S, sometimes yields results which do not have the precision required.
- The convergence criterion used in BW and BT may occasionally yield very unreliable results.
- Asymptotic convergence of ASW, ASWC, AB, GAS, DB and GDS is at a higher rate than of U2S and is almost the same as of BW and BT.

In view of this conclusions there is no reason to consider ASWC any further.

7.7.2. Conditional use of updating and fixed approximation

As far as the algorithms are concerned which are left for further evaluation (see subsection 7.6.9), only AB and DB are suitable for application of conditional use of updating. The modified algorithms are denoted by ABU and DBU, respectively. Conditional use of fixed approximation is applicable to AB, GAS, DB and GDS. These modified algorithms are denoted by ABF, GASF, DBF and GDSF, respectively. Furthermore, MORÉ & COSNARD [1979] give a modification of BT, which reuses old jacobian information, which will be denoted by BTM. In appendix II.7.2 we show the experimental results for testing these features. The test problems used are mentioned in subsection 7.5.3.B. For convenience we also recall the results for the basic algorithms.

Update approximation

The results show a successful performance of ABU and DBU. One more iteration step is incidentally required to solve problems 4, 7 and 8. However, the number of jacobian evaluations required by ABU, and the number of function evaluations required by DBU is always less than for AB and DB, respectively. A similar performance was obtained for problem 1 if the required precision is not very high ($\delta_f = \delta_{rx} = \delta_{ax} = 10^{-3}, 10^{-7}$). Only for high precision (10^{-11}) ABU and DBU use one more iteration step for all orders tested. Clearly, the positive effect of the use of conditional updating on the efficiency depends on the expensiveness of a jacobian evaluation (for ABU) or a function evaluation (for DBU). ABU is preferable, only if the jacobian evaluation is not very cheap, relative to a standard time unit. DBU is preferable, only if the function evaluation is not very cheap relative to a standard time unit. Notice that an iteration step with updating in ABU and DBU still requires a triangular decomposition of the jacobian approximation.

Fixed approximation

An iteration step with fixed approximation requires only $o(n^2)$ basic arithmetical operations and is therefore negligible relative to a normal step for large n. That makes conditional use of fixed approximation attractive. However, the results show that use of fixed approximation sometimes yields a considerable increase of the number of iteration steps required, particularly if high precision is required (e.g. problem 1, n=13). Furthermore, use of fixed approximation decreases the robustness and reliability of the algorithms (see GASF and GDSF for problem 1, n=6,9 and note that for n=6 the precision, 10^{-3}, is not reached).

Reusing old jacobian information in BT

The effect of this feature on problem 4, 7 and 8 is very small. For this reason we did not perform precision tests. Moreover, we discard BT and BTM based on the conclusions of subsection 7.6.8.

Table 7.9 gives an illustration of the effect of conditional updating and fixed approximation on the efficiency of the algorithms. We give in this table the times (see (7.1)) required by the algorithms to solve problem 8 (n = 13,46).

This table also shows that ABU is about as efficient as AB for t_J less than or equal to one standard time unit. For large t_J (n or n^2 standard time units), ABU may be up to 30% more efficient than AB. DBU is about as efficient as DB as long as t_F is less than or equal to 1/n standard time unit. For larger t_F, DBU may become about 30% more efficient than DB. Use of conditional fixed approximation may increase the efficiency from 0 up to 50%.

For n = 46 and $t_F \geq 1/n$, the efficiencies of ABU and ABF, and of DBU and DBF are about the same.

table 7.9. (T(P,p) for several algorithms)

	t_F	t_J	AB	ABU	ABF	DB	DBU	DBF	GAS	GASF	GDS	GDSF
problem 8, n=13	$1/n^2$	$1/n$	4	4	2	4	4	2	40	24	40	24
	$1/n$	1	8	7	4	8	7	4	45	27	45	27
	1	n	61	48	34	61	48	34	110	69	110	69
	n	n^2	750	580	420	750	580	420	960	610	960	610
	$1/n$	$1/n$	5	5	3	8	7	4	41	25	45	27
	1	1	13	12	10	61	48	34	51	33	110	69
	n	n	120	110	110	750	580	420	180	140	960	610
	n	1	73	72	82	750	580	420	120	110	960	610
problem 8, n=46	$1/n^2$	$1/n$	4	4	3	4	4	3	50	30	50	30
	$1/n$	1	8	7	6	8	7	6	55	33	55	33
	1	n	190	150	150	190	150	150	290	170	290	170
	n	n^2	8700	6600	6600	8700	6600	6600	11000	6700	11000	6700
	$1/n$	$1/n$	4	4	3	8	7	6	50	30	55	33
	1	1	13	12	11	190	150	150	61	39	290	170
	n	n	420	370	370	8700	6600	6600	560	440	11000	6700
	n	1	240	240	240	8700	6600	6600	330	310	11000	6700

Conclusions

- Use of conditional updating may increase the efficiency of the algorithms (up to about 30%), particularly if $t_J > 1$ (for AB) and $t_F > 1/n$ (for DB) and if the required precision is not very high.
- If very high precision is required (almost machine precision), than it is better not to use conditional updating.
- Conditional updating has little effect on the robustness and reliability of the algorithms.
- Use of conditional fixed approximation may increase the efficiency of the algorithms (up to about 50%), if relatively low precision is required ($< \sqrt{\varepsilon}$, say).

- If high precision is required, then conditional fixed approximation may seriously decrease the efficiency, robustness and reliability of the algorithms.
- Reusing old jacobian information in algorithm BT seems not to yield an increase of the efficiency of this algorithm, hence BT and BTM are discarded.
- If a method for updating a triangular decomposition would have been used in algorithms ABU or DBU, then increase of efficiency would be about as high as for conditional fixed approximation. In ABU and DBU the approximate jacobian is updated and a new triangular decomposition has to be performed. We shall not work out updating of triangularly decomposed matrices here. We just notice that it might increase the efficiency of ABU or DBU.

7.7.3. Scaling

Implicit scaling is only applicable to the selected basic algorithms AB and DB. We consider two choices for the implicit scaling matrix A (see remark 4.18 and (sub)section 6.4.5 and 6.10): (i) A = H_0, denoted by ABIS1 and DBIS1 and (ii) A = H_k (k=0,1,...), denoted by ABIS2 and DBIS2. Explicit scaling is applicable to all selected basic Newton-like algorithms AB, GAS, DB, GDS and U2S. Application of explicit scaling, as described in section 6.10, to BW runs into problems. BW does not use approximations to the jacobian, neither at the initial point, nor at any other point obtained during the process. Thus, application of scaling would require an additional evaluation of a jacobian approximation, which is very unattractive. Therefore, we do not consider BW with scaling. The basic Newton-like algorithms with explicit scaling are denoted by SCAB, SCGAS, SCDB, SCGDS and SCU2S, respectively. The experimental results are given in appendix II.7.3. We use the test problems mentioned in subsection 7.5.3.C. In ABIS1/2 and DBIS1/2 we use the normal stopping criteria on the function and variables. If explicit scaling is used, then the stopping criteria are evaluated for the scaled function and variables. The stopping criteria use vector norms and are, therefore, based on the largest vector element. If we would adapt the required precision using the norms of the scaling matrices, then we would require too high precision for the small components of the original function and/or argument vector.

The comparison of AB with its modifications ABIS1, ABIS2 and SCAB shows that the number of failures is reduced best with explicit scaling (SCAB). ABIS2 is better only for problem 2. This can be explained by the observation that scaling based on the jacobian at the initial point (as is done in SCAB) reduces the condition number due to bad scaling at the solution only slightly for this problem (see table I.2). We expect that reusing of SCAB after failure, which yields rescaling, might give better performance. For the same reason, scaling does not yield a better performance for problem 16. Note that ABIS2 performs worse than the other algorithms for this problem. The results of ABIS1 for problems 10 and 11 appear to be worse than of AB. Sometimes ABIS1 fails if AB does not and sometimes vice versa. Therefore, the results indicate that implicit scaling with $A = H_0$ is not a useful strategy. Implicit scaling with $A = H_k$ yields a somewhat better performance. However, for problems 10 and 11 it fails as many times as AB. One reason for this is that computation of $\| B_k \backslash F_k \|^2$ (which is the approximation to the level function at x_k) yields large errors if the condition number of B_k is large. Hence, the problems that we want to avoid by using scaling, show up in the scaling itself. This behaviour is confirmed by the fact that ABIS2 fails more often with error message 2 (no progress during restraining, possibly due to error in function) than the other algorithms. SCAB reduces the number of failures of AB with about 50% for this set of test problems. Particularly, for problem 10 and 11 we obtain a reduction of the number of failures from 17 to 5. Note that SCAB never performs worse than AB for these test problems. For DB, DBIS1/2 and SCDB we can make similar observations as for AB, ABIS1/2 and SCAB.

The results for the generalized algorithms show much less differences between algorithms with and without scaling. An important reason for this is that generalized algorithms do not necessarily fail if the jacobian approximation is singular due to bad scaling. Moreover, for some problems the scaled algorithms fail while the unscaled algorithms do not. Therefore, scaling seems to be unattractive for generalized algorithms.

Finally, explicit scaling for the inverse-updating method U2S increases robustness as well as efficiency of the algorithm slightly. Note that this suggests that explicit scaling may also be useful in algorithms using conditional updating.

Conclusions

- Implicit scaling does not always yield a performance which is at least as good as for the unscaled algorithms. Moreover, the algorithms are less robust than if explicit scaling is used. Therefore, implicit scaling is rejected.
- Explicit scaling may considerably increase the robustness of the non-generalized algorithms AB and DB, while efficiency is increased slightly.
- Explicit scaling for generalized algorithms has, on the average, only little effect. For particular problems the effect is sometimes negative. Therefore we reject explicit scaling for generalized algorithms.
- The effect of explicit scaling on inverse-updating algorithms is positive but small.

7.7.4. Reduction of problems with linear components

In appendix II.7.4 we give some results for the case that problem 1 is reduced to a one-dimensional nonlinear problem (see section 6.11 and theorem 1.27). Our intention is to show the effect of the reduction method, not to show the superiority or inferiority of the one-dimensional equation solver that is used. The method chosen for solving the one-dimensional problem is given in BUS & DEKKER [1975]. As this method requires an interval in which the zero is searched for, we first search for a point in which the function has a sign opposite to the one at the initial point. A modification of Newton's method (if the analytic jacobian is available) or a linear interpolation method is used to find such a point. A precise description is given in the description of the ALGOL 60 package based on the results of this thesis. (see BUS [1980]).

Note that, for all given orders, we only require one singular value decomposition of a $n \times n$ matrix and 11 evaluations of the nonlinear function component. The overhead cost in the one-dimensional equation solver is negligible with respect to the time needed for the function evaluations and matrix decomposition.

Our conclusion is that for problems with linear function components, the above method provides a more robust and efficient algorithm, than solving the system as if all function components would be nonlinear.

7.8. CONCLUSIONS

Based on the experimental evaluation, described in sections 7.6 and 7.7, we obtain the following useful algorithms.

1. If the analytic jacobian is available:

   (SC)AB(U) + GAS.

2. If no analytic jacobian is available:

   (SC)DB(U) + GDS or
   (SC)U2S or
   BW.

Here "+" means that, when the first algorithm fails, we subsequently use the second algorithm. Parentheses denote that the feature should be optional to the user of the algorithm. Conditional updating should be made optional, because it is only preferable if the required precision is not very high. Scaling should be made optional because there might be problems for which scaling is undesirable. Moreover, as we only used a restricted test set for testing the scaling, it is not sure that it always yields better performance. If scaling is used in AB(U) or DB(U) we allow only 20 iteration steps. If no solution is found within this number of steps, we check whether rescaling might be useful and, if this is true, we use AB(U) or DB(U) again with at most 20 iteration steps. If it fails again GAS or GDS is used subsequently. If no scaling is allowed, we perform at most 40 iteration steps. These upper bounds on the number of iteration steps are based on the fact that the test results show that AB(U) or DB(U) almost always terminate in a reliable way within this number of iteration steps. We do not consider conditional fixed approximation, as we think that the loss of reliability in AB or DB and the loss of robustness in GAS or GDS is more important than the gain of efficiency if fixed approximation is used instead of update approximation. We give precise descriptions in ALGOL 68 of (SC)AB(U) + GAS and (SC)DB(U) + GDS at the end of this section. We will call these final poly-algorithms SNOLEQJ (Solution of NOnlinear EQations with analytic Jacobian) and SNOLEQ, respectively.

The choice of BW follows from the conclusions in subsections 7.6.8 and 7.7.2. We notice that our conclusions about BW, BT and BTM differ from the conclusions given by MORÉ & COSNARD [1979]. They conclude that BT is preferable above BW, and BTM above BT. We like to emphasize that the choice of the parameter that controls the difference step is a crucial one, which may change results drastically. Furthermore, the set of test problems of Moré and Cosnard seems to be chosen somewhat arbitrarily and might be not very representative for the class of problems considered here. Note that, up to now a definite choice between SNOLEQ and BW is difficult to make.

The choice of U2S is due to the fact that it is a very efficient algorithm, although it is not reliable nor robust.

If the analytic jacobian of the problem is available and is to be used, then the only choice left is SNOLEQJ. If no analytic jacobian is available, then we can choose one of the three algorithms SNOLEQ, BW and (SC)U2S. Experimental results for these algorithms are reported in appendix II.8. Final conclusions about these algorithms, based on these results as well as on other evaluation criteria, are given in chapter 8.

Description in ALGOL 68

It is assumed that the prelude nlsprl is extended with declarations of the Newton-like algorithms: abu, scabu, gas, dbu, scdbu and gds.

```
.proc  snoleq = .bool :
.begin
   .proc  nlp = .bool :
   .begin  .bool  ok:= .false ;
      .if  nongener
      .then  .if  scale
         .then  maxit:= 20; .if  .not  (ok:= scdbu)
            .then  torrix(warning,
               "second attempt of nongeneralized method, rescaled");
               ok:= scdbu
            .fi
         .else  maxit:= 40; ok:= dbu .fi ;
         .if  .not  ok .then  torrix(warning,
            "generalized method will be tried if allowed")
      .fi  .fi ; .if  .not  ok .and  gener
      .then  maxit:= 40; ok:= gds .fi ;
      ok
   .end  # nlp #;

   .if  linpart .then  reducenewt(nlp) .else  nlp .fi
.end  # snoleq #;
```

```
.proc  snoleqj = .bool :
.begin
   .proc  nlp = .bool :
   .begin  .bool  ok:= .false ;
      .if  nongener
      .then  .if  scale
         .then  maxit:= 20; .if  .not  (ok:= scabu)
            .then  torrix(warning,
               "second attempt of nongeneralized method, rescaled");
               ok:= scabu
            .fi
         .else  maxit:= 40; ok:= abu .fi ;
         .if  .not  ok .then  torrix(warning,
            "generalized method will be tried if allowed")
      .fi  .fi ; .if  .not  ok .and  gener
      .then  maxit:= 40; ok:= gas .fi ;
      ok
   .end  # nlp #;

   .if  linpart .then  reducenewt(nlp) .else  nlp .fi
.end  # snoleqj #;
```

CHAPTER 8

FINAL RESULTS AND CONCLUSIONS

In appendix II.8 we give the results for SNOLEQ and SNOLEQJ, with and without scaling, and for SCU2S. For convenience we recall the results of U2S and BW. We have not given the results for problem 5a (n=35), as these tests are relatively expensive and required to test reliability mainly. We already have enough information about the reliability of each algorithm separately. In the ALGOL 60 tests described here, we only perform a second call of the nongeneralized method if rescaling yields a scaling matrix with a condition number ≥ 100.

Performance tables

The results for problem 1 show that scaling is not always favourable. For n = 6, 8 and 10, SNOLEQ(J) with scaling fails, but without scaling the performance is nice. As is seen in appendix I.1, scaling based on the jacobian matrix at x_0 may have a negative effect on the condition number of the jacobian at the solution. We like to note also that algorithm BW performs relatively well because the problem has n-1 linear components, so that approximation of these components is exact.

The results for problem 2 show that scaling is preferable in SNOLEQ(J) for this problem. Note that scaling has no effect for U2S. If no scaling is performed, then SNOLEQ(J) switches to the generalized algorithm after a few iterations. Clearly SNOLEQ(J) with or without scaling is preferable for this problem.
The results for problem 4 show a behaviour of SNOLEQJ (with and without scaling) which is comparable to the behaviour of ASW (see appendix II.1), which was the best among the tested algorithms which use the analytic jacobian. However, SNOLEQ performs somewhat worse than DSW. Scaling yields sometimes better, sometimes worse results.

The results for problem 5 show a good behaviour for all algorithms. Note that the number of jacobian evaluations is one less than the number of iteration steps for SNOLEQJ. This indicates that in the third iteration step updating is used to approximate the jacobian. The number of function evaluations required indicate that this also holds for SNOLEQ.

The results for problem 7 are as was expected. Scaling hardly has any effect.

Representative test sets

In table 8.1 we give the relative efficiencies $\bar{E}(n,t_F,t_J)$ (see (7.2)) for SNOLEQ(J), with and without scaling, SCU2S, U2S and BW, for the selected orders and values of t_F and t_J. Furthermore, we give the percentages of problems solved for each order and the mean value of these percentages for all selected orders.

Conclusions

Efficiency:
For small problems the efficiency is about the same for all algorithms. Therefore, we restrict our conclusions to problems of order greater than or equal to 13.

- SNOLEQJ, using the analytic jacobian, is considerably more efficient than SNOLEQ, if the jacobian evaluation time satisfies $t_J < nt_F$. In fact, for $(t_F,t_J) = (n^{-1},n^{-1})$, (1,1), (n,n) and (n,1), SNOLEQJ is the most efficient algorithm.
- Conditional use of updating in SNOLEQJ might be efficient only if t_J is greater than or equal to one standard time unit (see definition 7.1).
- Scaling slightly improves the efficiency on the whole, but it might be opposite for particular problems.
- SNOLEQ(S) is more efficient than BW for cheap functions and about as efficient for expensive functions, provided that calculation of n components of the function vector at different points is as expensive as calculation of a complete function vector at one point. In the case that the last calculation is cheaper than the first, the numbers for BW given in table 8.1 are not valid. BW will be less efficient in that case.
- (SC)U2S is the most efficient algorithm not using analytic derivatives (up to 2 or 3 times as efficient as SNOLEQ(S) and BW)

table 8.1.

Relative efficiencies for selected programs

n	t_F	t_J	SNOLEQJS	SNOLECJ	SNOLEQS	SNOLEQ	SCU2S	U2S	BW
2	n^{-2}	n^{-1}	7	7	7	7	9	10	11
	n^{-1}	1	9	10	10	11	11	13	14
	1	n	15	16	16	17	15	18	20
	n	n^2	26	27	28	29	24	27	31
	n^{-1}	n^{-1}	8	8	10	11	11	13	14
	1	1	12	12	16	17	15	18	20
	n	n	20	21	28	29	24	27	31
	n	1	17	17	28	29	24	27	31
% solved			100	100	96	96	88	84	96
13	n^{-2}	n^{-1}	5	6	5	8	4	5	20
	n^{-1}	1	8	9	8	11	6	6	23
	1	n	47	51	47	53	26	27	59
	n	n^2	550	590	550	600	290	290	530
	n^{-1}	n^{-1}	5	7	8	11	6	6	23
	1	1	13	14	47	53	26	27	59
	n	n	110	120	550	600	290	290	530
	n	1	79	80	550	600	290	290	530
% solved			88	88	84	80	68	72	92
24	n^{-2}	n^{-1}	5	8	7	9	3	3	22
	n^{-1}	1	8	11	10	13	4	4	25
	1	n	82	91	85	90	36	38	96
	n	n^2	1900	2000	1900	1900	800	840	1800
	n^{-1}	n^{-1}	5	8	10	13	4	4	25
	1	1	13	17	85	90	36	38	96
	n	n	210	220	1900	1900	800	840	1800
	n	1	140	150	1900	1900	800	840	1800
% solved			88	88	84	80	72	72	92
35	n^{-2}	n^{-1}	5	8	5	5	3	3	25
	n^{-1}	1	8	11	8	8	4	4	28
	1	n	120	130	120	120	46	49	140
	n	n^2	3900	4100	4000	4000	1500	1600	4100
	n^{-1}	n^{-1}	5	8	8	8	4	4	28
	1	1	13	17	120	120	46	49	140
	n	n	310	320	4000	4000	1500	1600	4100
	n	1	200	210	4000	4000	1500	1600	4100
% solved			80	80	84	76	68	68	80
46	n^{-2}	n^{-1}	8	10	10	15	3	3	24
	n^{-1}	1	12	14	13	19	4	4	27
	1	n	180	190	190	190	58	60	170
	n	n^2	8100	8100	8200	8100	2600	2600	6700
	n^{-1}	n^{-1}	8	10	13	19	4	4	27
	1	1	18	20	190	190	58	60	170
	n	n	470	480	8200	8100	2600	2600	6700
	n	1	310	310	8200	8100	2600	2600	6700
% solved			88	88	84	84	76	68	84
mean %			89	89	86	82	74	73	89

Robustness:

- SNOLEQJ is somewhat more robust than SNOLEQ.
- BW is about as robust as SNOLEQJ.
- (SC)U2S is considerably less robust than the other algorithms.
- Scaling may improve the robustness, but there are problems for which scaling has a negative effect on the performance of the algorithms.

Reliability:

- (SC)U2S is not reliable (see subsection 7.6.7).
- BW is not reliable (see subsection 7.6.8).
- SNOLEQ(J)(S) is reliable if the generalized algorithms are not used. If these are used, then one or two informative messages from the non-generalized algorithm might be given, together with a possibly non-informative final message.

Other conclusions obtained from experiments:

- BW requires the choice of a value controlling the step sizes in difference formulas approximating the derivatives. This choice may be critical and is sometimes difficult to make. In the other algorithms, all quantities needed are approximated in the algorithms.
- Use of conditional updating is profitable as long as the error in the approximation to the solution is not required to be almost as small as the machine precision. The profit in SNOLEQJ is small or nought if the jacobian evaluation time is about 1/n or less standard time units.
- If some components of the problem to be solved are linear, then reduction of the problem to a smaller nonlinear problem is advisable.

Storage:

- The storage required in SNOLEQ(J)(S) is $2n^2+o(n)$.
- The storage required in (SC)U2S and BW is $n^2+o(n)$.

Mathematical basis:

- Convergence theory (global, (semi-)local) for the algorithms used in SNOLEQ(J)(S) is given in chapter 5. Based on this theory a mathematical justification for these algorithms is given in chapter 6.
- For BW and U2S (semi-)local convergence results are given in literature (see BROWN [1969, 1973], BROYDEN [1970a] and DENNIS & MORE [1977]).

Program structure and ease of use:

- Program structure of SNOLEQ(J)(S) is based on the modular structure of the ALGOL 68 system given in chapter 6. These programs are longer and more complicated than BW and U2S. However, SNOLEQ(J)(S) is easy to use. A users manual for these algorithms in ALGOL 60 is given in BUS [1980].

Summary

SNOLEQ and SNOLEQJ, with possible scaling and reduction for problems with linear components, are robust, reliable and reasonably efficient algorithms, with a sound mathematical basis and defined in a structured and modular way. They are easy to use. (SC)U2S may be preferable in those cases in which efficiency is much more important than robustness and reliability. We prefer SNOLEQ(S) above (SC)U2S for use in a software library, because in that situation robustness and reliability are important, as such software will be used for many different problems and by many different users. We also prefer SNOLEQ(S) above BW although robustness and efficiency, particularly for expensive functions, are almost the same. Our preference is based on the unreliability of BW, the fact that a parameter value has to be chosen which is difficult to choose and because of the component wise evaluation of the function. Finally, if the analytic jacobian can be obtained and $t_J \leq nt_F$, then SNOLEQJ is preferable above one of the other algorithms.

REFERENCES

BRANIN jr., F.H., [1972], *Widely convergent method for finding multiple solutions of simultaneous nonlinear equations,* IBM J. Res. Develop., 504-522.

BRENT, R.P., [1973a], *Some efficient algorithms for solving systems of nonlinear equations,* SIAM J. Numer. Anal. 10, 327-343.

BRENT, R.P., [1973b], *Algorithms for minimization without derivatives,* Prentice Hall, Englewood Cliffs, N.J.

BREZINSKI, C., [1975], *Numerical stability of a quadratic method for solving systems of nonlinear equations,* Computing 14, 205-212.

BROWN, K.M., [1969], *A quadratically convergent Newton-like method based upon Gaussian elimination,* SIAM J. Numer. Anal. 6, 560-569.

BROWN, K.M., [1973], *Computer oriented algorithms for solving systems of simultaneous nonlinear algebraic equations,* in: Numerical solution of systems of nonlinear algebraic equations, BYRNE, G.D. & C.A. HALL (eds), Academic Press, New York.

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

BROYDEN, C.G., [1969], *A new method of solving nonlinear simultaneous equations. Algorithm 44 CACM.* Comput. J. 12, 94-99.

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

BROYDEN, C.G., [1970b], *Recent developments in solving nonlinear algebraic systems,* in: Numerical methods for nonlinear algebraic equations, RABINOWITZ, P.(ed.), Gordon & Breach, 61-73, London.

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

BROYDEN, C.G., [1973], *Quasi-Newton, or modification methods,* in: Numerical solution of systems of nonlinear algebraic equations, BYRNE, G.D. & C.A. HALL (eds), Academic Press, New York.

BUS, J.C.P., [1972], *Linear systems with error estimation and iterative refinement* (dutch), LR 3.4.19, Mathematisch Centrum, Amsterdam. See also HEMKER [1979] section 3.1.1.1.1.1.1.

BUS, J.C.P., [1977], *Convergence of Newton-like methods for solving systems of nonlinear equations,* Numer. Math. 27, 271-281.

BUS, J.C.P., [1980], *An ALGOL 60 package for the solution of nonlinear equations,* Mathematisch Centrum, Amsterdam, to appear.

BUS, J.C.P. & T.J. DEKKER, [1975], *Two efficient algorithms with guaranteed convergence for finding a zero of a function,* ACM Trans. Math. Software 1, 330-345.

BUSINGER, P.A., [1971], *Monitoring the numerical stability of Gaussian elimination,* Numer. Math. 16, 360-361.

CROWDER, H.P., R.S. DEMBO & J.M. MULVEY, [1977], *Guidelines for reporting computational experiments in mathematical programming,* Harvard Univ. HBS 77-8.

DAVIDON, W.C., [1959], *Variable metric methods for minimization,* A.E.C. Research & Development Report ANL 5990.

DEKKER, T.J., [1968], *ALGOL 60 procedures in numerical algebra,* part 1, Tract 22, Mathematisch Centrum, Amsterdam.

DEKKER, T.J., [1971], *Numerical Algebra* (dutch), Syllabus 12, Mathematisch Centrum, Amsterdam.

DENNIS jr., J.E., [1971], *On the convergence of Broyden's method for nonlinear systems of equations,* Math. Comp. 25, 559-567.

DENNIS jr., J.E., [1975], *A brief survey of convergence results for quasi-Newton methods,* Cornell Univ. rep. TR 238.

DENNIS jr., J.E. & J.J. MORÉ, [1977], *Quasi-Newton methods, motivation and theory,* SIAM Rev. 19, 46-89.

DEUFLHARD, P., [1974a], *A modified Newton method for the solution of ill-conditioned systems of nonlinear equations with application to multiple shooting,* Numer. Math. 22, 289-315.

DEUFLHARD, P., [1974b], *A relaxation strategy for the modified Newton method* in: Conference on optimization and optimal control, BULIRSCH, R., W. OETTLI & J. STOER (eds), Oberwolfach. Springer, Berlin.

DEUFLHARD P. & G. HEINDL, [1979], *Affine invariant convergence theorems for Newton's method and extensions to related methods,* SIAM J. Numer. Anal. 16, 1-10.

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

GAY, D.M., [1975], *Implementing Brown's method,* Univ. of Texas, Austin, Center for Numer. Anal., Rep. CNA-109.

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

GOLUB, G.H. & W. KAHAN, [1965], *Calculating the singular values and pseudo-inverse of a matrix,* SIAM J. Numer. Anal. 2, 205-224.

GOLUB, G.H. & C. REINSCH, [1971], *Singular value decomposition and least squares solutions,* in: Handbook of automatic Computation, vol. 2, Linear Algebra, WILKINSON J.M. & C. REINSCH (eds), Springer, Heidelberg.

GRIEND, J.A. van de, [1978], *Optimization of functions of one variable* (dutch), Dissertation, Univ. of Leiden.

HEMKER, P.W. et al, [1979], *NUMAL, A library of numerical procedures in ALGOL* 60, Third revision, Mathematisch Centrum, Amsterdam.

HEMKER, P.W. & D.T. WINTER, [1979], *A preliminary report on numerical operators in ALGOL* 68, NW 66/79, Mathematisch Centrum, Amsterdam.

KANTOROVICH, L.W. & G.P. AKILOW, [1964], *Functional analysis in normed spaces* (german), publ. by P.H. Müller, transl. from Russ. by H. Langer and R. Kühne, Berlin, Akademie-Verlag, Math. Lehrbücher und Monographien: 2.17.

MEULEN, S.G. van der & M. VELDHORST, [1978], *TORRIX, A programming system for operations on vectors and matrices over arbitrary fields and of variable size,* Vol. I, Tract 86, Mathematisch Centrum, Amsterdam.

MIEL, G.J., [1977], *Exit criteria for Newton-type iterations,* Univ. of Calgary, Dep. o. Math. & Stat., Res. paper 363.

MORÉ, J.J. & M.Y. COSNARD, [1979], *Numerical solution of nonlinear Equations,* ACM Trans. Math. Software 5, 64-85.

ORTEGA, J.M. & W.C. RHEINBOLDT, [1970], *Iterative solution of nonlinear equations in several variables*, Academic Press, New York & London.

PENROSE, R., [1955], *A generalized inverse for matrices*, Proc. Cambridge Philos. Soc., 51, 406-413.

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

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

RHEINBOLDT, W.C., [1969], *Local mapping relations and global implicit function theorems*, Trans. Amer. Math. Soc. 138, 183-198.

RHEINBOLDT, W.C., [1974], *Methods for solving systems of nonlinear equations*, Region. Conf. Series in Appl. Math. 14, SIAM, Philadelphia.

ROSENBROCK, H.H., [1960], *An automatic method for finding the greatest or least value of a function*, Comput. J. 3, 175-184.

SLUIS, A. van der, [1969], *Condition numbers and equilibration of matrices*, Numer. Math. 14, 14-23.

WILKINSON, J.H., [1965], *The algebraic eigenvalue problem*, Clarendon Press, Oxford.

WIJNGAARDEN, A. van, et al. (eds), [1976], *Revised report on the algorithmic language ALGOL* 68, tract 50, Mathematisch Centrum, Amsterdam.

SAMENVATTING

Het oplossen van een stelsel niet-lineaire algebraïsche vergelijkingen is een veel voorkomend wiskundig probleem. Bijvoorbeeld vereist de oplossing van twee-punts randwaardeproblemen, elliptische randwaardeproblemen, integraalvergelijkingen of optimale besturingsproblemen dikwijls de oplossing van een stelsel niet-lineaire vergelijkingen. Een dergelijk stelsel vergelijkingen kan men noteren als $F(x) = 0$, voor een zekere functie $F : D \rightarrow \mathbb{R}^n$, waarbij D een niet-lege open verzameling is in $\mathbb{R}^n$. In dit proefschrift beperken we ons tot functies F waarvan de afgeleide (jacobiaan) bestaat en Lipschitz-continu is op D. Deze problemen worden geanalyseerd en numerieke oplossingsmethoden worden ontwikkeld en getest op hun bruikbaarheid.

In hoofdstuk 1 worden daartoe een aantal basisbegrippen en resultaten uit de analyse en numerieke algebra geformuleerd.

Hoofdstuk 2 is gewijd aan een stelling betreffende existentie en ééduidigheid van een oplossing van een stelsel niet-lineaire vergelijkingen.

In hoofdstuk 3 worden enige methoden voor benadering van de jacobiaan van het stelsel beschreven en geanalyseerd, ter voorbereiding op de definitie in hoofdstuk 4, van een klasse van iteratieve methoden voor het oplossen van stelsels niet-lineaire vergelijkingen. Deze klasse van zogenaamde Newton-achtige methoden is, zoals de naam zegt, afgeleid van de methode van Newton (of Newton-Raphson) door daarin de jacobiaan te vervangen door een geschikt gekozen benadering.

In hoofdstuk 5 wordt een consistente theorie ontwikkeld betreffende globale en (semi-) lokale convergentie van de geïntroduceerde klasse van methoden. Gebaseerd op de theoretische resultaten van hoofdstuk 5, wordt in hoofdstuk 6 een praktische uitwerking van de Newton-achtige methoden behandeld. Hierbij leidt gebruik van een gegeven formule voor approximatie van de Lipschitz-constante tot enige waardevolle modificaties van bekende Newton-achtige methoden. De algoritmen worden in dit hoofdstuk op modulaire wijze gedefinieerd. Hierbij wordt gebruik gemaakt van de programmeertaal ALGOL 68, teneinde een formele en niet-ambigue definitie mogelijk te maken.

In hoofdstuk 7 wordt een schema ontwikkeld voor experimentele vergelijking van de beschreven algoritmen. Tevens worden in dit hoofdstuk de experimenten beschreven en de experimentele resultaten geëvalueerd. Deze evaluatie leidt tot de introductie van twee poly-algoritmen (combinaties van Newton-achtige algoritmen); één voor problemen waarvan de analytische afgeleiden gegeven zijn en één voor problemen waarvoor dit niet het geval is.

In hoofdstuk 8 worden de experimentele resultaten gegeven van deze poly-algoritmen, alsmede van twee andere algoritmen die voor de beschouwde klasse van problemen zinvol kunnen zijn. Het blijkt dat de geïntroduceerde poly-algoritmen tot robuuste, betrouwbare en redelijk efficiënte programmatuur leiden, die geschikt is voor opname in een programmatheek.

APPENDIX I

TEST PROBLEMS

In this appendix we describe fourteen test problems of variable order and two of fixed order. Some of these problems depend on (a) parameter(s) which influence(s) the values of the problem indicators. As far as possible we describe the peculiar properties which make those problems worthwhile as test problems. Furthermore, we give numerical approximations to the problem indicators (see remark 7.5; we used $h = 10^{-3}$ for the computation of $\bar{\omega}$). These values are computed with a CYBER 73 ($\varepsilon = 2^{-47}$). As these values only give an indication of certain properties of the problem, we only give one significant digit. If the jacobian (at x_0 or x^*) appears to be numerically singular (triangular decomposition can not be performed, see subsection 6.2.1) then an S is given in the tables and no values for $\bar{\omega}$ and $\bar{\beta}$. For all problems analytic expressions for the jacobian elements are given, so that all problems can be used for Newton methods as well. The values for ε_{rf}, ε_{rj}, ε_{af} and ε_{aj} are no sharp upper bounds on the errors in $fl_\varepsilon(F(x))$ and $fl_\varepsilon(J(x))$ (see (6.3) and (6.4)). These values are reasonable bounds to be used in the test runs. The precision asked for is chosen to be standard

$$\delta_f = \delta_{rx} = \delta_{ax} = 10^{-7}.$$

For some problems we make an exception to this rule. Such exceptions are stated explicitly in the descriptions. We refer to a problem which is described in section I.i (i=1,2,...,16) as problem i.

I.1. (BROWN [1969])

function:

$$F_1(x) = -1 + \prod_{k=1}^{n} \xi_k,$$
$$F_i(x) = -(n+1) + \xi_i + \sum_{k=1}^{n} \xi_k, \quad i = 2,\dots,n.$$

jacobian:

$$(J(x))_{ij} = \begin{cases} \prod\limits_{\substack{k=1\\k\neq j}}^{n} \xi_k, & i = 1,\ j = 1,\ldots n, \\ 1, & i \neq j,\ i = 2,\ldots,n,\ j = 1,\ldots,n, \\ 2, & i = j,\ i \neq 1. \end{cases}$$

initial guess: $\xi_i = 0.5, \quad i = 1,\ldots,n.$

solutions:

a. $\xi_i^* = 1, \quad i = 1,\ldots,n,$

b. $\xi_1^* = 1/p^{n-1},\ \xi_i^* = p\ (i=1,\ldots,n),$
   for p a real zero of $np^n - (n+1)p^{n-1} + 1 = 0,\ p \neq 1.$

precisions: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = n\varepsilon.$

particular properties: The condition number of the jacobian matrix at the initial guess increases with increasing order due to the values of the jacobian elements in the first row which are all equal to $2^{-(n-1)}$. Scaling improves the condition number but using the same scaling throughout may increase the condition number at the solutions. (See table I.1).

<u>table I.1.</u>

	starting guess						solution a			solution b			
n	$\bar{\kappa}_0$	$\bar{\omega}_0$	$\bar{\beta}_0$	$\bar{\kappa}_0^s$	$\Vert D_1\Vert$	$\Vert D_2\Vert$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}_*^s$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}_*^s$	p
2	10	$_{10}-11$	2	10	1	1	7	2	7	9	2	9	0.5
3	40	5	7	40	1	1	10	1	10	5	0.9	5	-0.43426
4	$2_{10}2$	8	20	$2_{10}2$	1	1	20	0.5	20	30	3	30	0.86888
5	$5_{10}2$	10	70	$1_{10}2$	2^2	1	30	4	30	6	1	20	-0.57904
6	$1_{10}3$	10	$2_{10}2$	$2_{10}2$	2^3	1	40	10	50	60	5	60	0.94215
7	$4_{10}3$	10	$4_{10}2$	$2_{10}2$	2^4	1	60	20	$1_{10}2$	8	$7_{10}-3$	60	-0.65564
8	$9_{10}3$	20	$1_{10}3$	$3_{10}2$	2^5	1	80	30	$3_{10}2$	90	6	$3_{10}2$	0.96769
9	$2_{10}4$	20	$2_{10}3$	$4_{10}2$	2^6	1	$1_{10}2$	5	$6_{10}2$	10	$3_{10}-3$	$3_{10}2$	-0.70521
10	$6_{10}4$	20	$5_{10}3$	$4_{10}2$	2^7	1	$1_{10}2$	10	$1_{10}3$	$1_{10}2$	8	$1_{10}3$	0.97943
13	$7_{10}5$	30	$6_{10}4$	$7_{10}2$	2^{10}	1	$2_{10}2$	6	$1_{10}4$	$3_{10}2$	$1_{10}-4$	40	-0.76739
24	$1_{10}11$	50	$5_{10}9$	$5_{10}3$	2^{20}	1	$6_{10}2$	20	$3_{10}7$	$7_{10}2$	20	$3_{10}7$	0.99648
35	S	-	-	$_{10}4$	2^{31}	1	$1_{10}3$	8	$8_{10}10$	40	$2_{10}-4$	$1_{10}10$	-0.88370
46	S	-	-	$2_{10}4$	2^{42}	1	$2_{10}3$	25	S	$2_{10}3$	40	S	0.99905

For small order, it can be shown that the jacobian is singular at a point in $S_F(x_0,I)$. It is expected that this also holds for larger order. That means that the conditions for convergence of Newton-like methods (condition 5.3) are not satisfied for this starting guess.

In restrained methods, restraining occurs especially at the first step.

The problem has n-1 linear function components.

I.2. (Generalization of a function given by POWELL [1970])

function:

$$F_1(x) = c \prod_{k=1}^{n} \xi_k - 1,$$

$$F_i(x) = \exp(-\xi_{i-1}) + \exp(-\xi_i) - (1+\frac{1}{c}), \quad i = 2,\dots,n,$$

for some real parameter $c > 0$.

jacobian:

$$(J(x))_{ij} = \begin{cases} c \prod_{\substack{k=1 \\ k\neq j}}^{n} \xi_k, & i = 1,\ j = 1,\dots,n \\ 0, & j > i,\ j < i-1,\ i = 2,\dots,n,\ j = 1,\dots,n, \\ -\exp(-\xi_j), & j = i,\ j = i-1,\ i = 2,\dots,n. \end{cases}$$

initial guess:

$\xi_i = 1$, for $2 \le i \le n$, i even,

$\xi_i = c^{-2/n}$, for $1 \le i \le n$, i odd.

solution: e.g. for $n = 2$, $c = {}_{10}4$: $x^* = (1.098_{10}{-5}, 9.106)^T$.

precision: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = n\varepsilon$.

particular properties: The scaling of the variables can be controlled by the parameter c. For large c, the condition number at the starting point as well as at the solution can be improved by choosing a fixed scaling matrix based on the jacobian at x_0. However, the condition number at the solution will still be large. The problem indicators are given in table I.2. For $n \ge 24$ no solution is known.

table I.2.

		starting guess						solution		
n	c	$\bar{\kappa}_0$	$\bar{\omega}_0$	$\bar{\beta}_0$	$\bar{\kappa}^s_0$	$\Vert D_1\Vert$	$\Vert D_2\Vert$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}^s_*$
2	10	40	0.7	0.6	8	2^4	1	$2_{10}2$	2	20
3	10	8	6	0.2	8	1	1	10	3	10
3	10^2	10	2	0.7	4	2^3	1	$3_{10}2$	1	40
3	10^3	30	2	0.9	4	2^4	1	$4_{10}3$	1	$3_{10}2$
3	10^4	60	2	0.9	4	2^5	1	$6_{10}4$	1	$2_{10}3$
3	10^5	$1_{10}2$	2	1	6	2^7	1	$7_{10}5$	1	$6_{10}3$
3	10^6	$3_{10}2$	2	1	5	2^8	1	$9_{10}6$	1	$3_{10}4$
3	10^7	$6_{10}2$	2	1	5	2^9	1	$1_{10}8$	1	$2_{10}5$
3	10^8	$1_{10}3$	1	1	5	2^{10}	1	$1_{10}9$	1	$1_{10}6$
3	10^9	$3_{10}3$	1	1	5	2^{11}	1	$1_{10}10$	1	$6_{10}6$
3	10^{10}	$6_{10}3$	1	1	5	2^{12}	1	$1_{10}11$	1	$3_{10}7$
3	10^{11}	$1_{10}4$	1	1	4	2^{13}	1	$1_{10}12$	2	$2_{10}8$
13	10	50	0.4	1	50	1	1	$2_{10}4$	0.5	$2_{10}4$
24	10	$1_{10}3$	2	$2_{10}2$	$1_{10}3$	1	1	-	-	-
35	10	$3_{10}2$	0.3	70	$3_{10}2$	1	1	-	-	-
46	10	$5_{10}3$	3	$1_{10}3$	$5_{10}3$	1	1	-	-	-

I.3. (Generalization of a function given by POWELL [1970])

function:

$$F_i(x) = \prod_{k=1}^{i} \xi_k - 1, \qquad i = 1,\dots,n.$$

jacobian:

$$(J(x))_{ij} = \begin{cases} 0, & i = 1,\dots,n,\ j > i, \\ \prod\limits_{\substack{k=1 \\ k\neq j}}^{i} \xi_k, & i = 1,\dots,n,\ j \le i. \end{cases}$$

initial guess:

$$\xi_i = \begin{array}{ll} -1, & i = 1,\dots,n,\ i \text{ odd}, \\ 2, & i = 2,\dots,n,\ i \text{ even}. \end{array}$$

solution: $\xi_i^* = 1, \quad i = 1,\dots,n.$

precision: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = n\varepsilon$.

particular properties: $F_1(x)$ is linear and the jacobian matrix is lower triangular. The values of $\bar{\omega}_0$ and $\bar{\omega}_*$ are reasonably small for all orders; the same holds for $\bar{\kappa}_*$. However, κ_0 increases rapidly for increasing order. Scaling improves this number but spoils the value of $\bar{\kappa}_*$, if the same scaling matrices are used at x_0 and x^*. Note that $\bar{\beta}_0$ remains small while κ_0 increases, which shows that the bad condition of $J(x_0)$ for large n is not reflected in $\|(J(x_0))^{-1}F(x_0)\|$.

<u>table I.3.</u>

	starting guess						solution		
n	$\bar{\kappa}_0$	$\bar{\omega}_0$	$\bar{\beta}_0$	$\bar{\kappa}_0^s$	$\|D_1\|$	$\|D_2\|$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}_*^s$
2	6	0.8	2	6	1	1	3	0.3	3
13	$7_{10}2$	0.8	3	90	2^8	1	20	2	$1_{10}3$
24	$5_{10}4$	0.8	3	$1_{10}2$	2^{14}	1	30	2	$7_{10}4$
35	$2_{10}6$	0.8	3	$3_{10}2$	2^{20}	1	50	2	$6_{10}6$
46	$1_{10}8$	0.8	3	$3_{10}2$	2^{26}	2^4	60	2	$1_{10}8$

<u>I.4. (Generalized function of ROSENBROCK [1960])</u>

function:

$$F_1(x) = -4c(\xi_2 - \xi_1^2)\xi_1 - 2(1-\xi_1),$$

$$F_i(x) = 2c(\xi_i - \xi_{i-1}) - 4c(\xi_{i+1} - \xi_i^2)\xi_i - 2(1-\xi_i), \qquad i = 2,\dots,n-1,$$

$$F_n(x) = 2c(\xi_n - \xi_{n-1}^2).$$

jacobian:

$$(J(x))_{nn} = 2c, \qquad (J(x))_{11} = 12c\xi_1^2 - 4c\xi_2 + 2,$$

$$(J(x))_{ij} = \begin{cases} 12c\xi_i^2 - 4c\xi_{i+1} + 2 + 2c, & i = j,\ i = 2,\dots,n-1, \\ -4c\xi_j, & i = 2,\dots,n \text{ and } j = i-1, \\ -4c\xi_i, & i = 1,\dots,n-1 \text{ and } j = i+1, \\ 0, & i,j = 1,\dots,n,\ j > i+1,\ j < i-1. \end{cases}$$

initial guess:

$\xi_i = -1.2, \quad i = 1,\dots,n,\ i \text{ odd},$
$\quad\ 1, \quad\ \ i = 2,\dots,n,\ i \text{ even}.$

solution: $\xi_i^* = 1, \quad i = 1,\dots,n.$

precision: $\varepsilon_{rf} = \varepsilon_{rj} = 10\varepsilon,\ \varepsilon_{af} = \varepsilon_{aj} = 10c\varepsilon.$

Note that the error in F(x) may be large at the solution due to cancellation of significant digits and a large value of c. Cancellation of digits also occurs in some terms of the jacobian; we assumed that in the domain of interest $\|x\|$ is not much greater than 1.

<u>table I.4.</u>

		starting guess						solution		
n	c	$\bar{\kappa}_0$	$\bar{\omega}_0$	$\bar{\beta}_0$	$\bar{\kappa}_0$	$\|D_1\|$	$\|D_2\|$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}_*$
2	10	60	20	0.2	40	2^2	1	30	$6_{10}{-11}$	30
2	10^2	60	0.9	0.4	30	2	2	$3_{10}3$	$1_{10}3$	$2_{10}3$
2	10^3	60	$7_{10}{-2}$	0.4	40	2	1	$3_{10}4$	$5_{10}3$	$2_{10}4$
2	10^4	60	$7_{10}{-3}$	0.4	40	2^2	1	$3_{10}5$	$4_{10}4$	$3_{10}5$
2	10^5	60	$7_{10}{-4}$	0.4	40	2^2	1	$3_{10}6$	$4_{10}5$	$3_{10}6$
2	10^6	60	$7_{10}{-5}$	0.4	40	2	1	$3_{10}7$	$4_{10}6$	$2_{10}7$
2	10^7	60	$7_{10}{-6}$	0.4	30	2^2	2	$2_{10}8$	$4_{10}7$	$2_{10}8$
13	10	30	0.2	3	10	2^2	1	10	4	20
13	10^2	30	0.2	3	10	2^2	1	$4_{10}3$	$3_{10}2$	$2_{10}3$
13	10^3	30	0.2	3	10	2^2	1	$4_{10}4$	$5_{10}2$	$2_{10}4$
13	10^4	30	0.2	3	10	2^2	1	$4_{10}5$	$8_{10}3$	$2_{10}5$
13	10^5	30	0.2	3	10	2^2	1	$4_{10}6$	$8_{10}4$	$2_{10}6$
13	10^6	30	0.2	3	10	2^2	1	$4_{10}7$	$8_{10}5$	$3_{10}7$
13	10^7	30	0.2	3	10	2^2	1	$4_{10}8$	$8_{10}6$	$2_{10}8$
24	10	70	0.3	3	30	2^2	1	10	4	20
24	10^4	70	0.2	3	30	2^2	1	$2_{10}4$	$1_{10}4$	$3_{10}4$
24	10^7	70	0.2	3	30	2^2	1	$4_{10}8$	$4_{10}6$	$2_{10}8$
35	10	30	0.2	3	10	2^2	1	10	4	20
35	10^4	30	0.2	3	10	2^2	1	$2_{10}4$	$1_{10}4$	$3_{10}4$
35	10^7	30	0.2	3	10	2^2	1	$4_{10}8$	$6_{10}6$	$2_{10}8$
46	10	70	0.2	4	30	2^2	1	10	4	20
46	10^4	70	0.2	3	30	2^2	1	$2_{10}4$	$6_{10}3$	$3_{10}4$
46	10^7	70	0.2	3	30	2^2	1	$4_{10}8$	$4_{10}6$	$2_{10}8$

particular properties: For fixed order $n = 2$ or 13 and increasing c we have increasing values for $\bar{\kappa}_*$ and $\bar{\omega}_*$. At the starting point most indicator values remain almost independent of c, except for $\bar{\omega}_0$ which decreases with increasing c. Scaling yields no significant improvement of the condition number. $\|D_1\|$ and $\|D_2\|$ are less than or equal to 4, although for large c all diagonal elements of D_1 are very small (e.g. for $c = 10^7$, the smallest diagonal element of D_1 is 2^{-28}).

I.5. (GHERI & MANCINO [1971])

function:

$$F_i(x) = 14n\xi_i + (i-\tfrac{n}{2})^3 + \sum_{\substack{k=1\\k\neq i}}^{n} z_{ik}(\sin^5(\ell n(z_{ik})) + \cos^5(\ell n(z_{ik}))),\ i = 1,\dots,n,$$

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

jacobian:

$$(J(x))_{ij} = \begin{cases} 14n, & i = j,\ i = 1,\dots,n, \\ \dfrac{\xi_j}{z_{ij}}\left[\sin^5(\ell_{ij})+\cos^5(\ell_{ij})+5\sin^4(\ell_{ij})\cos(\ell_{ij})-5\cos^4(\ell_{ij})\sin(\ell_{ij})\right] \\ \text{with } \ell_{ij} = \ell n(z_{ij}),\ i,j = 1,\dots,n,\ i \neq j. \end{cases}$$

initial guess:

$$x_0 = -F(0)\left(\frac{C_1+C_2}{2C_1C_2}\right)$$

where $C_1 = 20n-6$, $C_2 = 8n+6$.

solution: depends on n. In general many solutions exist.

precision: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = 100n\varepsilon$.

(These values are only rough estimates of the upperbound on the errors. In fact they should depend on the precision of the special functions ℓn, sin and cos.)

particular properties: As appears from table I.5 the values of the problem indicators are all small, almost independent of the order. One can say that this problem is easily solvable for any reasonable order. One can obtain theoretical bounds on some of the values (see BUS [1977, section 4]):

$$\kappa(x) \le \left(\frac{100n^2-60n+9}{-2n^2+69n-12}\right)^{\frac{1}{2}}$$

$$\omega(x) \le 36\sqrt{n-1}\,[-2n^2+69n-12]^{-\frac{1}{2}},$$

which are bounded for $n \le 34$.

table I.5.

n	starting point						solution [1)]		
	$\bar{\kappa}_0$	$\bar{\omega}_0$	$\bar{\beta}_0$	$\bar{\kappa}_0$	$\|D_1\|$	$\|D_2\|$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}_*$
2	1	0.1	$4_{10}-3$	1	1	1	1	0.1	1
13	1	$1_{10}-2$	0.4	1	1	1	1	$2_{10}-2$	1
24	1	$5_{10}-3$	2	1	1	1	1	$2_{10}-3$	1
35	1	$3_{10}-3$	5	1	1	1	1	$2_{10}-3$	1
46	1	$3_{10}-4$	10	1	1	1	1	$2_{10}-3$	1

1) We have chosen an approximation to one possible solution obtained by one of our algorithms; for other solutions we expect to obtain almost the same results.

For these problems the tolerances are chosen to be

$$\delta_f = {}_{10}{-6}; \qquad \delta_{rx} = \delta_{ax} = {}_{10}{-6}.$$

Problem 5 will be used also for testing the effect of large errors in the function and jacobian. The following problem will be referred to as problem 5a.

function: Let F be problem 5. Then

$$\tilde{F}_i(x) = F_i(x)(1+p_i(x)) + q_i(x), \quad i = 1,\dots,n,$$

where $p_i(x)$ and $q_i(x)$ are randomly chosen in the intervals $[-p,p]$ and $[-q,q]$, respectively , for all $i \le n$ and x and with p and q real parameters $0 \le p,\ q \le 1$.

jacobian: Let J be the jacobian of problem 5. Then

$$(\tilde{J}(x))_{ij} = (J(x))_{ij}(1+p_{ij}(x)) + q_{ij}(x), \quad i = 1,\dots,n,$$

where again $p_{ij}(x)$ and $q_{ij}(x)$ are randomly chosen in $[-p,p]$ and $[-q,q]$ for all x, i and j $(i,j \le n)$.

precisions: $\varepsilon_{rf} = \varepsilon_{rj} = 100n\varepsilon + p$, $\varepsilon_{af} = \varepsilon_{aj} = 100n\varepsilon + q$.

The values for p and q depend on the experiment and on the machine precision.

tolerances: We have to choose at least $\delta_f \ge \varepsilon_{af}$. Specific choices depend on the testing objectives.

1.6. (BROYDEN [1971])

function:

$$F_i(x) = (1+100\xi_i^2)\xi_i + 1 - 100 \sum_{k\in I_i} (\xi_k+\xi_k^2),$$

where $I_i = \{k \mid k\neq i,\ \max(1,i-2) \le k \le \min(n,i+2)\}$.

jacobian:

$$(J(x))_{ij} = \begin{cases} 1 + 300\xi_i^2, & i = 1,\dots,n,\ i = j, \\ 0, & i,j = 1,\dots,n,\ i \neq j,\ j < i-2,\ j > i+2, \\ -100(1+2\xi_j), & i,j = 1,\dots n,\ \ i \neq j,\ i-2 \le j \le i+2. \end{cases}$$

initial guess: $\xi_i = -1, \quad i = 1,\dots,n.$

solution: depends on n.

precisions: $\varepsilon_{rf} = \varepsilon_{rj} = \varepsilon_{aj} = 5\varepsilon$, $\varepsilon_{af} = 100\varepsilon$.

particular properties: The jacobian of this problem is a quinta-diagonal matrix. Based on the values given in table I.6 we may expect this problem to be easily solvable by Newton-like programs.

table I.6.

n	initial guess						solution		
	$\bar{\kappa}_0$	$\bar{\omega}_0$	$\bar{\beta}_0$	$\bar{\kappa}^s_0$	$\|D_1\|$	$\|D_2\|$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}^s_*$
2	1	1	0.4	2	1	1	2	3	2
13	7	0.6	0.6	7	1	1	7	0.9	7
24	9	0.5	0.8	9	1	1	7	0.8	7
35	9	0.4	0.9	9	1	1	8	0.6	8
46	9	0.3	1	9	1	1	8	0.7	8

1.7. (BROYDEN [1971])

function:

$F_1(x) = (3-c\xi_1)\xi_1 + 1 - 2\xi_2,$

$F_i(x) = (3-c\xi_i)\xi_i + 1 - \xi_{i-1} - 2\xi_{i+1}, \quad i = 2,\dots,n-1,$

$F_n(x) = (3-c\xi_n)\xi_n + 1 - \xi_{n-1},$

for c a real parameter, $c > 0$.

jacobian:

$$(J(x))_{ij} = \begin{cases} 3 - 2c\xi_i, & i = j,\ i = 1,\dots,n, \\ -1, & j = i-1,\ i = 2,\dots,n, \\ -2, & j = i+1,\ i = 1,\dots,n-1, \\ 0, & j \neq i,i-1,i+1,\ i,j = 1,\dots,n. \end{cases}$$

initial guess: $\xi_i = -1,\ i = 1,\dots,n.$

solution: depends on n and c. E.g. for $n = 2$ we have in general four solutions, see particular properties.

precision: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = 5\varepsilon.$

particular properties: For small values of c, these problems are expected to be easily solvable as the approximate values of the problem indicators are small (see table I.7).

table I.7.

		starting guess						solution [*)]		
n	c	$\bar{\kappa}_0$	$\bar{\omega}_0$	$\bar{\beta}_0$	$\bar{\kappa}^s_0$	$\|D_1\|$	$\|D_2\|$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}^s_*$
2	10	1	0.7	0.7	1	1	1	2	5	2
2	10^4	1	0.7	0.7	1	1	1	1	10^2	1
13	10	1	0.3	2	1	1	1	2	1	2
13	10^4	1	0.3	2	1	1	1	1	50	1
24	10	1	0.2	2	1	1	1	2	0.8	2
24	10^4	1	0.2	2	1	1	1	1	40	1
35	10	1	0.2	3	1	1	1	2	0.6	2
35	10^4	1	0.2	3	1	1	1	1	40	1
46	10	1	0.1	3	1	1	1	2	0.6	2
46	10^4	1	0.1	3	1	1	1	1	40	1

*) The values at the solution are given for one particular solution obtained by one of our algorithms. Other solutions exist.

For n = 2 the solutions of this problem are the intersective points of the parabolas:

$$\xi_2 = \tfrac{1}{2}(-c\xi_1^2+3\xi_1+1) \text{ and } \xi_1 = -c\xi_2^2 + 3\xi_2 + 1.$$

The four solutions (if c is large enough (e.g. $\geq$ 10) we have four intersective points) come closer to each other if c increases. In the degenerate case that c = ∞ we have one solution at the origin. In table I.7a we give for various values of c the values of $\bar{\omega}^*$ (which appear to be of the same magnitude for the various solutions) and the maximum of the norms of the solutions.

table I.7a.

c	0.1	1	5	10	50	10^2	10^3	10^4	10^5	10^8
$\bar{\omega}_*$	$9\,10^{-2}$	0.7	2	5	7	10	30	10^2	$3\,10^2$	$7\,10^3$
$\max\|x^*\|$	20	3	0.9	0.5	0.2	0.1	$3\,10^{-2}$	$1\,10^{-2}$	$3\,10^{-3}$	10^{-4}

I.8. (MORÉ & COSNARD [1979])

function:

$F_1(x) = 2\xi_1 - \xi_2 + \frac{1}{2}h^2(\xi_1+t_1+1)^3,$

$F_i(x) = 2\xi_i - \xi_{i+1} - \xi_{i-1} + \frac{1}{2}h^2(\xi_i+t_i+1)^3, \quad i = 2,\dots,n-1,$

$F_n(x) = 2\xi_n - \xi_{n-1} + \frac{1}{2}h^2(\xi_n+t_n+1)^3,$

where $t_i = hi$ $(i=1,\dots,n)$ and $h = 1/(n+1)$.

jacobian:

$$(J(x))_{ij} = \begin{cases} 2 + \frac{3}{2}h^2(\xi_i+t_i+1)^2, & i = j,\ i = 1,\dots,n, \\ -1, & j = i+1, i-1,\ i = 1,\dots,n, \\ 0, & i,j = 1,\dots,n,\ j < i-1,\ j > i. \end{cases}$$

initial guess: $\xi_i = 0.5, \quad i = 1,2,\dots,n.$

solution: depends on n.

precisions: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = 5\varepsilon.$

particular properties: Using Gerschgorin's theorem we can bound the condition number of the jacobian by

$$\kappa(x) \le \max_{1\le i\le n} \left(1 + \frac{4}{3h^2(\xi_i+t_i+1)^2}\right).$$

Note that the right hand side increases for increasing n (decreasing h) and becomes infinite if $\xi_i = -(1 + \frac{i}{n+1})$. For the starting point we obtain

$$\kappa_0 \le 1 + \frac{4}{3h^2(1.5+t_1)^2}$$

which yields for n = 2, 13, 24, 35 and 46, respectively 4.6, 110, 330, 740 and 1300 approximately. In table I.8 we give the approximations to the values of the problem indicators, for the selected orders.

table I.8.

n	initial guess						solution		
	$\bar{\kappa}_0$	$\bar{\omega}_0$	$\bar{\beta}_0$	$\bar{\kappa}^s_0$	$\lVert D_1\rVert$	$\lVert D_2\rVert$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}^s_*$
2	2	0.3	0.8	5	2^2	1	3	0.2	6
13	50	0.1	2	50	1	1	60	0.1	60
24	200	$7_{10}-2$	3	200	1	1	200	$8_{10}-2$	200
35	300	$6_{10}-2$	3	300	1	1	400	$7_{10}-2$	400
46	600	$5_{10}-2$	4	600	1	1	700	$6_{10}-2$	700

I.9. (MORÉ & COSNARD [1979])

function:

$$F_i(x) = \xi_i + \tfrac{1}{2}h\Big((1-t_i)\sum_{k=1}^{i} t_k(\xi_k+t_k+1)^3 + t_i\sum_{k=i+1}^{n}(1-t_k)(\xi_k+t_k+1)^3\Big),$$

$$i = 1,\dots,n,$$

where $t_i = ih(i=1,\dots,n)$ and $h = 1/(n+1)$.

jacobian:

$$(J(x))_{ij} = \begin{cases} 1 + \frac{3}{2}ht_i(1-t_i)(\xi_i+t_i+1)^2, & i = j,\ i = 1,\dots,n, \\ \frac{3}{2}ht_i(1-t_j)(\xi_j+t_j+1)^2, & i,j = 1,\dots,n,\ j > i, \\ \frac{3}{2}ht_j(1-t_i)(\xi_j+t_j+1)^2, & i,j = 1,\dots,n,\ j < i. \end{cases}$$

table I.9.

n	initial guess						solution		
	$\bar{\kappa}_0$	$\bar{\omega}_0$	$\bar{\beta}_0$	$\bar{\kappa}^s_0$	$\lVert D_1\rVert$	$\lVert D_2\rVert$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}^s_*$
2	1	0.3	0.8	1	1	1	1	0.3	1
13	2	0.1	2	2	1	1	1	0.1	1
24	2	$7_{10}-2$	3	2	1	1	1	$7_{10}-2$	1
35	2	$6_{10}-2$	3	2	1	1	1	$7_{10}-2$	1
46	2	$5_{10}-2$	4	2	1	1	1	$6_{10}-2$	1

initial guess: $\xi_i = 0.5, \quad i = 1,\dots,n.$

solution: depends on n.

precision: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = 2\varepsilon n.$

particular properties: This problem is easily solvable (see table I.9).

I.10. (FLETCHER & POWELL [1963])

function:

$F(x) = Au(x) + Bv(x) - e,$

where A and B are n×n matrices with integer elements, independent of x and chosen randomly in $[-m,+m]$, $u(x)$ and $v(x) \in \mathbb{R}^n$ for all $x \in D$ and $e \in \mathbb{R}^n$ is chosen such that for a given $x^* \in \mathbb{R}^n$, which is chosen randomly with elements in $[-b^*,b^*] \in \mathbb{R}$:

$$e = Au(x^*) + Bv(x^*).$$

We choose

$$\begin{aligned} &m = 100 \\ &b^* = \pi, \\ &u(x) = (\sin(\xi_1), \dots, \sin(\xi_n))^T, \\ &v(x) = (\cos(\xi_1), \dots, \cos(\xi_n))^T. \end{aligned}$$

jacobian:

$J(x) = A \frac{d}{dx} u(x) + B \frac{d}{dx} v(x),$

with $\frac{d}{dx} u(x) = \mathrm{diag}(\cos(\xi_1), \dots, \cos(\xi_n))^T,$

$\quad\quad \frac{d}{dx} v(x) = \mathrm{diag}(-\sin(\xi_1), \dots, -\sin(\xi_n))^T.$

initial guess: $x_0 = x^* + p$, with $p \in \mathbb{R}^n$ chosen randomly with elements in $[-b_p,+b_p]$, we choose $b_p = 0.01\pi$.

solution: x^*, randomly chosen, see above.

precisions: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = 100n\varepsilon.$

particular properties: We can influence the row or column scaling of the jacobian by multiplying a certain row or column of A and B with some scaling factor. In table I.10 we give values of the problem indicators for several values of s_r and s_c and for the set of selected orders. If s_r or s_c is unequal 1 then the (n//2+1)-th row or column, respectively, of A and B are multiplied by s_r or s_c.

table I.10.

			initial guess						solution		
n	s_r	s_c	$\bar{\kappa}_0$	$\bar{\omega}_0$	$\bar{\beta}_0$	$\bar{\kappa}^s_0$	$\Vert D_1\Vert$	$\Vert D_2\Vert$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}^s_*$
2	1	1	3	0.7	$4_{10}{-2}$	3	1	1	3	1	3
2	10^{-3}	1	$2_{10}3$	0.7	$4_{10}{-2}$	3	2^{10}	1	$2_{10}3$	1	3
2	10^{-6}	1	$2_{10}6$	0.7	$4_{10}{-2}$	3	2^{20}	1	$2_{10}6$	0.5	3
2	10^{-9}	1	$2_{10}9$	0.7	$4_{10}{-2}$	3	2^{30}	1	$2_{10}9$	1	3
2	10^{-14}	1	S	-	-	3	2^{46}	1	S	-	3
2	1	10^{-3}	$1_{10}3$	5	0.5	3	2	2^{9}	$1_{10}3$	$6_{10}{-2}$	3
2	1	10^{-6}	$1_{10}6$	$6_{10}{-2}$	$5_{10}2$	3	2	2^{19}	$1_{10}6$	0.3	3
2	1	10^{-9}	$6_{10}9$	1	$2_{10}6$	3	2	2^{29}	$5_{10}9$	$1_{10}{-3}$	10
2	1	10^{-14}	S	-	-	4	2	2^{45}	S	-	4
13	1	1	20	0.9	$8_{10}{-2}$	20	1	1	20	1	20
13	10^{-3}	1	$6_{10}3$	0.9	$8_{10}{-2}$	30	2^{8}	1	$7_{10}3$	1	30
13	10^{-6}	1	$6_{10}6$	0.9	$8_{10}{-2}$	20	2^{20}	1	$7_{10}6$	1	20
13	10^{-9}	1	$6_{10}9$	0.9	$8_{10}{-2}$	20	2^{30}	1	$7_{10}9$	1	20
13	10^{-14}	1	S	-	-	20	2^{46}	1	S	-	20
13	1	10^{-3}	$4_{10}3$	5	0.3	20	1	2^{10}	$3_{10}3$	0.1	20
13	1	10^{-6}	$4_{10}6$	$6_{10}{-2}$	$3_{10}2$	20	1	2^{20}	$3_{10}6$	0.3	20
13	1	10^{-9}	$4_{10}9$	$6_{10}{-2}$	$3_{10}5$	20	1	2^{30}	$3_{10}9$	$7_{10}{-3}$	20
13	1	10^{-14}	S	-	-	20	1	2^{46}	S	-	20
24	1	1	30	1	0.1	30	2	1	20	0.8	30
35	1	1	70	2	0.1	70	1	1	60	1	60
46	1	1	$3_{10}2$	6	0.1	$3_{10}2$	1	1	$3_{10}2$	4	$3_{10}2$

Note that scaling of the function does not influence the values of $\bar{\omega}_0$ or $\bar{\omega}_*$. This is not true for scaling of the variables. This problem is, for n = 2, 13, particularly suitable for testing scaling strategies.

Scaling based on the jacobian at x_0 yields good scaling everywhere in the domain.

<u>I.11.</u>

function, jacobian, starting guess and solution: as problem 10 with

$$u(x) = (\exp(\xi_1), \ldots, \exp(\xi_n))^T,$$
$$v(x) = (\exp(-\xi_1), \ldots, \exp(-\xi_n))^T,$$
$$m = 100,\ b^* = 1,\ b_p = 0.1.$$

<u>table I.11.</u>

			initial guess								
n	s_r	s_c	$\bar\kappa_0$	$\bar\omega_0$	$\bar\beta_0$	$\bar\kappa_0^s$	$\lVert D_1\rVert$	$\lVert D_2\rVert$	$\bar\kappa_*$	$\bar\omega_*$	$\bar\kappa_*^s$
2	1	1	4	2	0.1	2	2	1	4	3	3
2	$_{10}-3$	1	$4_{10}3$	2	0.1	2	2^{11}	1	$4_{10}3$	3	3
2	$_{10}-6$	1	$4_{10}6$	2	0.1	2	2^{21}	1	$4_{10}6$	3	3
2	$_{10}-9$	1	$4_{10}9$	2	0.1	2	2^{31}	1	$4_{10}9$	2	2
2	$_{10}-14$	1	S	–	–	2	2^{48}	1	S	–	2
2	1	$_{10}-3$	$2_{10}3$	0.5	8	4	1	2^{9}	$2_{10}3$	0.6	5
2	1	$_{10}-6$	$2_{10}6$	0.3	$8_{10}3$	4	1	2^{19}	$2_{10}6$	0.6	5
2	1	$_{10}-9$	$1_{10}10$	7	$7_{10}7$	4	1	2^{29}	–	–	– *)
2	1	$_{10}-14$	S	–	–	4	1	2^{42}	S	–	5
13	1	1	60	0.5	0.3	60	2	1	40	0.8	40
13	$_{10}-3$	1	$3_{10}4$	0.5	0.3	80	2^{10}	1	$2_{10}4$	1	50
13	$_{10}-6$	1	$3_{10}7$	0.5	0.3	60	2^{20}	1	$2_{10}7$	1	40
13	$_{10}-9$	1	$3_{10}10$	0.5	0.3	60	2^{31}	1	$2_{10}10$	2	40
13	$_{10}-14$	1	S	–	–	70	2^{47}	1	S	–	40
13	1	$_{10}-3$	$7_{10}3$	2	2	60	2	2^{10}	$6_{10}3$	1	40
13	1	$_{10}-6$	$7_{10}6$	1	$2_{10}3$	60	2	2^{20}	$6_{10}6$	1	40
13	1	$_{10}-9$	$7_{10}9$	1	$2_{10}6$	60	2	2^{30}	–	–	– *)
13	1	$_{10}-14$	S	–	–	60	2	2^{47}	S	–	40
24	1	1	90	2	0.3	$1_{10}2$	2	1	70	2	80
35	1	1	70	0.6	0.3	70	1	1	60	0.4	60
46	1	1	$1_{10}2$	0.7	0.4	$_{10}2$	1	1	80	0.7	80

*) algorithm used did not find a solution

precisions: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = 100n\varepsilon$.

particular properties: As for problem 10 we can influence row or column scaling of the jacobian by multiplying a certain row or column of A and B by some scaling factors. In table I.11 we give the approximate values of the problem indicators.

I.12.

function, jacobian, starting guess and solution: As problem 10 with

$$u(x) = (\ell n(\xi_1+10),\ \ldots,\ \ell n(\xi_n+10))^T,$$
$$v(x) = (\ell n(10-\xi_1),\ \ldots,\ \ell n(10-\xi_n))^T,$$
$$m = 10,\ b^* = 1,\ b_p = 0.1.$$

precisions: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = 100n\varepsilon$.

particular properties: The function is undefined for all x with $\xi_j \geq 10$ or $\xi_j \leq -10$ for some j, $1 \leq j \leq n$. In table I.12 we give the values of the problem indicators for selected orders.

table I.12.

n	initial guess						solution		
	$\bar{\kappa}_0$	$\bar{\omega}_0$	$\bar{\beta}_0$	$\bar{\kappa}^s_0$	$\lVert D_1\rVert$	$\lVert D_2\rVert$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}^s_*$
2	4	0.2	0.1	4	1	1	3	0.2	3
13	10	9_{10}-2	0.3	10	1	1	10	8_{10}-2	10
24	40	8_{10}-2	0.3	40	1	1	40	0.3	40
35	50	0.1	0.3	50	1	1	50	0.2	50
46	60	0.3	0.4	60	1	1	60	0.3	60

I.13.

function:

$$F(x) = A(x)u(x) + B(x)v(x) - e,$$

where $(A(x))_{ij} = k^{(1)}_{ij}(a(x))_{ij}$, $(B(x))_{ij} = k^{(2)}_{ij}(b(x))_{ij}$, for $i,j = 1,\ldots,n$, with $k^{(\ell)}_{ij}$ $(i,j = 1,\ldots,n,\ \ell = 1,2)$ chosen randomly in $[-m,m]$, $(a(x))_{ij}$ and

$(b(x))_{ij}$ given functions $(i,j = 1,\dots,n)$ for $x \in D$ and $e \in \mathbb{R}^n$ chosen such that for given $x^* \in \mathbb{R}^n$

$$e = A(x^*)u(x^*) + B(x^*)v(x^*).$$

We choose

$$(a(x))_{ij} = \exp(\xi_i+\xi_j), \qquad i,j = 1,\dots,n,$$
$$(b(x))_{ij} = \exp(-(\xi_i+\xi_j)), \qquad i,j = 1,\dots,n,$$
$$u(x) = v(x) = (\xi_1,\dots,\xi_n)^T,$$
$$m = 10,$$

and x^* randomly with elements in $[-b^*,b^*] \in \mathbb{R}$ with $b^* = 1$.

jacobian:

$$J(x) = (\frac{d}{dx} A(x))u(x) + A(x)\frac{d}{dx} u(x) + (\frac{d}{dx} B(x))v(x) + B(x)\frac{d}{dx} v(x).$$

Here $\frac{d}{dx} u(x) = \frac{d}{dx} v(x) = I$.

initial guess: $x_0 = x^* + p$, with $p \in \mathbb{R}^n$ chosen randomly with elements in $[-b_p,+b_p]$. We choose $b_p = 0.1$.

solution: x^*, chosen randomly, see above.

presicions: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = 10^3\, n\varepsilon$.

particular properties: The approximate values of the problem indicators are given in table I.13.

table I.13.

n	initial guess						solution		
	$\bar\kappa_0$	$\bar\omega_0$	$\bar\beta_0$	$\bar\kappa_0^s$	$\|D_1\|$	$\|D_2\|$	$\bar\kappa_*$	$\bar\omega_*$	$\bar\kappa_*^s$
2	7	2	0.1	4	2	2	5	4	4
13	80	2	0.5	40	4	1	70	4	40
24	$3_{10}2$	2	0.7	$2_{10}2$	4	1	$2_{10}2$	4	$1_{10}2$
35	$9_{10}2$	20	0.9	$7_{10}2$	2	1	-	-	- *)
46	$2_{10}2$	2	0.5	$1_{10}2$	4	1	$3_{10}2$	3	$3_{10}2$

*) algorithm used did not find a solution

I.14.

function, jacobian, starting guess and solution: As problem 13 with

$$(a(x))_{ij} = \xi_i + \xi_j \quad i,j = 1,\dots,n,$$
$$(b(x))_{ij} = (\xi_i+\xi_j+10)^{-1} \quad i,j = 1,\dots,n,$$
$$u(x) = (\sin(\xi_1),\dots,\sin(\xi_n))^T,$$
$$v(x) = (\cos(\xi_1),\dots,\cos(\xi_n))^T,$$
$$m = 100, \quad b^* = \pi, \quad b_p = 0.01\pi.$$

precisions: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = {}_{10}3n\varepsilon$.

particular properties: The function is undefined for x with $\xi_i + \xi_j + 10 = 0$, for some i and j, $1 \le i,j \le n$.

table I.14.

n	initial guess						solution		
	$\bar\kappa_0$	$\bar\omega_0$	$\bar\beta_0$	$\bar\kappa_0^s$	$\|D_1\|$	$\|D_2\|$	$\bar\kappa_*$	$\bar\omega_*$	$\bar\kappa_*^s$
2	60	10	$5_{10}-2$	60	2	1	50	8	50
13	70	1	$8_{10}-2$	60	2^2	1	60	6	60
24	40	1	0.1	40	2^2	1	50	2	50
35	$3_{10}2$	6	0.1	$3_{10}2$	2^2	1	$6_{10}2$	20	$6_{10}2$
46	$2_{10}2$	2	0.1	$1_{10}2$	2	1	-	-	- *)

*) algorithm used did not find a solution

I.15. (POWELL [1962])

function:

$$F_1(x) = 2(\xi_1+10\xi_2) + 40(\xi_1-\xi_4)^3,$$
$$F_2(x) = 20(\xi_1+10\xi_2) + 4(\xi_2-2\xi_3)^3,$$
$$F_3(x) = 10(\xi_3-\xi_4) - 8(\xi_2-2\xi_3)^3,$$
$$F_4(x) = -10(\xi_3-\xi_4) - 40(\xi_1-\xi_4)^3.$$

jacobian:

$$J(x) = \begin{pmatrix} 2+120(\xi_1-\xi_4)^2 & 20 & 0 & -120(\xi_1-\xi_4)^2 \\ 20 & 200+12(\xi_2-2\xi_3)^2 & -24(\xi_2-2\xi_3)^2 & 0 \\ 0 & -24(\xi_2-2\xi_3) & 10+48(\xi_2-2\xi_3)^2 & -10 \\ -120(\xi_1-\xi_4)^2 & 0 & -10 & 10+120(\xi_1-\xi_4)^2 \end{pmatrix}$$

initial guess: $x_0 = (3, -1, 0, 1)^T$.

solution: $x^* = (0, 0, 0, 0)^T$.

precision: $\varepsilon_{rf} = \varepsilon_{af} = \varepsilon_{rj} = \varepsilon_{aj} = 10\varepsilon$.

particular properties: The jacobian matrix is singular (rank equals 2) at the solution. This makes the problem especially suitable for testing the robustness of the stopping criteria used, as convergence will become linear close to the solution, for Newton-like algorithms.
The values of the problem indicators are:

$\bar{\kappa}_0 = 2_{10}2$; $\bar{\omega}_0 = 0.2$; $\bar{\beta}_0 = 2$; $\bar{\kappa}_0^s = 3_{10}2$; $\|D_1\| = 2^4$; $\|D_2\| = 1$; $\bar{\kappa}_* = \bar{\kappa}_*^s = \infty$.

<u>I.16. (BREZINSKI [1975])</u>

function:

$F_1(x) = \xi_1 - c^3\xi_2^2$,

$F_2(x) = \xi_2 - 1/\xi_1$,

for some real parameter c, $0 < c < 0.5$.

jacobian:

$$J(x) = \begin{pmatrix} 1 & -2c^3\xi_2 \\ \xi_1^{-2} & 1 \end{pmatrix}$$

initial guess: $x_0 = (2/c, 2/c)^T$.

solution: $x^* = (c, 1/c)^T$.

precision: $\varepsilon_{rf} = \varepsilon_{rj} = 5\varepsilon$; $\varepsilon_{af} = \varepsilon_{aj} = 5c\varepsilon$.

particular properties: For small c the solution vector contains a very small and a very large element. I.e. the variables are badly scaled around the solution. This problem is particularly suitable for testing the influence of bad variable scaling on the performance of the algorithms. In table I.16 we give the values of the problem indicators for various c.

table I.16.

	initial guess						solution		
c	$\bar{\kappa}_0$	$\bar{\omega}_0$	$\bar{\beta}_0$	$\bar{\kappa}^s_0$	$\Vert D_1 \Vert$	$\Vert D_2 \Vert$	$\bar{\kappa}_*$	$\bar{\omega}_*$	$\bar{\kappa}^s_*$
1	9	0.6	2	3	2^3	2	2	1	7
10	20	8	0.2	1	2^4	1	$1_{10}4$	60	$8_{10}2$
10^2	20	80	$2_{10}-2$	1	2^4	1	$1_{10}8$	40	$3_{10}7$
10^3	20	$5_{10}2$	$2_{10}-3$	1	2^4	1	$4_{10}12$	0	$3_{10}11$
10^4	20	$1_{10}3$	$2_{10}-4$	1	2^4	1	S	-	S
10^5	20	$5_{10}3$	$2_{10}-5$	1	2^4	1	S	-	S

APPENDIX II

TEST RESULTS

In this appendix we give the experimental results obtained by running the ALGOL 60 programs for the various test problems. The ALGOL 60 programs use the NUMAL software library (HEMKER et al. [1979]). We have used the CYBER 73 computer with the NOS-BE system and the current ALGOL 3.343 compiler. The machine precision is 2^{-47}. We use the following notation in the tables.

MS	:	number of iteration steps required;
MF	:	number of function evaluations required;
N	:	the order of the problem;
C	:	the value of the parameter or values of parameters (e.g. in results for various order for problem 10, c = 1"-03,1 means that $s_r = 10^{-3}$ and $s_c = 1$);
MJ	:	the number of jacobian evaluations required;
MSV	:	the number of iterations with singular value decomposition (only in II.8).

For problem 5a, P and Q have the same meaning as p and q in the problem description.
If a program fails to solve a problem than we do not give the number of function evaluations required, but instead we give a "*" followed by the number of the error message given (number i means message texti as given in the problem prelude in ALGOL 68 (nlsprl in section 6.1)). In all tests we use fixed upper bounds on the allowed number of function evaluations. These are, with

M = min(100, 600 // n):

M for algorithms using the analytic jacobian;
M(n+1) for algorithms using difference approximation;

2M for update algorithms.
In the algorithms of component wise approximation, BW, BT and BTM, we allow M iteration steps. We have multiplied M with (n+1) for difference algorithms as in each iteration step n evaluations of the function are required to approximate the jacobian. As update algorithms are expected to use more iteration steps than algorithms with analytic jacobian we used 2M for these programs. These upper bounds decrease for increasing order. Although this might cause more failures for problems of high order, this bound is necessary for practical reasons, to be able to perform all tests within a reasonable amount of CPU hours.
The interpretations of these results are given in chapters 7 and 8. The test results are divided into 8 groups:

1. results for ASW, AS, AB, AI and AE;
2. results for GAS, GAB and GAI;
3. results for DSW, DS, DB, DI and DE;
4. results for GDS, GDB and GDI;
5. results for U1S and U2S;
6. results for BW and BT;
7. results to test special properties and features (convergence criteria, conditional updating and fixed approximation, scaling and reduction of problems with linear function components);
8. results of the algorithms SNOLEQ(S), SNOLEQJ(S), (SC)U2S and BW, which are selected based on the experimental results in 1. up to 7.

Note that in I.6, MS denotes the number of function component evaluations divided by n (rounded to below). Furthermore here BW(1) and BT(1) mean that we use the value $\sqrt[3]{\varepsilon}$ for the difference steps in BW and BT, and BW(2) and BT(2) indicate that $\sqrt{\varepsilon}$ is used. Without this indication $\sqrt[3]{\varepsilon}$ is used.

II.1. RESULTS for ASW, AS, AB, AI AND AE

RESULTS FOR PROBLEM 1

N	ASW		AS		AB		AI		AE	
	MS	MF	MS	MF	MS	MF	MS	MF	MS	MF
2	1	2	1	2	1	2	1	2	1	3
3	6	7	6	7	6	8	6	8	6	9
4	14	15	14	15	8	19	7	14	8	15
5	17	18	2	*09	7	14	7	14	9	16
6	59	60	2	*09	10	16	10	16	8	15
7	33	34	2	*09	6	15	6	15	9	16
8	2	*05	2	*05	9	22	9	22	3	*08
9	2	*05	2	*05	6	18	6	18	6	10
10	2	*05	2	*05	11	22	11	22	2	*09
13	2	*05	2	*05	7	25	7	25	2	*09
24	1	*11	2	*05	2	*09	3	4	3	5
35	1	*11	2	*05	2	*08	3	4	3	5

RESULTS FOR PROBLEM 2, N = 3

C	ASW		AS		AB		AI		AE	
	MS	MF	MS	MF	MS	MF	MS	MF	MS	MF
1"01	4	5	4	5	4	5	4	5	4	5
1"02	6	7	6	7	6	8	6	8	7	13
1"03	7	8	7	8	8	10	8	10	8	16
1"04	9	10	9	10	9	12	9	12	16	32
1"05	10	11	10	11	11	15	10	13	31	62
1"06	11	12	7	*09	8	*09	8	*09	44	*09
1"07	12	13	5	*09	7	*09	6	*09	154	*09
1"08	13	14	6	*08	5	*09	5	*09	147	*04
1"09	15	16	5	*09	5	*09	4	*09	149	*04
1"10	16	17	4	*08	3	*09	4	*08	143	*04
1"11	18	19	3	*08	3	*09	3	*08	139	*04

RESULTS FOR PROBLEM 4

N	C	ASW		AS		AB		AI		AE	
		MS	MF	MS	MF	MS	MF	MS	MF	MS	MF
2	1"01	8	9	8	9	17	32	14	22	24	42
2	1"02	6	7	6	7	2	*09	2	*09	2	*09
2	1"03	6	7	2	*09	2	*09	2	*09	2	*09
2	1"04	6	7	2	*09	2	*09	2	*09	2	*09
2	1"05	5	6	2	*08	2	*08	2	*08	2	*08
2	1"06	5	6	2	*08	2	*08	2	*08	2	*08
2	1"07	5	6	2	*08	2	*08	2	*08	2	*08
13	1"01	9	10	9	10	12	17	14	19	100	*04
13	1"02	31	32	31	32	77	*09	30	*09	14	26
13	1"03	45	*04	8	*09	8	*09	100	*04	100	*09
13	1"04	11	12	11	12	8	*09	8	*09	7	*09
13	1"05	45	*04	8	*09	8	*09	8	*09	9	*09
13	1"06	19	20	9	*08	43	*09	8	*09	8	*09
13	1"07	45	*04	8	*09	9	*09	9	*09	8	*08

RESULTS FOR PROBLEM 5

N	ASW		AS		AB		AI		AE	
	MS	MF	MS	MF	MS	MF	MS	MF	MS	MF
2	2	3	2	3	2	3	2	3	2	3
13	3	4	3	4	3	4	3	4	3	4
24	3	4	3	4	3	4	3	4	3	5
35	3	4	3	4	3	4	3	4	3	5
46	3	4	3	4	3	4	3	4	3	5

RESULTS FOR PROBLEM 5A, N = 35

P	Q	ASW			AS			AB			AI			AE		
		MS	MF	!!F!!	MS	MF	!!F!!	MS	MF	!!F!!	MS	MF	!!F!!	MS	MF	!!F!!
1"-12	1"-12	3	4	4"-11	3	4	4"-11	3	4	4"-11	3	4	4"-11	3	5	3"-11
1"-12	1"-10	3	4	4"-11	3	4	4"-11	3	4	4"-11	3	4	4"-11	3	5	5"-11
1"-12	1"-08	3	4	7"-11	3	4	7"-11	3	4	7"-11	3	4	7"-11	3	5	5"-11
1"-12	1"-06	3	4	1"-09	3	4	1"-09	3	4	1"-09	3	4	1"-09	3	5	6"-10
1"-12	1"-04	3	4	1"-07	3	4	1"-07	3	4	1"-07	3	4	1"-07	3	5	6"-08
1"-12	1"-02	3	4	4"-04	3	4	4"-04	3	4	4"-04	3	4	4"-04	4	6	2"-05
1"-10	1"-12	3	4	3"-10	3	4	3"-10	3	4	3"-10	3	4	3"-10	3	5	2"-10
1"-08	1"-12	3	4	2"-08	3	4	2"-08	3	4	2"-08	3	4	2"-08	3	5	2"-08
1"-06	1"-12	3	4	2"-06	3	4	2"-06	3	4	2"-06	3	4	2"-06	3	5	2"-06
1"-04	1"-12	3	4	2"-04	3	4	2"-04	3	4	2"-04	3	4	2"-04	3	5	2"-04
1"-02	1"-12	16	*04	2"-02	17	*04	2"+01	14	*09	2"-02	4	*01	2"-02	3	*01	2"-02

RESULTS FOR PROBLEM 7, N = 2

C	ASW		AS		AB		AI		AE	
	MS	MF	MS	MF	MS	MF	MS	MF	MS	MF
1"-1	3	4	3	4	3	4	3	4	3	4
1"+0	4	5	4	5	4	5	4	5	4	5
5"+0	5	6	5	6	5	6	5	6	5	6
1"+1	5	6	5	6	5	6	5	6	5	6
5"+1	6	7	6	7	6	7	6	7	6	7
1"+2	7	8	7	8	7	8	7	8	7	8
1"+3	9	10	9	10	9	10	9	10	9	10
1"+4	10	11	10	11	10	11	10	11	10	11
1"+5	12	13	12	13	12	13	12	13	12	13
1"+8	17	18	17	18	17	18	17	18	17	18

RESULTS FOR ORDER 2

FN	C	ASW		AB	
		MS	MF	MS	MF
1	-	1	2	1	2
2	1"1	5	6	5	7
3	-	2	3	2	3
4	1"1	8	9	17	32
4	1"4	6	7	2	*09
4	1"7	5	6	2	*08
5	-	2	3	2	3
6	-	5	6	5	6
7	1"1	5	6	5	6
7	1"4	10	11	10	11
8	-	4	5	4	5
9	-	4	5	4	5
10	1"+00,1	3	4	3	4
10	1"-03,1	3	4	3	4
10	1"-06,1	3	4	3	4
10	1"-09,1	3	4	3	4
10	1"-14,1	99	*04	2	*08
10	1,1"-03	4	5	4	5
10	1,1"-06	4	5	3	*08
10	1,1"-09	7	8	2	*08
10	1,1"-14	2	*05	2	*05
11	-	4	5	4	5
12	-	2	3	2	3
13	-	4	5	4	5
14	-	4	5	4	5

RESULTS FOR ORDER 13

FN	C	ASW		AB	
		MS	MF	MS	MF
1	-	2	*05	7	25
2	1"1	13	14	9	*09
3	-	14	15	2	*05
4	1"1	9	10	12	17
4	1"4	11	12	8	*09
4	1"7	45	*04	9	*09
5	-	3	4	3	4
6	-	5	6	5	6
7	1"1	5	6	5	6
7	1"4	10	11	10	11
8	-	4	5	4	5
9	-	4	5	4	5
10	1"+00,1	3	4	3	4
10	1"-03,1	3	4	3	4
10	1"-06,1	3	4	3	4
10	1"-09,1	3	4	3	4
10	1"-14,1	45	*04	2	*08
10	1,1"-03	4	5	4	5
10	1,1"-06	4	5	3	*09
10	1,1"-09	7	8	2	*08
10	1,1"-14	45	*04	2	*08
11	-	4	5	4	5
12	-	2	3	2	3
13	-	5	6	5	6
14	-	3	4	3	4

RESULTS FOR ORDER 24

FN	C	ASW		AB	
		MS	MF	MS	MF
1	-	1	*11	2	*09
2	1"1	2	*05	2	*09
3	-	24	*04	2	*05
4	1"1	9	10	9	10
4	1"4	12	13	7	*09
4	1"7	24	*04	7	*08
5	-	3	4	3	4
6	-	5	6	5	6
7	1"1	5	6	5	6
7	1"4	10	11	10	11
8	-	4	5	4	5
9	-	4	5	4	5
10	1"+00,1	3	4	3	4
10	1"-03,1	3	4	3	4
10	1"-06,1	3	4	3	4
10	1"-09,1	3	4	3	4
10	1"-14,1	24	*04	2	*08
10	1,1"-03	4	5	4	5
10	1,1"-06	4	5	3	4
10	1,1"-09	6	7	2	*08
10	1,1"-14	24	*04	2	*08
11	-	4	5	4	5
12	-	3	4	3	4
13	-	5	6	5	6
14	-	3	4	3	4

RESULTS FOR ORDER 35

FN	C	ASW		AB	
		MS	MF	MS	MF
1	-	1	*11	2	*08
2	1"1	2	*05	2	*08
3	-	16	*04	2	*05
4	1"1	9	10	9	*09
4	1"4	12	13	8	*09
4	1"7	16	*04	9	*09
5	-	3	4	3	4
6	-	5	6	5	6
7	1"1	5	6	5	6
7	1"4	10	11	10	11
8	-	4	5	4	5
9	-	4	5	4	5
10	1"+00,1	3	4	3	4
10	1"-03,1	3	4	3	4
10	1"-06,1	3	4	3	4
10	1"-09,1	3	4	3	4
10	1"-14,1	16	*04	2	*08
10	1,1"-03	4	5	4	5
10	1,1"-06	4	5	3	4
10	1,1"-09	9	10	2	*08
10	1,1"-14	16	*04	2	*08
11	-	4	5	4	5
12	-	3	4	3	4
13	-	16	*04	11	*09
14	-	5	6	5	6

RESULTS FOR ORDER 46

FN	C	ASW		AB	
		MS	MF	MS	MF
1	-	1	*11	1	*02
2	1"1	1	*11	2	*08
3	-	12	*04	2	*10
4	1"1	9	10	9	10
4	1"4	12	13	7	*09
4	1"7	12	*04	7	*08
5	-	3	4	3	4
6	-	5	6	5	6
7	1"1	5	6	5	6
7	1"4	10	11	10	11
8	-	4	5	4	5
9	-	4	5	4	5
10	1"+00,1	4	5	4	5
10	1"-03,1	4	5	4	5
10	1"-06,1	4	5	4	5
10	1"-09,1	4	5	4	5
10	1"-14,1	12	*04	2	*08
10	1,1"-03	8	9	2	*09
10	1,1"-06	7	8	2	*08
10	1,1"-09	6	7	2	*08
10	1,1"-14	12	*04	2	*08
11	-	4	5	4	5
12	-	3	4	3	4
13	-	7	*05	6	10
14	-	4	5	4	5

II.2. RESULTS FOR GAS, GAB AND GAI

RESULTS FOR PROBLEM 1

N	GAS		GAB		GAI	
	MS	MF	MS	MF	MS	MF
2	2	3	2	3	2	3
3	7	8	7	9	7	9
4	15	16	9	20	8	15
5	18	19	8	15	8	15
6	60	61	11	17	11	17
7	34	35	7	16	7	16
8	41	*03	10	23	10	23
9	52	53	6	18	6	18
10	60	*04	12	23	12	23
13	45	*04	8	26	8	26
24	1	*11	1	*04	1	*04
35	5	6	5	6	5	6

RESULTS FOR PROBLEM 2, N = 3

C	GAS		GAB		GAI	
	MS	MF	MS	MF	MS	MF
1"01	5	6	5	6	5	6
1"02	7	8	7	9	7	9
1"03	8	9	9	11	9	11
1"04	9	10	10	13	10	13
1"05	11	12	12	16	11	14
1"06	12	13	13	17	13	16
1"07	13	14	14	19	20	35
1"08	14	15	15	20	15	19
1"09	16	17	17	23	16	21
1"10	17	18	18	24	18	23
1"11	18	19	29	56	30	59

RESULTS FOR PROBLEM 4

N	C	GAS		GAB		GAI	
		MS	MF	MS	MF	MS	MF
2	1"01	9	10	17	32	15	23
2	1"02	7	8	11	*04	15	*04
2	1"03	6	7	8	*04	12	*04
2	1"04	6	7	7	*04	10	*04
2	1"05	5	6	6	*04	10	*04
2	1"06	5	6	6	*04	8	*04
2	1"07	5	6	2	*02	2	*01
13	1"01	10	11	13	18	15	20
13	1"02	32	33	12	*04	14	*04
13	1"03	45	*04	12	*04	12	*04
13	1"04	12	13	9	*04	10	*04
13	1"05	45	*04	11	*04	10	*04
13	1"06	45	*04	18	*04	16	*04
13	1"07	45	*04	11	*04	13	*04

RESULTS FOR PROBLEM 5

N	GAS		GAB		GAI	
	MS	MF	MS	MF	MS	MF
2	2	3	2	3	2	3
13	3	4	3	4	3	4
24	3	4	3	4	3	4
35	3	4	3	4	3	4
46	3	4	3	4	3	4

RESULTS FOR PROBLEM 5A, N = 35

P	Q	GAS			GAB			GAI		
		MS	MF	!!F!!	MS	MF	!!F!!	MS	MF	!!F!!
1"-12	1"-12	3	4	4"-11	3	4	4"-11	3	4	4"-11
1"-12	1"-10	3	4	5"-11	3	4	5"-11	3	4	5"-11
1"-12	1"-08	3	4	7"-11	3	4	7"-11	3	4	7"-11
1"-12	1"-06	3	4	1"-09	3	4	1"-09	3	4	1"-09
1"-12	1"-04	3	4	1"-07	3	4	1"-07	3	4	1"-07
1"-12	1"-02	3	4	3"-04	3	4	3"-04	3	4	3"-04
1"-10	1"-12	3	4	3"-10	3	4	3"-10	3	4	3"-10
1"-08	1"-12	3	4	2"-08	3	4	2"-08	3	4	2"-08
1"-06	1"-12	3	4	2"-06	3	4	2"-06	3	4	2"-06
1"-04	1"-12	3	4	2"-04	3	4	2"-04	3	4	2"-04
1"-02	1"-12	16	*04	2"-02	7	*04	2"-01	4	*01	2"-02

RESULTS FOR PROBLEM 7, N = 2

C	GAS		GAB		GAI	
	MS	MF	MS	MF	MS	MF
1"-1	4	5	4	5	4	5
1"+0	5	6	5	6	5	6
5"+0	6	7	6	7	6	7
1"+1	6	7	6	7	6	7
5"+1	7	8	7	8	7	8
1"+2	8	9	8	9	8	9
1"+3	9	10	9	10	9	10
1"+4	11	12	11	12	11	12
1"+5	12	13	12	13	12	13
1"+8	17	18	17	18	17	18

RESULTS FOR ORDER 2

FN	C	GAS		GAB	
		MS	MF	MS	MF
1	-	2	3	2	3
2	1"1	6	7	6	8
3	-	3	4	3	4
4	1"1	9	10	17	32
4	1"4	6	7	7	*04
4	1"7	5	6	2	*02
5	-	2	3	2	3
6	-	6	7	6	7
7	1"1	6	7	6	7
7	1"4	11	12	11	12
8	-	5	6	5	6
9	-	5	6	5	6
10	1"+00,1	3	4	3	4
10	1"-03,1	3	4	3	4
10	1"-06,1	3	4	3	4
10	1"-09,1	3	4	3	4
10	1"-14,1	3	4	3	4
10	1,1"-03	5	6	5	6
10	1,1"-06	5	6	5	6
10	1,1"-09	99	*04	4	7
10	1,1"-14	3	4	3	4
11	-	4	5	4	5
12	-	3	4	3	4
13	-	4	5	4	5
14	-	4	5	4	5

RESULTS FOR ORDER 13

FN	C	GAS		GAB	
		MS	MF	MS	MF
1	-	45	*04	8	26
2	1"1	14	15	6	*04
3	-	14	15	4	*04
4	1"1	10	11	13	18
4	1"4	12	13	9	*04
4	1"7	45	*04	11	*04
5	-	3	4	3	4
6	-	5	6	5	6
7	1"1	6	7	6	7
7	1"4	11	12	11	12
8	-	5	6	5	6
9	-	5	6	5	6
10	1"+00,1	3	4	3	4
10	1"-03,1	3	4	3	4
10	1"-06,1	3	4	3	4
10	1"-09,1	3	4	3	4
10	1"-14,1	3	4	3	4
10	1,1"-03	5	6	5	6
10	1,1"-06	5	6	5	6
10	1,1"-09	5	*03	3	4
10	1,1"-14	3	4	3	4
11	-	4	5	4	5
12	-	3	4	3	4
13	-	6	7	6	7
14	-	4	5	4	5

RESULTS FOR ORDER 24

FN	C	GAS		GAB	
		MS	MF	MS	MF
1	-	1	*11	1	*04
2	1"1	24	*04	3	*04
3	-	24	*04	2	*04
4	1"1	10	11	10	11
4	1"4	12	13	8	*04
4	1"7	24	*04	8	*04
5	-	3	4	3	4
6	-	5	6	5	6
7	1"1	6	7	6	7
7	1"4	11	12	11	12
8	-	5	6	5	6
9	-	5	6	5	6
10	1"+00,1	4	5	4	5
10	1"-03,1	4	5	4	5
10	1"-06,1	4	5	4	5
10	1"-09,1	4	5	4	5
10	1"-14,1	4	5	4	5
10	1,1"-03	5	6	5	6
10	1,1"-06	5	6	4	5
10	1,1"-09	7	*03	4	5
10	1,1"-14	4	5	4	5
11	-	4	5	4	5
12	-	3	4	3	4
13	-	5	6	5	6
14	-	4	5	4	5

RESULTS FOR ORDER 35

FN	C	GAS		GAB	
		MS	MF	MS	MF
1	-	5	6	5	6
2	1"1	16	*04	2	*04
3	-	16	*04	2	*04
4	1"1	10	11	8	*04
4	1"4	12	13	8	*04
4	1"7	16	*04	9	*04
5	-	3	4	3	4
6	-	5	6	5	6
7	1"1	6	7	6	7
7	1"4	11	12	11	12
8	-	5	6	5	6
9	-	5	6	5	6
10	1"+00,1	4	5	4	5
10	1"-03,1	4	5	4	5
10	1"-06,1	4	5	4	5
10	1"-09,1	4	5	4	5
10	1"-14,1	4	5	4	5
10	1,1"-03	5	6	5	6
10	1,1"-06	5	6	4	5
10	1,1"-09	8	*03	4	5
10	1,1"-14	3	4	3	4
11	-	4	5	4	5
12	-	4	5	4	5
13	-	16	*04	7	*04
14	-	6	7	6	7

RESULTS FOR ORDER 46

FN	C	GAS		GAB	
		MS	MF	MS	MF
1	-	5	6	5	6
2	1"1	1	*11	2	*04
3	-	12	*04	2	*04
4	1"1	10	11	10	11
4	1"4	12	13	7	*04
4	1"7	12	*04	7	*04
5	-	3	4	3	4
6	-	5	6	5	6
7	1"1	6	7	6	7
7	1"4	11	12	11	12
8	-	5	6	5	6
9	-	5	6	5	6
10	1"+00,1	4	5	4	5
10	1"-03,1	4	5	4	5
10	1"-06,1	4	5	4	5
10	1"-09,1	4	5	4	5
10	1"-14,1	4	5	4	5
10	1,1"-03	9	10	6	10
10	1,1"-06	8	9	7	8
10	1,1"-09	4	5	4	5
10	1,1"-14	4	5	4	5
11	-	4	5	4	5
12	-	4	5	4	5
13	-	12	*04	7	11
14	-	4	5	4	5

II.3. RESULTS FOR DSW, DS, DB, DI and DE

RESULTS FOR PROBLEM 1

N	DSW		DS		DB		DI		DE	
	MS	MF	MS	MF	MS	MF	MS	MF	MS	MF
2	2	7	2	7	2	7	2	7	2	8
3	6	25	6	25	6	26	6	26	6	28
4	14	71	14	71	8	51	7	42	8	47
5	17	103	2	*08	7	49	7	49	9	61
6	59	414	2	*08	10	76	10	76	8	63
7	33	265	2	*08	6	57	6	57	9	79
8	2	*05	2	*05	9	94	9	94	3	*05
9	2	*05	2	*05	6	72	6	72	6	64
10	2	*05	2	*05	11	132	11	132	2	*08
13	2	*05	2	*05	7	116	7	116	1	*01
24	1	*11	3	76	1	*08	3	76	3	77
35	1	*05	3	109	1	*05	3	109	3	110

RESULTS FOR PROBLEM 2, N = 3

C	DSW		DS		DB		DI		DE	
	MS	MF	MS	MF	MS	MF	MS	MF	MS	MF
1"01	4	17	4	17	4	17	4	17	4	17
1"02	6	25	6	25	6	26	6	26	7	34
1"03	7	29	7	29	8	34	8	34	9	44
1"04	9	37	9	37	9	39	9	39	15	77
1"05	10	41	7	*08	8	*08	7	*08	14	*08
1"06	11	45	6	*08	6	*08	6	*08	50	*08
1"07	12	49	5	*08	5	*08	5	*08	59	*04
1"08	13	53	5	*08	5	*08	5	*08	59	*04
1"09	15	61	4	*08	4	*08	4	*08	64	*04
1"10	16	65	3	*08	3	*08	4	*08	63	*04
1"11	17	69	2	*08	2	*08	3	*08	60	*04

RESULTS FOR PROBLEM 4

N	C	DSW		DS		DB		DI		DE	
		MS	MF	MS	MF	MS	MF	MS	MF	MS	MF
2	1"01	8	25	8	25	17	66	14	50	25	93
2	1"02	6	19	6	19	2	*09	2	*09	2	*09
2	1"03	6	19	2	*08	2	*08	2	*08	2	*08
2	1"04	7	22	2	*08	2	*08	2	*08	2	*08
2	1"05	9	28	2	*08	2	*08	2	*08	2	*08
2	1"06	14	43	2	*08	2	*08	2	*08	2	*08
2	1"07	22	67	2	*08	2	*08	2	*08	2	*08
13	1"01	9	127	9	127	12	173	14	201	87	*04
13	1"02	32	449	32	449	59	*09	15	226	14	208
13	1"03	43	603	8	*08	8	*09	70	*04	87	*04
13	1"04	11	155	7	*08	8	*08	8	*08	7	*08
13	1"05	43	*04	7	*08	8	*08	8	*08	9	*08
13	1"06	43	*04	8	*08	8	*08	8	*08	8	*08
13	1"07	43	*04	8	*08	9	*08	9	*08	8	*08

RESULTS FOR PROBLEM 5

N	DSW		DS		DB		DI		DE	
	MS	MF	MS	MF	MS	MF	MS	MF	MS	MF
2	2	7	2	7	2	7	2	7	2	7
13	3	43	3	43	3	43	3	43	3	43
24	3	76	3	76	3	76	3	76	3	77
35	3	109	3	109	3	109	3	109	3	110
46	3	142	3	142	3	142	3	142	3	143

RESULTS FOR PROBLEM 5A, N = 35

P	Q	DSW			DS			DB			DI			DE		
		MS	MF	!!F!!	MS	MF	!!F!!	MS	MF	!!F!!	MS	MF	!!F!!	MS	MF	!!F!!
1"-12	1"-12	3	109	2"-10	3	109	2"-10	3	109	2"-10	3	109	2"-10	3	110	2"-10
1"-12	1"-10	3	109	2"-10	3	109	2"-10	3	109	2"-10	3	109	2"-10	3	110	2"-10
1"-12	1"-08	3	109	2"-09	3	109	2"-09	3	109	2"-09	3	109	2"-09	3	110	5"-10
1"-12	1"-06	3	109	2"-04	3	109	2"-04	3	109	2"-04	3	109	2"-04	3	110	8"-05
1"-12	1"-04	17	*04	6"+03	11	*08	2"+06	5	*08	1"+02	7	*08	4"+01	7	*08	4"+01
1"-12	1"-02	17	*04	3"+03	5	*08	4"+03	5	*08	3"+03	3	*01	3"+03	1	*01	3"+03
1"-10	1"-12	3	109	6"-10	3	109	6"-10	3	109	6"-10	3	109	6"-10	3	110	5"-10
1"-08	1"-12	3	109	5"-08	3	109	5"-08	3	109	5"-08	3	109	5"-08	3	110	6"-08
1"-06	1"-12	3	109	2"-04	3	109	2"-04	3	109	2"-04	3	109	2"-04	3	110	2"-04
1"-04	1"-12	17	*04	5"+01	4	*08	1"+03	7	*08	6"+00	4	*01	1"+01	4	*01	5"+00
1"-02	1"-12	17	*04	5"+03	10	*04	1"+03	4	*08	1"+02	2	*01	1"+02	2	*01	1"+02

RESULTS FOR PROBLEM 7, N = 2

C	DSW		DS		DB		DI		DE	
	MS	MF	MS	MF	MS	MF	MS	MF	MS	MF
1"-1	3	10	3	10	3	10	3	10	3	10
1"+0	4	13	4	13	4	13	4	13	4	13
5"+0	5	16	5	16	5	16	5	16	5	16
1"+1	5	16	5	16	5	16	5	16	5	16
5"+1	6	19	6	19	6	19	6	19	6	19
1"+2	7	22	7	22	7	22	7	22	7	22
1"+3	9	28	9	28	9	28	9	28	9	28
1"+4	10	31	10	31	10	31	10	31	10	31
1"+5	12	37	12	37	12	37	12	37	12	37
1"+8	17	52	17	52	17	52	17	52	17	52

RESULTS FOR ORDER 2

FN	C	DSW		DB	
		MS	MF	MS	MF
1	-	2	7	2	7
2	1"1	5	16	5	17
3	-	3	10	3	10
4	1"1	8	25	17	66
4	1"4	7	22	2	*08
4	1"7	22	67	2	*08
5	-	2	7	2	7
6	-	5	16	5	16
7	1"1	5	16	5	16
7	1"4	10	31	10	31
8	-	4	13	4	13
9	-	4	13	4	13
10	1"+00,1	3	10	3	10
10	1"-03,1	3	10	3	10
10	1"-06,1	2	7	2	7
10	1"-09,1	3	10	1	*08
10	1"-14,1	2	*05	1	*08
10	1,1"-03	4	13	4	13
10	1,1"-06	5	16	2	*08
10	1,1"-09	29	*05	1	*08
10	1,1"-14	1	*05	1	*05
11	-	4	13	4	13
12	-	2	7	2	7
13	-	4	13	4	13
14	-	4	13	4	13

RESULTS FOR ORDER 13

FN	C	DSW		DB	
		MS	MF	MS	MF
1	-	2	*05	7	116
2	1"1	13	183	9	*09
3	-	14	197	2	*05
4	1"1	9	127	12	173
4	1"4	11	155	8	*08
4	1"7	43	*04	9	*08
5	-	3	43	3	43
6	-	5	71	5	71
7	1"1	5	71	5	71
7	1"4	10	141	10	141
8	-	4	57	4	57
9	-	4	57	4	57
10	1"+00,1	3	43	3	43
10	1"-03,1	3	43	3	43
10	1"-06,1	3	43	3	43
10	1"-09,1	43	*04	1	*08
10	1"-14,1	43	*04	1	*08
10	1,1"-03	4	57	4	57
10	1,1"-06	5	71	2	*08
10	1,1"-09	12	*05	1	*08
10	1,1"-14	1	*05	1	*05
11	-	4	57	4	57
12	-	2	29	2	29
13	-	5	71	5	71
14	-	3	43	3	43

RESULTS FOR ORDER 24

FN	C	DSW		DB	
		MS	MF	MS	MF
1	-	1	*11	1	*08
2	1"1	2	*05	2	*08
3	-	24	*04	2	*05
4	1"1	9	226	9	226
4	1"4	12	301	7	*08
4	1"7	24	*04	7	*08
5	-	3	76	3	76
6	-	5	126	5	126
7	1"1	5	126	5	126
7	1"4	10	251	10	251
8	-	4	101	4	101
9	-	4	101	4	101
10	1"+00,1	3	76	3	76
10	1"-03,1	3	76	3	76
10	1"-06,1	3	76	3	76
10	1"-09,1	24	*04	1	*08
10	1"-14,1	24	*04	1	*08
10	1,1"-03	4	101	4	101
10	1,1"-06	5	126	2	*08
10	1,1"-09	24	*04	1	*08
10	1,1"-14	1	*05	1	*05
11	-	4	101	4	101
12	-	3	76	3	76
13	-	5	126	5	126
14	-	3	76	3	76

RESULTS FOR ORDER 35

FN	C	DSW		DB	
		MS	MF	MS	MF
1	-	1	*05	1	*05
2	1"1	2	*05	2	*08
3	-	17	*04	2	*05
4	1"1	9	325	10	*09
4	1"4	12	433	8	*08
4	1"7	17	*04	9	*08
5	-	3	109	3	109
6	-	5	181	5	181
7	1"1	5	181	5	181
7	1"4	10	361	10	361
8	-	4	145	4	145
9	-	4	145	4	145
10	1"+00,1	3	109	3	109
10	1"-03,1	3	109	3	109
10	1"-06,1	4	145	4	145
10	1"-09,1	17	*04	1	*08
10	1"-14,1	17	*04	1	*08
10	1,1"-03	4	145	4	145
10	1,1"-06	17	*04	1	*08
10	1,1"-09	17	*04	1	*08
10	1,1"-14	1	*05	1	*05
11	-	4	145	4	145
12	-	3	109	3	109
13	-	17	*04	13	*08
14	-	5	181	5	181

RESULTS FOR ORDER 46

FN	C	DSW		DB	
		MS	MF	MS	MF
1	-	1	*05	1	*05
2	1"1	1	*11	2	*08
3	-	13	*04	2	*05
4	1"1	9	424	9	424
4	1"4	12	565	7	*08
4	1"7	13	*04	7	*08
5	-	3	142	3	142
6	-	5	236	5	236
7	1"1	5	236	5	236
7	1"4	10	471	10	471
8	-	4	189	4	189
9	-	4	189	4	189
10	1"+00,1	4	189	4	189
10	1"-03,1	4	189	4	189
10	1"-06,1	4	189	1	*08
10	1"-09,1	13	*04	1	*08
10	1"-14,1	13	*04	1	*08
10	1,1"-03	10	471	2	*08
10	1,1"-06	13	*04	1	*08
10	1,1"-09	13	*04	1	*08
10	1,1"-14	1	*05	1	*05
11	-	4	189	4	189
12	-	3	142	3	142
13	-	10	*05	6	286
14	-	4	189	4	189

II.4. RESULTS FOR GDS, GDB and GDI

RESULTS FOR PROBLEM 1

N	GDS		GDB		GDI	
	MS	MF	MS	MF	MS	MF
2	2	7	2	7	2	7
3	7	29	7	30	7	30
4	15	76	9	56	8	47
5	18	109	8	55	8	55
6	29	204	11	83	11	83
7	32	257	7	65	7	65
8	49	442	10	103	10	103
9	51	511	6	72	6	72
10	60	*04	12	143	12	143
13	46	*04	8	130	8	130
24	5	126	5	126	5	126
35	5	181	5	181	5	181

RESULTS FOR PROBLEM 2, N = 3

C	GDS		GDB		GDI	
	MS	MF	MS	MF	MS	MF
1"01	5	21	5	21	5	21
1"02	7	29	7	30	7	30
1"03	8	33	9	38	9	38
1"04	10	41	10	43	10	43
1"05	11	45	12	52	11	47
1"06	12	49	13	56	13	55
1"07	14	57	15	65	20	95
1"08	23	*07	15	*02	15	*02
1"09	63	*03	14	*02	14	*02
1"10	27	*03	16	*02	17	*01
1"11	100	*04	20	*02	14	*02

RESULTS FOR PROBLEM 4

N	C	GDS		GDB		GDI	
		MS	MF	MS	MF	MS	MF
2	1"01	9	28	17	66	15	53
2	1"02	7	22	26	*04	34	*04
2	1"03	8	25	20	*04	30	*04
2	1"04	8	*07	19	*04	23	*04
2	1"05	8	*03	4	*02	4	*02
2	1"06	8	*03	4	*02	4	*02
2	1"07	8	*03	4	*02	4	*02
13	1"01	10	141	12	173	14	201
13	1"02	33	463	30	*04	16	241
13	1"03	46	645	26	*04	30	*04
13	1"04	12	169	23	*04	25	*04
13	1"05	20	281	19	*04	20	*04
13	1"06	11	*03	19	*04	20	*04
13	1"07	9	*03	10	*02	10	*02

RESULTS FOR PROBLEM 5

N	GDS		GDB		GDI	
	MS	MF	MS	MF	MS	MF
2	2	7	2	7	2	7
13	3	43	3	43	3	43
24	3	76	3	76	3	76
35	3	109	3	109	3	109
46	3	142	3	142	3	142

RESULTS FOR PROBLEM 5A, N = 35

P	Q	GDS			GDB			GDI		
		MS	MF	!!F!!	MS	MF	!!F!!	MS	MF	!!F!!
1"-12	1"-12	3	109	4"-10	3	109	4"-10	3	109	4"-10
1"-12	1"-10	3	109	4"-10	3	109	4"-10	3	109	4"-10
1"-12	1"-08	3	109	4"-09	3	109	4"-09	3	109	4"-09
1"-12	1"-06	3	109	2"-07	3	109	2"-07	3	109	2"-07
1"-12	1"-04	7	253	6"-05	7	253	6"-05	7	253	6"-05
1"-12	1"-02	17	*04	3"+03	6	*04	3"+03	3	*01	3"+03
1"-10	1"-12	3	109	5"-10	3	109	5"-10	3	109	5"-10
1"-08	1"-12	3	109	4"-08	3	109	4"-08	3	109	4"-08
1"-06	1"-12	3	109	4"-04	3	109	4"-04	3	109	4"-04
1"-04	1"-12	17	*04	2"-01	10	*07	5"-02	4	*01	7"-02
1"-02	1"-12	17	*04	7"+03	8	*07	7"+01	4	*01	1"+02

RESULTS FOR PROBLEM 7, N = 2

C	GDS		GDB		GDI	
	MS	MF	MS	MF	MS	MF
1"-1	4	13	4	13	4	13
1"+0	5	16	5	16	5	16
5"+0	6	19	6	19	6	19
1"+1	6	19	6	19	6	19
5"+1	7	22	7	22	7	22
1"+2	8	25	8	25	8	25
1"+3	9	28	9	28	9	28
1"+4	11	34	11	34	11	34
1"+5	12	37	12	37	12	37
1"+8	17	52	17	52	17	52

RESULTS FOR ORDER 2

FN	C	GDS		GDB	
		MS	MF	MS	MF
1	-	2	7	2	7
2	1"1	6	19	6	20
3	-	3	10	3	10
4	1"1	9	28	17	66
4	1"4	8	*07	19	*04
4	1"7	8	*03	4	*02
5	-	2	7	2	7
6	-	6	19	6	19
7	1"1	6	19	6	19
7	1"4	11	34	11	34
8	-	5	16	5	16
9	-	5	16	5	16
10	1"+00,1	3	10	3	10
10	1"-03,1	3	10	3	10
10	1"-06,1	3	10	3	10
10	1"-09,1	3	10	3	10
10	1"-14,1	3	10	3	10
10	1,1"-03	5	16	5	16
10	1,1"-06	6	19	5	19
10	1,1"-09	3	10	3	10
10	1,1"-14	3	10	3	10
11	-	4	13	4	13
12	-	3	10	3	10
13	-	4	13	4	13
14	-	4	13	4	13

RESULTS FOR ORDER 13

FN	C	GDS		GDB	
		MS	MF	MS	MF
1	-	46	*04	8	130
2	1"1	14	197	22	*04
3	-	15	211	23	*04
4	1"1	10	141	12	173
4	1"4	12	169	23	*04
4	1"7	9	*03	10	*02
5	-	3	43	3	43
6	-	5	71	5	71
7	1"1	6	85	6	85
7	1"4	11	155	11	155
8	-	5	71	5	71
9	-	5	71	5	71
10	1"+00,1	3	43	3	43
10	1"-03,1	3	43	3	43
10	1"-06,1	6	*03	4	*02
10	1"-09,1	3	43	3	43
10	1"-14,1	3	43	3	43
10	1,1"-03	5	71	5	71
10	1,1"-06	6	*03	4	*02
10	1,1"-09	3	43	3	43
10	1,1"-14	3	43	3	43
11	-	4	57	4	57
12	-	3	43	3	43
13	-	6	85	6	85
14	-	4	57	4	57

RESULTS FOR ORDER 24

FN	C	GDS		GDB	
		MS	MF	MS	MF
1	-	5	126	5	126
2	1"1	25	*04	13	*04
3	-	25	*04	18	*04
4	1"1	10	251	10	251
4	1"4	12	301	20	*04
4	1"7	8	*03	8	*02
5	-	3	76	3	76
6	-	5	126	5	126
7	1"1	6	151	6	151
7	1"4	11	276	11	276
8	-	5	126	5	126
9	-	5	126	5	126
10	1"+00,1	4	101	4	101
10	1"-03,1	4	101	4	101
10	1"-06,1	4	101	4	101
10	1"-09,1	4	101	4	101
10	1"-14,1	4	101	4	101
10	1,1"-03	5	126	5	126
10	1,1"-06	6	*03	5	*02
10	1,1"-09	4	101	4	101
10	1,1"-14	4	101	4	101
11	-	4	101	4	101
12	-	3	76	3	76
13	-	5	126	5	126
14	-	4	101	4	101

RESULTS FOR ORDER 35

FN	C	GDS		GDB	
		MS	MF	MS	MF
1	-	5	181	5	181
2	1"1	17	*04	4	*02
3	-	12	*03	3	*02
4	1"1	10	361	15	*04
4	1"4	12	433	14	*04
4	1"7	12	*07	11	*02
5	-	3	109	3	109
6	-	5	181	5	181
7	1"1	6	217	6	217
7	1"4	11	397	11	397
8	-	5	181	5	181
9	-	5	181	5	181
10	1"+00,1	4	145	4	145
10	1"-03,1	4	145	4	145
10	1"-06,1	4	145	4	145
10	1"-09,1	4	145	4	145
10	1"-14,1	4	145	4	145
10	1,1"-03	5	181	5	181
10	1,1"-06	3	109	3	109
10	1,1"-09	3	109	3	109
10	1,1"-14	3	109	3	109
11	-	4	145	4	145
12	-	4	145	4	145
13	-	17	*04	15	*04
14	-	6	217	6	217

RESULTS FOR ORDER 46

FN	C	GDS		GDB	
		MS	MF	MS	MF
1	-	5	236	5	236
2	1"1	1	*11	5	*02
3	-	13	*04	13	*04
4	1"1	10	471	10	471
4	1"4	12	565	13	*04
4	1"7	8	*03	8	*02
5	-	3	142	3	142
6	-	5	236	5	236
7	1"1	6	283	6	283
7	1"4	11	518	11	518
8	-	5	236	5	236
9	-	5	236	5	236
10	1"+00,1	4	189	4	189
10	1"-03,1	4	189	4	189
10	1"-06,1	4	189	4	189
10	1"-09,1	4	189	4	189
10	1"-14,1	4	189	4	189
10	1,1"-03	10	471	6	285
10	1,1"-06	4	189	4	189
10	1,1"-09	4	189	4	189
10	1,1"-14	4	189	4	189
11	-	4	189	4	189
12	-	4	189	4	189
13	-	7	*03	7	333
14	-	4	189	4	189

II.5. RESULTS FOR U1S AND U2S

RESULTS FOR PROBLEM 1

N	U1S		U2S	
	MS	MF	MS	MF
2	2	5	2	5
3	15	19	11	15
4	9	14	9	14
5	10	16	10	16
6	199	*04	11	18
7	199	*04	199	*04
8	2	*11	199	*04
9	2	*11	2	*11
10	2	*11	2	*11
13	2	*11	2	*11
24	1	*11	1	*11
35	1	*11	2	*11

RESULTS FOR PROBLEM 2, N = 3

C	U1S		U2S	
	MS	MF	MS	MF
1"01	8	12	8	12
1"02	12	16	17	21
1"03	34	38	199	*04
1"04	61	*11	94	*11
1"05	28	32	3	*11
1"06	199	*04	4	*11
1"07	199	*04	5	*11
1"08	62	*11	199	*04
1"09	199	*04	199	*04
1"10	199	*04	4	*11
1"11	199	*04	4	*11

RESULTS FOR PROBLEM 4

N	C	U1S		U2S	
		MS	MF	MS	MF
2	1"01	199	*04	151	154
2	1"02	199	*04	190	193
2	1"03	199	*04	199	*04
2	1"04	199	*04	199	*04
2	1"05	199	*04	199	*04
2	1"06	199	*04	199	*04
2	1"07	199	*04	199	*04
13	1"01	91	*04	91	*04
13	1"02	91	*04	91	*04
13	1"03	91	*04	91	*04
13	1"04	91	*04	91	*04
13	1"05	91	*04	91	*04
13	1"06	91	*04	91	*04
13	1"07	91	*04	91	*04

RESULTS FOR PROBLEM 5

N	U1S		U2S	
	MS	MF	MS	MF
2	2	5	2	5
13	4	18	4	18
24	4	29	4	29
35	5	41	5	41
46	4	51	4	51

RESULTS FOR PROBLEM 5A, N = 35

P	Q	U1S			U2S		
		MS	MF	!!F!!	MS	MF	!!F!!
1"-12	1"-12	3	39	7"-04	3	39	7"-04
1"-12	1"-10	3	39	7"-04	3	39	7"-04
1"-12	1"-08	3	39	7"-04	3	39	7"-04
1"-12	1"-06	4	40	5"-04	4	40	4"-04
1"-12	1"-04	15	51	4"-04	15	51	4"-04
1"-12	1"-02	33	*04	4"+03	33	*04	8"+03
1"-10	1"-12	3	39	7"-04	3	39	7"-04
1"-08	1"-12	3	39	7"-04	3	39	7"-04
1"-06	1"-12	3	39	6"-04	3	39	6"-04
1"-04	1"-12	4	40	2"-04	4	40	2"-04
1"-02	1"-12	33	*04	1"-02	33	*04	3"-02

RESULTS FOR PROBLEM 7, N = 2

C	U1S		U2S	
	MS	MF	MS	MF
1"-1	5	8	5	8
1"+0	7	10	7	10
5"+0	9	12	9	12
1"+1	10	13	10	13
5"+1	13	16	13	16
1"+2	14	17	14	17
1"+3	17	20	15	18
1"+4	19	22	17	20
1"+5	20	23	20	23
1"+8	24	27	24	27

RESULTS FOR ORDER 2

FN	C	U1S		U2S	
		MS	MF	MS	MF
1	-	2	5	2	5
2	1"1	16	19	24	27
3	-	4	7	4	7
4	1"1	199	*04	151	154
4	1"4	199	*04	199	*04
4	1"7	199	*04	199	*04
5	-	2	5	2	5
6	-	7	10	7	10
7	1"1	10	13	10	13
7	1"4	19	22	17	20
8	-	6	9	6	9
9	-	6	9	6	9
10	1"+00,1	4	7	4	7
10	1"-03,1	4	7	4	7
10	1"-06,1	4	7	4	7
10	1"-09,1	4	7	4	7
10	1"-14,1	4	7	4	7
10	1,1"-03	8	11	8	11
10	1,1"-06	8	11	8	11
10	1,1"-09	199	*04	199	*04
10	1,1"-14	199	*04	199	*04
11	-	6	9	6	9
12	-	4	7	4	7
13	-	6	9	6	9
14	-	6	9	6	9

RESULTS FOR ORDER 13

FN	C	U1S		U2S	
		MS	MF	MS	MF
1	-	2	*11	2	*11
2	1"1	91	*04	91	*04
3	-	91	*04	3	*11
4	1"1	91	*04	91	*04
4	1"4	91	*04	91	*04
4	1"7	91	*04	91	*04
5	-	4	18	4	18
6	-	13	27	13	27
7	1"1	17	31	17	31
7	1"4	24	38	22	36
8	-	6	20	6	20
9	-	6	20	6	20
10	1"+00,1	5	19	5	19
10	1"-03,1	5	19	5	19
10	1"-06,1	5	19	5	19
10	1"-09,1	5	19	5	19
10	1"-14,1	5	19	5	19
10	1,1"-03	7	21	7	21
10	1,1"-06	10	24	10	24
10	1,1"-09	20	34	17	31
10	1,1"-14	91	*04	91	*04
11	-	8	22	8	22
12	-	4	18	4	18
13	-	12	26	12	26
14	-	7	21	7	21

RESULTS FOR ORDER 24

FN	C	U1S		U2S	
		MS	MF	MS	MF
1	-	1	*11	1	*11
2	1"1	2	*11	2	*11
3	-	49	*04	3	*11
4	1"1	49	*04	19	*04
4	1"4	49	*04	49	*04
4	1"7	49	*04	49	*04
5	-	4	29	4	29
6	-	13	38	13	38
7	1"1	17	42	17	42
7	1"4	25	50	22	47
8	-	6	31	6	31
9	-	6	31	6	31
10	1"+00,1	6	31	6	31
10	1"-03,1	6	31	6	31
10	1"-06,1	6	31	6	31
10	1"-09,1	6	31	6	31
10	1"-14,1	6	31	6	31
10	1,1"-03	8	33	8	33
10	1,1"-06	12	37	12	37
10	1,1"-09	15	40	22	47
10	1,1"-14	49	*04	49	*04
11	-	9	34	9	34
12	-	4	29	4	29
13	-	18	43	19	44
14	-	7	32	7	32

RESULTS FOR ORDER 35

FN	C	U1S		U2S	
		MS	MF	MS	MF
1	-	2	*11	2	*11
2	1"1	2	*11	2	*11
3	-	33	*04	3	*11
4	1"1	33	*04	33	*04
4	1"4	33	*04	33	*04
4	1"7	33	*04	33	*04
5	-	5	41	5	41
6	-	13	49	13	49
7	1"1	17	53	17	53
7	1"4	26	62	22	58
8	-	6	42	6	42
9	-	6	42	6	42
10	1"+00,1	6	42	6	42
10	1"-03,1	6	42	6	42
10	1"-06,1	6	42	6	42
10	1"-09,1	6	42	6	42
10	1"-14,1	6	42	6	42
10	1,1"-03	11	47	11	47
10	1,1"-06	15	51	15	51
10	1,1"-09	15	51	19	55
10	1,1"-14	33	*04	33	*04
11	-	9	45	9	45
12	-	5	41	5	41
13	-	19	55	33	*04
14	-	10	46	10	46

RESULTS FOR ORDER 46

FN	C	U1S		U2S	
		MS	MF	MS	MF
1	-	2	*11	2	*11
2	1"1	1	*11	1	*11
3	-	25	*04	3	*11
4	1"1	25	*04	25	*04
4	1"4	25	*04	25	*04
4	1"7	25	*04	25	*04
5	-	4	51	4	51
6	-	13	60	13	60
7	1"1	17	64	17	64
7	1"4	25	*04	22	69
8	-	6	53	6	53
9	-	6	53	6	53
10	1"+00,1	7	54	7	54
10	1"-03,1	7	54	7	54
10	1"-06,1	7	54	7	54
10	1"-09,1	7	54	7	54
10	1"-14,1	7	54	7	54
10	1,1"-03	11	58	11	58
10	1,1"-06	22	69	22	69
10	1,1"-09	18	65	25	*04
10	1,1"-14	25	*04	25	*04
11	-	10	57	10	57
12	-	5	52	5	52
13	-	25	*04	18	65
14	-	8	55	8	55

II.6. RESULTS FOR BW AND BT

RESULTS FOR PROBLEM 1

N	BW(1)		BW(2)		BT(1)		BT(2)	
	MS	MF	MS	MF	MS	MF	MS	MF
2	2	5	2	5	2	5	2	5
3	7	21	7	21	7	21	7	21
4	15	52	7	24	15	52	7	24
5	19	76	7	28	18	72	7	28
6	60	270	8	36	60	270	8	36
7	34	170	8	40	34	170	8	40
8	66	363	8	44	67	368	8	44
9	55	330	1	*05	55	330	8	48
10	61	*04	1	*05	61	*04	8	52
13	8	64	1	*05	9	72	9	72
24	9	121	1	*05	9	121	9	121
35	9	171	1	*05	10	190	1	*05

RESULTS FOR PROBLEM 2, N = 3

C	BW(1)		BW(2)		BT(1)		BT(2)	
	MS	MF	MS	MF	MS	MF	MS	MF
1"01	5	15	5	15	5	15	5	15
1"02	7	21	7	21	7	21	7	21
1"03	8	24	8	24	8	24	8	24
1"04	11	*05	31	*11	10	30	10	30
1"05	10	*05	32	*11	11	33	11	33
1"06	12	*05	36	*11	13	39	13	39
1"07	14	*05	55	*01	12	*11	71	*11
1"08	13	*05	100	*04	13	*05	12	*05
1"09	12	*05	24	*01	100	*04	7	*05
1"10	16	*05	22	*01	100	*04	20	60
1"11	20	*05	100	*04	21	63	21	63

RESULTS FOR PROBLEM 4

N	C	BW(1)		BW(2)		BT(1)		BT(2)	
		MS	MF	MS	MF	MS	MF	MS	MF
2	1"01	13	32	13	32	13	32	13	32
2	1"02	17	42	16	40	18	45	16	40
2	1"03	28	70	22	55	32	80	22	55
2	1"04	100	*04	20	50	100	*04	20	50
2	1"05	100	*04	29	72	100	*04	29	72
2	1"06	100	*04	72	180	100	*04	72	180
2	1"07	100	*04	100	*04	100	*04	100	*04
13	1"01	9	72	9	72	47	*04	47	*04
13	1"02	47	*04	26	208	47	*04	47	*04
13	1"03	15	120	16	128	47	*04	27	216
13	1"04	20	160	13	104	47	*04	24	192
13	1"05	47	*04	17	138	47	*04	23	184
13	1"06	20	*01	20	*01	47	*04	47	*04
13	1"07	47	*04	47	*04	47	*04	47	*04

RESULTS FOR PROBLEM 5

N	BW		BT	
	MS	MF	MS	MF
2	3	8	3	8
13	3	24	3	24
24	4	54	4	54
35	4	76	4	76
46	4	98	4	98

RESULTS FOR PROBLEM 5A, N = 35

P	Q	BW			BT		
		MS	MF	!!F!!	MS	MF	!!F!!
1"-12	1"-12	3	57	2"-09	3	57	2"-09
1"-12	1"-10	3	57	2"-09	3	57	2"-09
1"-12	1"-08	3	57	2"-09	3	57	2"-09
1"-12	1"-06	3	57	2"-09	3	57	2"-09
1"-12	1"-04	3	57	2"-09	3	57	2"-09
1"-12	1"-02	3	57	2"-09	3	57	2"-09
1"-10	1"-12	3	57	2"-09	3	57	2"-09
1"-08	1"-12	3	57	2"-08	3	57	2"-08
1"-06	1"-12	3	57	2"-06	3	57	2"-06
1"-04	1"-12	3	57	2"-04	3	57	2"-04
1"-02	1"-12	3	57	2"-02	3	57	2"-02

RESULTS FOR PROBLEM 7, N = 2

C	BW(1)		BW(2)		BT(1)		BT(2)	
	MS	MF	MS	MF	MS	MF	MS	MF
1"-1	4	10	4	10	4	10	4	10
1"+0	5	12	5	12	5	12	5	12
5"+0	6	15	6	15	6	15	6	15
1"+1	6	15	6	15	6	15	7	17
5"+1	7	17	7	17	8	20	8	20
1"+2	8	20	8	20	8	20	8	20
1"+3	9	22	10	25	9	22	10	25
1"+4	12	30	11	27	12	30	11	27
1"+5	13	32	13	32	13	32	13	32
1"+8	21	52	18	45	21	52	18	45

RESULTS FOR ORDER 2

FN	C	BW		BT	
		MS	MF	MS	MF
1	-	2	5	2	5
2	1"1	6	15	6	15
3	-	2	5	2	5
4	1"1	13	32	13	32
4	1"4	20	50	20	50
4	1"7	100	*04	100	*04
5	-	3	8	3	8
6	-	6	15	6	15
7	1"1	6	15	6	15
7	1"4	12	30	12	30
8	-	5	13	5	13
9	-	5	13	4	10
10	1"+00,1	3	8	3	8
10	1"-03,1	3	8	3	8
10	1"-06,1	3	8	3	8
10	1"-09,1	3	8	3	8
10	1"-14,1	3	8	1	*05
10	1,1"-03	4	10	4	10
10	1,1"-06	4	10	4	10
10	1,1"-09	3	8	3	8
10	1,1"-14	3	8	1	*05
11	-	5	13	5	13
12	-	3	8	3	8
13	-	5	13	5	13
14	-	4	10	5	13

RESULTS FOR ORDER 13

FN	C	BW		BT	
		MS	MF	MS	MF
1	-	8	64	9	72
2	1"1	4	*05	2	*05
3	-	2	16	2	16
4	1"1	9	72	47	*04
4	1"4	13	104	24	192
4	1"7	47	*04	47	*04
5	-	3	24	3	24
6	-	6	48	6	48
7	1"1	6	48	7	56
7	1"4	12	96	12	96
8	-	4	32	5	40
9	-	4	32	4	32
10	1"+00,1	4	32	4	32
10	1"-03,1	4	32	4	32
10	1"-06,1	4	32	4	32
10	1"-09,1	4	32	4	32
10	1"-14,1	4	32	1	*05
10	1,1"-03	5	40	5	40
10	1,1"-06	6	48	47	*04
10	1,1"-09	4	32	5	40
10	1,1"-14	4	32	4	32
11	-	5	40	5	40
12	-	3	24	3	24
13	-	5	40	5	40
14	-	5	40	4	32

RESULTS FOR ORDER 24

FN	C	BW		BT	
		MS	MF	MS	MF
1	-	9	121	9	121
2	1"1	2	*11	2	*11
3	-	2	27	2	27
4	1"1	9	122	26	*04
4	1"4	20	322	26	*04
4	1"7	26	*04	26	*04
5	-	4	54	4	54
6	-	6	81	6	81
7	1"1	6	81	7	95
7	1"4	12	162	12	162
8	-	4	54	5	68
9	-	4	54	4	54
10	1"+00,1	5	68	4	54
10	1"-03,1	5	68	4	54
10	1"-06,1	5	68	4	54
10	1"-09,1	5	68	4	54
10	1"-14,1	5	68	1	*05
10	1,1"-03	5	68	6	81
10	1,1"-06	6	81	26	*04
10	1,1"-09	5	68	8	108
10	1,1"-14	5	68	8	108
11	-	6	81	6	81
12	-	4	54	4	54
13	-	7	95	6	81
14	-	5	68	4	54

RESULTS FOR ORDER 35

FN	C	BW		BT	
		MS	MF	MS	MF
1	-	9	171	10	190
2	1"1	2	*11	2	*11
3	-	2	38	2	38
4	1"1	9	171	18	*04
4	1"4	18	*04	18	*04
4	1"7	18	*04	18	*04
5	-	4	76	4	76
6	-	6	114	6	114
7	1"1	6	114	7	133
7	1"4	12	228	12	228
8	-	4	76	5	95
9	-	4	76	4	76
10	1"+00,1	5	95	5	95
10	1"-03,1	5	95	5	95
10	1"-06,1	5	95	5	95
10	1"-09,1	5	95	5	95
10	1"-14,1	5	95	1	*05
10	1,1"-03	18	*04	6	114
10	1,1"-06	13	247	18	*04
10	1,1"-09	8	152	11	*05
10	1,1"-14	7	133	6	114
11	-	6	114	5	95
12	-	4	76	3	57
13	-	18	*04	18	*04
14	-	7	133	6	114

RESULTS FOR ORDER 46

FN	C	BW		BT	
		MS	MF	MS	MF
1	-	10	245	1	*05
2	1"1	2	*11	2	*11
3	-	2	49	2	49
4	1"1	9	221	14	*04
4	1"4	14	*04	14	*04
4	1"7	14	*04	14	*04
5	-	4	98	4	98
6	-	6	147	6	147
7	1"1	6	147	7	171
7	1"4	12	294	12	294
8	-	4	98	5	122
9	-	4	98	4	98
10	1"+00,1	6	147	5	122
10	1"-03,1	6	147	5	122
10	1"-06,1	6	147	5	122
10	1"-09,1	6	147	5	122
10	1"-14,1	6	147	1	*05
10	1,1"-03	5	122	7	171
10	1,1"-06	7	171	14	*04
10	1,1"-09	14	*04	5	122
10	1,1"-14	5	122	4	98
11	-	6	147	5	122
12	-	4	98	3	73
13	-	9	221	6	147
14	-	5	122	5	122

II.7. RESULTS FOR SPECIAL PROPERTIES AND FEATURES

II.7.1.

In these tables we use the notation $P = -{}^{10}\log(\|\bar{x} - x^*\|)$, where x^* is the solution and $\bar{x}$ the computed approximation. Problems 1 and 7 are run for $\delta_f = \delta_{rx} = \delta_{ax} = 10^{-3}$, 10^{-7} and 10^{-11}. The results for these three required precisions are given on the three subsequent lines given for each problem. Failure is denoted by a "*". Problem 15 is run for $\delta_f = \delta_{rx} = \delta_{ax} = c$, with c as given in column three. For this problem we give also the number of iteration steps performed and the error message (behind the star), if the algorithm fails.

EXPERIMENTS TO TEST CONVERGENCE CRITERIA

FN	N	C	ASWC		ASW		AB		GAS	
			MS	P	MS	P	MS	P	MS	P
1	5	–	17	7	16	4	6	6	17	7
			18	13	17	7	7	12	18	13
			19	13	18	13	7	12	19	13
1	6	–	59	8	58	4	9	5	59	8
			60	13	59	8	10	8	60	13
			61	13	60	13	11	5	61	14
1	7	–	33	9	32	5	5	4	33	9
			34	13	33	9	6	9	34	13
			35	13	34	13	7	13	35	13
1	8	–	*		*		8	4	*	
			*		*		9	7	*	
			*		*		10	13	*	
1	9	–	*		*		4	3	51	9
			*		*		6	13	52	13
			*		*		6	13	53	15
1	10	–	*		*		9	4	*	
			*		*		11	12	*	
			*		*		11	12	*	
1	13	–	*		*		6	4	*	
			*		*		7	9	*	
			*		*		8	12	*	
1	24	–	*		*		*		*	
			*		*		*		*	
			*		*		*		*	
7	2	10	5	9	4	5	4	5	5	9
			6	15	5	9	5	9	6	15
			7	15	6	15	6	15	7	15
7	2	"3	8	8	8	8	8	8	8	8
			9	13	9	13	9	13	9	13
			10	13	9	13	9	13	10	13
7	2	"5	11	8	11	8	11	8	11	8
			12	13	12	13	12	13	12	13
			13	14	13	14	13	14	13	14
7	2	"8	16	9	16	9	16	9	16	9
			17	11	17	11	17	11	17	11
			18	11	18	11	18	11	18	11
15	4	"-1	7	1	*5	7	*7	2	7	1
15	4	"-2	9	1	*5	7	*7	2	9	1
15	4	"-3	14	2	*5	7	*7	2	14	2
15	4	"-4	20	3	*5	7	*7	2	20	3
15	4	"-5	26	4	*5	7	*7	2	26	4
15	4	"-6	31	5	*5	7	*7	2	31	5
15	4	"-7	37	6	*5	7	*7	2	*3	5
15	4	"-8	*5	6	*5	7	*7	2	*3	5
15	4	"-9	*5	6	*5	7	*7	2	*3	5
15	4	"-10	*5	6	*5	7	*7	2	*3	5

FN	N	C	DB		GDS		U2S		BW		BT	
			MS	P	MS	P	MS	P	MS	P	MS	P
1	5	-	6	6	17	7	8	6	17	7	17	7
			7	12	18	13	10	13	19	13	18	13
			7	12	19	13	14	10	19	13	19	13
1	6	-	9	5	27	5	9	5	58	4	59	8
			10	8	29	15	11	9	60	15	60	13
			11	15	29	15	*		61	15	61	15
1	7	-	5	4	31	8	40	4	33	9	33	9
			6	9	32	13	*		34	13	34	13
			7	13	33	13	*		35	13	35	13
1	8	-	8	4	47	6	*		65	6	65	6
			9	7	49	13	*		66	12	67	14
			10	13	49	13	*		67	15	67	14
1	9	-	4	3	50	8	*		54	11	54	11
			6	14	51	13	*		55	14	55	13
			6	14	52	15	*		56	13	56	13
1	10	-	9	4	*		*		*		*	
			11	12	*		*		*		*	
			11	12	*		*		*		*	
1	13	-	6	4	*		*		6	4	7	7
			7	9	*		*		8	13	9	13
			8	12	*		*		9	13	9	13
1	24	-	*		*		*		6	1	8	1
			*		*		*		9	1	9	12
			*		*		*		10	15	10	1
7	2	10	4	5	5	9	8	6	5	10	5	8
			5	9	6	15	10	9	6	14	6	12
			6	15	7	15	12	15	7	15	7	15
7	2	"3	8	8	8	8	10	5	9	13	9	12
			9	13	9	13	15	9	9	13	9	13
			9	13	10	13	18	13	10	13	11	13
7	2	"5	11	8	11	8	15	7	12	10	12	10
			12	13	12	13	20	13	13	13	13	13
			13	14	13	14	21	13	15	14	15	14
7	2	"8	16	9	16	9	22	4	17	9	17	9
			17	11	17	11	24	11	21	11	21	11
			18	11	18	11	30	11	25	11	25	11
15	4	"-1	*8	3	7	1	10	1	*5	1	*5	1
15	4	"-2	*8	3	9	1	13	1	*5	1	*5	1
15	4	"-3	*8	3	14	2	19	2	*4	1	*5	1
15	4	"-4	*8	3	20	3	27	3	*4	1	*4	1
15	4	"-5	*8	3	26	4	35	4	*4	1	*4	1
15	4	"-6	*8	3	33	5	44	5	*4	1	*4	1
15	4	"-7	*8	3	*3	5	48	6	*4	1	*4	1
15	4	"-8	*8	3	*3	5	49	6	*4	1	*4	1
15	4	"-9	*8	3	*3	5	50	6	*4	1	*4	1
15	4	"-10	*8	3	*3	5	85	6	*4	1	*4	1

II.7.2.

RESULTS FOR CONDITIONAL UPDATING AND FIXED APPROXIMATION

FN	N	C	AB		ABU			ABF			DB		DBU		DBF		
			MS	MF	MS	MF	MJ	MS	MF	MJ	MS	MF	MS	MF	MS	MF	MJ
4	13	10	12	17	12	17	11	13	18	10	13	187	12	160	13	148	10
4	13	"2	77	*09	77	*09	77	77	*09	77	13	*09	60	*09	60	*09	60
7	2	0.1	3	4	3	4	2	3	4	2	3	10	3	8	3	8	2
7	2	1	4	5	4	5	3	6	7	2	4	13	4	11	6	11	2
7	2	5	5	6	5	6	4	7	8	3	5	16	5	14	7	14	3
7	2	10	5	6	6	7	4	6	7	4	5	16	6	15	6	15	4
7	2	50	6	7	7	8	5	8	9	5	6	17	7	18	8	19	5
7	2	"2	7	8	7	8	6	10	11	5	7	22	7	20	10	21	5
7	2	"3	9	10	9	10	8	10	11	7	9	28	9	26	10	25	7
7	2	"4	10	11	11	12	9	11	12	9	10	31	11	30	11	30	9
7	2	"5	12	13	12	13	11	15	16	10	12	37	12	35	15	36	10
7	2	"8	17	18	17	18	16	20	21	15	17	52	17	50	20	51	15
8	2	-	4	5	5	6	3	5	6	2	4	13	5	12	5	10	2
8	13	-	4	5	4	5	3	5	6	2	4	57	4	44	5	32	2
8	24	-	4	5	4	5	3	4	5	3	4	101	4	77	4	77	3
8	35	-	4	5	4	5	3	4	5	3	4	145	4	110	4	110	3
8	46	-	4	5	4	5	3	4	5	3	4	189	4	143	4	143	3

PRECISION TESTS FOR CONDITIONAL UPDATING AND FIXED APPROXIMATION

FN	N	AB		ABU		ABF		DB		DBU		DBF	
		MS	P	MS	P	MS	P	MS	P	MS	P	MS	P
1	5	6	6	6	6	6	4	6	6	6	6	6	4
		7	12	7	9	9	8	7	12	7	9	9	8
		7	12	8	13	12	12	7	12	8	15	12	12
1	6	9	5	9	5	9	5	9	5	9	5	9	5
		10	8	11	10	11	8	10	8	11	10	11	8
		11	15	12	14	13	11	11	15	12	14	13	11
1	7	5	4	6	5	6	4	5	4	6	5	6	4
		6	9	7	11	11	8	6	9	7	11	11	8
		7	13	8	15	16	12	7	13	8	13	16	12
1	8	8	4	8	4	8	4	8	4	8	4	8	4
		9	7	9	7	10	7	9	7	9	7	10	7
		10	13	11	13	13	11	10	13	11	13	13	11
1	9	4	3	4	3	4	3	4	3	4	3	4	3
		6	13	6	8	7	7	6	14	6	8	7	7
		6	13	7	14	10	12	6	14	7	13	10	12
1	10	9	4	9	4	9	4	9	4	9	4	9	4
		11	12	11	9	11	9	11	12	11	9	11	9
		11	12	12	13	12	11	11	12	12	13	12	11
1	13	6	4	6	4	11	3	6	4	6	4	11	3
		7	9	8	11	23	7	7	9	8	11	23	7
		8	12	9	13	35	11	8	12	9	12	35	11

RESULTS FOR CONDITIONAL UPDATING AND FIXED APPROXIMATION

FN	N	C	GAS		GASF			GDS		GDSF			BT		BTM	
			MS	MF	MS	MF	MJ	MS	MF	MS	MF	MJ	MS	MF	MS	MF
4	13	10	10	11	10	11	8	10	141	10	115	8	47	*04	47	*04
4	13	"2	32	33	32	33	31	33	463	33	450	32	47	*04	47	*04
7	2	0.1	4	5	4	5	2	4	13	4	9	2	4	10	3	9
7	2	1	5	6	7	8	2	5	16	7	12	2	5	12	4	12
7	2	5	6	7	8	9	3	6	19	8	15	3	6	15	6	18
7	2	10	6	7	7	8	4	6	19	7	16	4	6	15	6	18
7	2	50	7	8	8	9	5	7	22	8	19	5	8	20	7	20
7	2	"2	8	9	10	11	5	8	25	10	21	5	8	20	8	23
7	2	"3	9	10	10	11	7	9	28	10	25	7	9	22	9	25
7	2	"4	11	12	11	12	9	11	34	11	30	9	12	30	11	30
7	2	"5	12	13	15	16	10	12	37	15	36	10	13	32	12	32
7	2	"8	17	18	20	21	15	17	52	20	51	15	21	52	18	49
8	2	-	5	6	6	7	2	5	16	6	11	2	5	13	4	12
8	13	-	5	6	5	6	3	5	71	5	45	3	5	40	4	42
8	24	-	5	6	5	6	3	5	126	5	78	3	5	68	4	68
8	35	-	5	6	5	6	3	5	181	5	111	3	5	95	4	96
8	46	-	5	6	5	6	3	5	236	5	144	3	5	122	4	122

PRECISION TESTS FOR CONDITIONAL UPDATING AND FIXED APPROXIMATION

FN	N	GAS		GASF		GDS		GDSF	
		MS	P	MS	P	MS	P	MS	P
1	5	17	7	17	5	17	7	17	5
		18	13	20	10	18	13	20	10
		19	13	22	13	19	13	22	13
1	6	59	8	66	2	27	5	43	2
		60	13	100	*04	29	15	100	*04
		61	14	100	*04	29	15	100	*04
1	7	33	9	33	7	31	8	31	6
		34	13	35	11	32	13	33	9
		35	13	37	13	33	13	35	13
1	8	41	*03	74	*04	47	6	47	6
		41	*03	74	*04	49	13	49	11
		41	*03	74	*04	49	13	51	13
1	9	51	9	55	4	50	8	50	4
		52	13	66	7	51	13	54	8
		53	15	67	*04	52	15	58	13
1	10	60	*04	60	*04	60	*04	60	*04
		60	*04	60	*04	60	*04	60	*04
		60	*04	60	*04	60	*04	60	*04
1	13	45	*04	45	*04	47	*04	47	*04
		45	*04	45	*04	47	*04	47	*04
		45	*04	45	*04	47	*04	47	*04

II.7.3.

RESULTS OF SCALING TESTS

FN	N	C	AB		ABIS1		ABIS2		SCAB	
			MS	MF	MS	MF	MS	MF	MS	MF
1	24	-	2	*09	2	*01	2	*01	1	*08
2	3	10	4	5	4	5	4	5	4	5
2	3	"2	6	8	6	7	6	7	6	8
2	3	"3	8	10	7	8	7	8	7	8
2	3	"4	9	12	9	11	9	11	8	9
2	3	"5	11	15	10	12	10	12	10	11
2	3	"6	8	*09	8	*09	11	13	11	12
2	3	"7	7	*09	8	*09	13	15	8	*09
2	3	"8	5	*09	8	*08	14	16	8	*09
2	3	"9	5	*09	7	*09	15	17	7	*09
2	3	"10	3	*09	7	*08	17	19	7	*09
2	3	"11	3	*09	6	*08	18	20	7	*09
10	2	1, 1	3	4	3	4	3	4	2	3
10	2	"-3, 1	3	4	3	4	3	4	2	3
10	2	"-6, 1	3	4	3	4	3	4	2	3
10	2	"-9, 1	3	4	2	*09	2	*09	2	3
10	2	"-14, 1	2	*08	2	*08	2	*08	2	3
10	2	1, "-3	4	5	4	5	4	5	3	4
10	2	1, "-6	3	*08	3	*07	4	5	2	3
10	2	1, "-9	2	*08	2	*08	4	6	2	3
10	2	1, "-14	2	*05	2	*05	2	*05	2	3
10	13	1, 1	3	4	3	4	3	4	3	4
10	13	"-3, 1	3	4	3	4	3	4	3	4
10	13	"-6, 1	3	4	3	4	3	4	3	4
10	13	"-9, 1	3	4	2	*09	2	*09	3	4
10	13	"-14, 1	2	*08	2	*08	2	*08	3	4
10	13	1, "-3	4	5	4	5	4	5	3	4
10	13	1, "-6	3	*09	4	5	4	5	2	3
10	13	1, "-9	2	*08	2	*08	3	*02	3	4
10	13	1, "-14	2	*08	2	*08	2	*08	3	4
11	2	1, 1	4	5	4	5	4	5	3	4
11	2	"-3, 1	3	4	3	4	3	4	3	4
11	2	"-6, 1	3	4	3	4	3	4	3	4
11	2	"-9, 1	3	*08	3	*09	3	*09	3	4
11	2	"-14, 1	1	*05	1	*05	1	*05	3	4
11	2	1, "-3	8	10	13	31	7	10	8	10
11	2	1, "-6	10	*09	10	*09	17	28	2	*09
11	2	1, "-9	2	*09	2	*09	2	*07	2	*08
11	2	1, "-14	2	*09	2	*01	2	*01	2	*08
11	13	1, 1	4	5	4	5	4	5	3	4
11	13	"-3, 1	4	5	4	5	4	5	3	4
11	13	"-6, 1	4	5	3	*09	3	*09	3	4
11	13	"-9, 1	4	5	3	*09	3	*09	3	4
11	13	"-14, 1	2	*08	2	*08	2	*08	3	4
11	13	1, "-3	6	7	2	*02	2	*08	5	6
11	13	1, "-6	7	*09	2	*04	14	23	12	21
11	13	1, "-9	2	*09	2	*04	2	*07	2	*09
11	13	1, "-14	2	*09	2	*01	2	*01	2	*08
16	2	1	5	7	5	7	5	7	5	7
16	2	10	10	11	10	11	2	*02	10	11
16	2	"2	8	*09	10	*09	2	*02	7	*09
16	2	"3	9	*09	11	*09	2	*02	7	*09
16	2	"4	10	*09	12	*09	2	*02	4	*09
16	2	"5	10	*09	13	*09	2	*02	2	*09

FN	N	C	DB		DBIS1		DBIS2		SCDB	
			MS	MF	MS	MF	MS	MF	MS	MF
1	24	-	1	*08	2	*01	2	*01	1	*08
2	3	10	4	17	4	17	4	17	4	17
2	3	"2	6	26	6	25	6	25	6	26
2	3	"3	8	34	7	29	7	29	7	29
2	3	"4	9	39	9	38	9	38	8	33
2	3	"5	8	*08	10	42	10	42	10	41
2	3	"6	6	*08	8	*09	11	46	8	*08
2	3	"7	5	*08	8	*09	13	54	8	*08
2	3	"8	5	*08	7	*09	14	58	7	*08
2	3	"9	4	*08	7	*09	15	62	7	*08
2	3	"10	3	*08	7	*09	16	66	7	*08
2	3	"11	2	*08	7	*09	17	70	6	*08
10	2	1, 1	3	10	3	10	3	10	2	7
10	2	"-3, 1	3	10	3	10	3	10	2	7
10	2	"-6, 1	2	7	2	7	2	7	2	7
10	2	"-9, 1	1	*08	2	*09	2	*09	2	7
10	2	"-14, 1	1	*08	2	*05	2	*05	2	7
10	2	1, "-3	4	13	4	13	4	13	3	10
10	2	1, "-6	2	*08	3	*09	5	16	5	17
10	2	1, "-9	1	*08	11	*02	3	*02	2	*08
10	2	1, "-14	1	*08	1	*05	1	*05	1	*05
10	13	1, 1	3	43	3	43	3	43	3	43
10	13	"-3, 1	3	43	3	43	3	43	3	43
10	13	"-6, 1	3	43	3	43	3	43	3	43
10	13	"-9, 1	1	*08	3	*07	3	*09	3	43
10	13	"-14, 1	1	*08	8	*02	8	*02	3	43
10	13	1, "-3	4	57	4	57	4	57	3	43
10	13	1, "-6	2	*08	3	*09	5	71	3	43
10	13	1, "-9	1	*08	6	*02	3	*02	3	*08
10	13	1, "-14	1	*05	5	*01	1	*05	1	*05
11	2	1, 1	4	13	4	13	4	13	3	10
11	2	"-3, 1	3	10	3	10	3	10	3	10
11	2	"-6, 1	4	13	4	13	4	13	3	10
11	2	"-9, 1	1	*08	3	*09	3	*09	3	10
11	2	"-14, 1	1	*05	1	*05	1	*05	3	10
11	2	1, "-3	8	26	13	57	7	24	8	26
11	2	1, "-6	2	*08	10	*09	17	62	2	*08
11	2	1, "-9	1	*08	12	*09	29	113	2	*08
11	2	1, "-14	1	*05	1	*05	1	*05	1	*05
11	13	1, 1	4	57	4	57	4	57	3	43
11	13	"-3, 1	4	57	4	57	4	57	3	43
11	13	"-6, 1	4	57	4	57	4	57	3	43
11	13	"-9, 1	1	*08	3	*09	3	*09	3	43
11	13	"-14, 1	1	*08	2	*02	9	*02	3	43
11	13	1, "-3	6	85	2	*02	2	*02	5	71
11	13	1, "-6	2	*08	2	*02	14	205	2	*08
11	13	1, "-9	1	*08	4	*09	17	256	2	*05
11	13	1, "-14	1	*05	1	*05	1	*05	1	*05
16	2	1	5	17	5	17	5	17	5	17
16	2	10	10	31	10	31	2	*02	10	31
16	2	"2	8	*08	10	*09	2	*02	7	*09
16	2	"3	9	*08	11	*09	2	*02	7	*09
16	2	"4	9	*09	12	*09	2	*02	2	*08
16	2	"5	7	*08	11	*09	2	*02	2	*08

FN	N	C	SCGAS		GAS		SCGDS		GDS		SCU2S		U2S	
			MS	MF	MS	MF	MS	MF	MS	MF	MS	MF	MS	MF
1	24	-	1	*11	1	*11	25	*04	5	126	2	*11	2	*11
2	3	10	5	6	5	6	5	21	5	21	8	12	8	12
2	3	"2	7	8	7	8	7	29	7	29	17	21	17	21
2	3	"3	8	9	8	9	8	33	8	33	199	*04	199	*04
2	3	"4	10	11	9	10	10	41	10	41	199	*04	94	*11
2	3	"5	11	12	11	12	11	45	11	45	3	*11	3	*11
2	3	"6	12	13	12	13	12	49	12	49	199	*04	4	*11
2	3	"7	13	14	13	14	13	53	14	57	4	*11	5	*11
2	3	"8	14	15	14	15	14	57	23	*07	199	*04	199	*04
2	3	"9	16	17	16	17	16	65	63	*03	163	*11	199	*04
2	3	"10	17	18	17	18	17	69	27	*03	4	*11	4	*11
2	3	"11	18	19	18	19	19	*03	100	*04	4	*11	4	*11
10	2	1, 1	3	4	3	4	3	10	3	10	4	7	4	7
10	2	"-3, 1	3	4	3	4	3	10	3	10	3	6	4	7
10	2	"-6, 1	3	4	3	4	3	10	3	10	3	6	4	7
10	2	"-9, 1	3	4	3	4	3	10	3	10	3	6	4	7
10	2	"-14, 1	3	4	3	4	3	10	3	10	3	6	4	7
10	2	1, "-3	4	5	5	6	4	13	5	16	5	8	8	11
10	2	1, "-6	3	4	5	6	5	16	6	19	4	7	8	11
10	2	1, "-9	3	4	99	*04	4	13	3	10	4	7	199	*04
10	2	1, "-14	3	4	3	4	3	10	3	10	4	7	199	*04
10	13	1, 1	3	4	3	4	3	43	3	43	5	19	5	19
10	13	"-3, 1	3	4	3	4	3	43	3	43	4	18	5	19
10	13	"-6, 1	3	4	3	4	3	43	6	*03	4	18	5	19
10	13	"-9, 1	3	4	3	4	3	43	3	43	4	18	5	19
10	13	"-14, 1	3	4	3	4	3	43	3	43	4	18	5	19
10	13	1, "-3	4	5	5	6	4	57	5	71	5	19	7	21
10	13	1, "-6	3	4	5	6	3	43	6	*03	4	18	10	24
10	13	1, "-9	3	4	5	*03	3	43	3	43	4	18	17	31
10	13	1, "-14	3	4	3	4	3	43	3	43	1	*11	91	*04
11	2	1, 1	4	5	4	5	4	13	4	13	5	8	6	9
11	2	"-3, 1	4	5	4	5	4	13	4	13	5	8	5	8
11	2	"-6, 1	4	5	4	5	4	13	5	16	5	8	5	8
11	2	"-9, 1	4	5	4	5	4	13	3	10	5	8	5	8
11	2	"-14, 1	4	5	3	4	4	13	3	10	5	8	7	10
11	2	1, "-3	13	14	13	14	13	40	13	40	3	*11	11	14
11	2	1, "-6	1	*11	1	*11	1	*11	1	*11	6	9	1	*11
11	2	1, "-9	1	*11	1	*11	1	*11	3	10	1	*11	1	*11
11	2	1, "-14	1	*11	3	4	3	10	3	10	6	9	1	*11
11	13	1, 1	4	5	4	5	4	57	4	57	7	21	8	22
11	13	"-3, 1	4	5	4	5	4	57	4	57	7	21	8	22
11	13	"-6, 1	4	5	4	5	4	57	4	57	7	21	8	22
11	13	"-9, 1	4	5	4	5	4	57	4	57	7	21	8	22
11	13	"-14, 1	4	5	4	5	4	57	4	57	7	21	8	22
11	13	1, "-3	6	7	7	8	6	85	7	99	42	56	12	26
11	13	1, "-6	1	*11	1	*11	1	*11	46	*04	1	*11	1	*11
11	13	1, "-9	1	*11	1	*11	1	*11	4	57	1	*11	4	*11
11	13	1, "-14	1	*11	4	5	4	57	4	57	13	*11	1	*11
16	2	1	1	*11	1	*11	18	55	18	55	16	19	199	*04
16	2	10	11	12	11	12	11	34	11	34	35	38	35	38
16	2	"2	99	*04	18	19	14	*03	17	*03	199	*04	199	*04
16	2	"3	16	*03	99	*04	12	*07	18	*03	199	*04	199	*04
16	2	"4	15	16	23	*03	9	*07	21	*03	199	*04	199	*04
16	2	"5	13	*03	25	*03	3	*07	16	*07	199	*04	199	*04

II.7.4.

RESULTS FOR PROBLEM 1 WHEN REDUCED TO A ONE-DIMENSIONAL NONLINEAR PROBLEM (NO ANALYTIC DERIVATIVES USED)

	DB WITH REDUCTION	DB	
N	MF1	MS	MF
2	11	2	7
3	11	7	30
4	11	9	56
5	11	8	55
6	11	11	83
7	11	7	65
8	11	10	103
9	11	6	72
10	11	12	143
13	11		*
24	11		*

In this table MF1 denotes the number of evaluations of the nonlinear function component.

RESULTS FOR PROBLEM 1

N	SNOLEQJ(S)				SNOLEQJ				SNOLEQ(S)			SNOLEQ			SCU2S		U2S		BW	
	MS	MSV	MF	MJ	MS	MSV	MF	MJ	MS	MSV	MF	MS	MSV	MF	MS	MF	MS	MF	MS	MF
2	1	0	2	1	1	0	2	1	2	0	7	2	0	7	1	4	2	5	2	5
3	6	0	8	5	6	0	8	5	6	0	23	6	0	23	12	16	11	15	7	21
4	9	0	20	7	9	0	20	7	9	0	48	9	0	48	11	16	9	14	15	52
5	7	0	14	6	7	0	14	6	7	0	44	7	0	44	11	17	10	16	19	76
6	43	40	*04	43	11	0	17	9	43	40	*04	11	0	71	15	22	11	18	60	270
7	7	0	17	6	7	0	16	5	7	0	59	7	0	51	170	*04	199	*04	34	170
8	42	40	*04	42	9	0	22	9	42	40	*04	9	0	94	2	*11	199	*04	66	363
9	6	0	18	4	6	0	18	4	6	0	54	6	0	54	2	*11	2	*11	55	330
10	42	40	*04	42	11	0	22	10	42	40	*04	11	0	122	119	*04	2	*11	61	*04
13	6	0	26	5	8	0	26	6	6	0	91	8	0	104	3	*11	2	*11	8	64
24	3	1	*11	3	3	1	*11	3	6	4	199	41	40	*04	2	*11	1	*11	9	121
35	3	1	*11	3	41	40	*04	41	5	5	181	6	5	181	2	*11	2	*11	9	171

RESULTS FOR PROBLEM 2, N = 3

C	SNOLEQJ(S)				SNOLEQJ				SNOLEQ(S)			SNOLEQ			SCU2S		U2S		BW	
	MS	MSV	MF	MJ	MS	MSV	MF	MJ	MS	MSV	MF	MS	MSV	MF	MS	MF	MS	MF	MS	MF
1"01	5	0	6	3	5	0	6	3	5	0	15	5	0	15	8	12	8	12	5	15
1"02	6	0	8	6	6	0	8	6	6	0	26	6	0	26	17	21	17	21	7	21
1"03	8	0	10	8	8	0	10	8	8	0	34	8	0	34	199	*04	199	*04	8	24
1"04	8	0	9	8	9	0	12	9	8	0	33	9	0	39	199	*04	94	*11	11	*05
1"05	10	0	11	10	11	0	15	11	10	0	41	12	4	49	3	*11	3	*11	10	*05
1"06	11	0	12	11	13	5	17	13	11	0	42	13	7	53	199	*04	4	*11	12	*05
1"07	12	0	13	11	15	8	20	15	12	0	46	15	10	62	4	*11	5	*11	14	*05
1"08	14	0	15	12	16	11	21	16	14	0	51	17	12	*07	199	*04	199	*04	13	*05
1"09	15	0	17	14	17	12	23	17	15	0	59	44	40	*04	163	*11	199	*04	12	*05
1"10	16	0	17	15	18	15	24	18	16	0	62	43	40	*04	4	*11	4	*11	16	*05
1"11	17	0	18	16	20	17	27	20	20	6	*07	14	12	*04	4	*11	4	*11	20	*05

RESULTS FOR PROBLEM 4

N	C	SNOLEQJ(S)				SNOLEQJ				SNOLEQ(S)			SNOLEQ			SCU2S		U2S		BW	
		MS	MSV	MF	MJ	MS	MSV	MF	MJ	MS	MSV	MF	MS	MSV	MF	MS	MF	MS	MF	MS	MF
2	1"01	30	0	99	29	17	0	32	16	30	0	157	17	0	64	199	*04	151	154	13	32
2	1"02	8	6	18	8	8	6	18	8	8	6	32	8	6	32	199	*04	190	193	17	42
2	1"03	7	5	21	7	7	5	21	7	9	7	39	8	6	36	199	*04	199	*04	28	70
2	1"04	7	5	24	7	7	5	24	7	12	10	50	10	8	45	199	*04	199	*04	100	*04
2	1"05	6	4	22	6	6	4	27	6	42	40	*04	11	9	*07	199	*04	199	*04	100	*04
2	1"06	6	4	21	6	6	4	30	6	42	40	*04	9	7	*07	199	*04	199	*04	100	*04
2	1"07	7	5	20	7	6	4	32	6	12	10	*07	13	11	*07	199	*04	199	*04	100	*04
13	1"01	62	22	330	62	12	0	17	11	63	23	1137	12	0	160	91	*04	91	*04	9	72
13	1"02	51	21	196	51	65	25	352	65	37	14	598	80	40	*04	91	*04	91	*04	47	*04
13	1"03	46	39	59	46	36	28	46	36	45	38	630	37	29	515	91	*04	91	*04	15	120
13	1"04	17	10	25	17	48	40	*04	48	43	36	*07	43	35	*07	91	*04	91	*04	20	160
13	1"05	49	40	*04	49	49	40	*04	48	18	11	243	49	40	*04	91	*04	91	*04	47	*04
13	1"06	48	40	*04	48	50	40	*04	48	23	21	*07	31	21	*07	49	63	91	*04	20	*01
13	1"07	48	40	*04	48	25	15	41	23	14	12	*07	21	11	*07	66	80	91	*04	47	*04

RESULTS FOR PROBLEM 5

N	SNOLEQJ(S)				SNOLEQJ				SNOLEQ(S)			SNOLEQ			SCU2S		U2S		BW	
	MS	MSV	MF	MJ	MS	MSV	MF	MJ	MS	MSV	MF	MS	MSV	MF	MS	MF	MS	MF	MS	MF
2	2	0	3	2	2	0	3	2	2	0	7	2	0	7	3	6	2	5	3	8
13	3	0	4	2	3	0	4	2	3	0	30	3	0	30	4	18	4	18	3	24
24	3	0	4	2	3	0	4	2	3	0	52	3	0	52	4	29	4	29	4	54
35	3	0	4	2	3	0	4	2	3	0	74	3	0	74	4	40	5	41	4	76
46	3	0	4	2	3	0	4	2	3	0	96	3	0	96	4	51	4	51	4	98

RESULTS FOR PROBLEM 7, N = 2

C	SNOLEQJ(S)				SNOLEQJ				SNOLEQ(S)			SNOLEQ			SCU2S		U2S		BW	
	MS	MSV	MF	MJ	MS	MSV	MF	MJ	MS	MSV	MF	MS	MSV	MF	MS	MF	MS	MF	MS	MF
1"-1	3	0	4	2	3	0	4	2	3	0	8	3	0	8	5	8	5	8	4	10
1"+0	4	0	5	3	4	0	5	3	4	0	11	4	0	11	7	10	7	10	5	12
5"+0	5	0	6	4	5	0	6	4	5	0	14	5	0	14	9	12	9	12	6	15
1"+1	6	0	7	4	6	0	7	4	6	0	15	6	0	15	10	13	10	13	6	15
5"+1	7	0	8	5	7	0	8	5	7	0	18	7	0	18	12	15	13	16	7	17
1"+2	7	0	8	6	7	0	8	6	7	0	20	7	0	20	12	15	14	17	8	20
1"+3	9	0	10	8	9	0	10	8	9	0	26	9	0	26	12	15	15	18	9	22
1"+4	11	0	12	9	11	0	12	9	11	0	30	11	0	30	14	17	17	20	12	30
1"+5	12	0	13	11	12	0	13	11	12	0	35	12	0	35	17	20	20	23	13	32
1"+8	17	0	18	16	17	0	18	16	18	1	51	17	0	50	23	26	24	27	21	52

RESULTS FOR ORDER 2

FN	C	SNOLEQJ(S)				SNOLEQJ				SNOLEQ(S)			SNOLEQ			SCU2S		U2S		BW	
		MS	MSV	MF	MJ	MS	MSV	MF	MJ	MS	MSV	MF	MS	MSV	MF	MS	MF	MS	MF	MS	MF
1	-	1	0	2	1	1	0	2	1	2	0	7	2	0	7	1	4	2	5	2	5
2	1"1	5	0	6	4	5	0	7	5	5	0	14	8	0	26	13	16	24	27	6	15
3	-	2	0	3	2	2	0	3	2	3	0	10	3	0	10	3	6	4	7	2	5
4	1"1	30	0	99	29	17	0	32	16	30	0	157	17	0	64	199	*04	151	154	13	32
4	1"4	7	5	24	7	7	5	24	7	12	10	50	10	8	45	199	*04	199	*04	20	50
4	1"7	7	5	20	7	6	4	32	6	12	10	*07	13	11	*07	199	*04	199	*04	100	*04
5	-	2	0	3	2	2	0	3	2	2	0	7	2	0	7	3	6	2	5	3	8
6	-	6	0	7	4	6	0	7	4	6	0	15	6	0	15	7	10	7	10	6	15
7	1"1	6	0	7	4	6	0	7	4	6	0	15	6	0	15	10	13	10	13	6	15
7	1"4	11	0	12	9	11	0	12	9	11	0	30	11	0	30	14	17	17	20	12	30
8	-	5	0	6	3	5	0	6	3	5	0	12	5	0	12	6	9	6	9	5	13
9	-	5	0	6	3	5	0	6	3	5	0	12	5	0	12	6	9	6	9	5	13
10	1"+00,1	3	0	4	2	3	0	4	2	3	0	8	3	0	8	4	7	4	7	3	8
10	1"-03,1	3	0	4	2	3	0	4	2	3	0	8	3	0	8	4	7	4	7	3	8
10	1"-06,1	3	0	4	2	3	0	4	2	3	0	8	2	0	7	4	7	4	7	3	8
10	1"-09,1	3	0	4	2	3	0	4	2	3	0	8	3	2	8	4	7	4	7	3	8
10	1"-14,1	3	0	4	2	3	2	4	3	3	0	8	3	2	8	4	7	4	7	3	8
10	1,1"-03	5	0	6	2	4	0	5	4	5	0	10	4	0	13	5	8	8	11	4	10
10	1,1"-06	4	0	5	2	5	2	6	5	8	3	24	7	5	20	4	7	8	11	4	10
10	1,1"-09	3	0	4	2	13	11	14	13	3	0	8	3	2	8	4	7	199	*04	3	8
10	1,1"-14	3	0	4	2	3	2	4	3	3	3	10	4	3	10	4	7	199	*04	3	8
11	-	4	0	5	3	4	0	5	3	4	0	11	4	0	11	5	8	6	9	5	13
12	-	2	0	3	2	2	0	3	2	2	0	7	2	0	7	5	8	4	7	3	8
13	-	4	0	5	3	4	0	5	3	4	0	11	4	0	11	7	10	6	9	5	13
14	-	4	0	5	3	4	0	5	3	4	0	11	4	0	11	5	8	6	9	4	10

RESULTS FOR ORDER 13

FN	C	SNOLEQJ(S)				SNOLEQJ				SNOLEQ(S)			SNOLEQ			SCU2S		U2S		BW	
		MS	MSV	MF	MJ	MS	MSV	MF	MJ	MS	MSV	MF	MS	MSV	MF	MS	MF	MS	MF	MS	MF
1	-	6	0	26	5	8	0	26	6	6	0	91	8	0	104	3	*11	2	*11	8	64
2	1"1	49	40	*04	49	49	40	*04	49	49	40	*04	49	40	*04	91	*04	91	*04	4	*05
3	-	9	7	*03	9	9	7	*03	9	11	8	*11	9	7	*11	3	*11	3	*11	2	16
4	1"1	62	22	330	62	12	0	17	11	63	23	1137	12	0	160	91	*04	91	*04	9	72
4	1"4	17	10	25	17	48	40	*04	48	43	36	*07	43	35	*07	91	*04	91	*04	13	104
4	1"7	48	40	*04	48	25	15	41	23	14	12	*07	21	11	*07	66	80	91	*04	47	*04
5	-	3	0	4	2	3	0	4	2	3	0	30	3	0	30	4	18	4	18	3	24
6	-	6	0	7	3	6	0	7	3	6	0	46	6	0	46	10	24	13	27	6	48
7	1"1	6	0	7	4	6	0	7	4	6	0	59	6	0	59	17	31	17	31	6	48
7	1"4	11	0	12	9	11	0	12	9	11	0	129	11	0	129	14	28	22	36	12	96
8	-	4	0	5	3	4	0	5	3	4	0	44	4	0	44	6	20	6	20	4	32
9	-	5	0	6	2	5	0	6	2	5	0	32	5	0	32	6	20	6	20	4	32
10	1"+00,1	3	0	4	2	3	0	4	2	3	0	30	3	0	30	4	18	5	19	4	32
10	1"-03,1	3	0	4	2	3	0	4	2	3	0	30	3	0	30	5	19	5	19	4	32
10	1"-06,1	3	0	4	2	3	0	4	2	3	0	30	3	0	30	4	18	5	19	4	32
10	1"-09,1	3	0	4	2	3	0	4	2	3	0	30	4	3	44	5	19	5	19	4	32
10	1"-14,1	3	0	4	2	4	3	5	4	3	0	30	4	3	44	5	19	5	19	4	32
10	1,1"-03	5	0	6	2	4	0	5	4	5	0	32	4	0	57	8	22	7	21	5	40
10	1,1"-06	4	0	5	2	5	2	6	5	6	0	46	42	40	*04	57	*11	10	24	6	48
10	1,1"-09	3	0	4	2	5	3	6	5	4	0	44	3	2	30	78	*11	17	31	4	32
10	1,1"-14	3	0	4	2	3	2	4	3	3	3	43	4	3	43	85	*11	91	*04	4	32
11	-	5	0	6	2	5	0	6	2	5	0	32	5	0	32	7	21	8	22	5	40
12	-	2	0	3	2	2	0	3	2	2	0	29	2	0	29	11	25	4	18	3	24
13	-	6	0	7	4	6	0	7	4	6	0	59	6	0	59	11	25	12	26	5	40
14	-	4	0	5	2	4	0	5	2	4	0	31	4	0	31	5	19	7	21	5	40

RESULTS FOR ORDER 24

FN	C	SNOLEQJ(S)				SNOLEQJ				SNOLEQ(S)			SNOLEQ			SCU2S		U2S		BW	
		MS	MSV	MF	MJ	MS	MSV	MF	MJ	MS	MSV	MF	MS	MSV	MF	MS	MF	MS	MF	MS	MF
1	-	3	1	*11	3	3	1	*11	3	6	4	199	41	40	*04	2	*11	1	*11	9	121
2	1"1	4	1	*11	3	3	1	*11	3	4	1	*11	3	1	*11	2	*11	2	*11	2	*11
3	-	10	8	*03	10	4	2	*11	4	12	9	*11	42	40	*04	3	*11	3	*11	2	27
4	1"1	33	19	70	33	9	0	10	8	48	40	*04	9	0	202	49	*04	19	*04	9	122
4	1"4	17	10	26	16	17	10	26	16	15	8	335	17	10	386	49	*04	49	*04	20	322
4	1"7	20	13	21	18	20	11	38	17	10	5	*07	15	6	*07	37	*11	49	*04	26	*04
5	-	3	0	4	2	3	0	4	2	3	0	52	3	0	52	4	29	4	29	4	54
6	-	6	0	7	3	6	0	7	3	6	0	79	6	0	79	10	35	13	38	6	81
7	1"1	6	0	7	4	6	0	7	4	6	0	103	6	0	103	17	42	17	42	6	81
7	1"4	11	0	12	9	11	0	12	9	11	0	228	11	0	228	14	39	22	47	12	162
8	-	4	0	5	3	4	0	5	3	4	0	77	4	0	77	6	31	6	31	4	54
9	-	5	0	6	2	5	0	6	2	5	0	54	5	0	54	7	32	6	31	4	54
10	1"+00,1	4	0	5	2	4	0	5	2	4	0	53	4	0	53	5	30	6	31	5	68
10	1"-03,1	4	0	5	2	4	0	5	2	4	0	53	4	0	53	5	30	6	31	5	68
10	1"-06,1	4	0	5	2	4	0	5	2	4	0	53	4	0	53	5	30	6	31	5	68
10	1"-09,1	4	0	5	2	4	0	5	2	4	0	53	4	3	77	5	30	6	31	5	68
10	1"-14,1	4	0	5	2	4	3	5	4	4	0	53	4	3	77	5	30	6	31	5	68
10	1,1"-03	4	0	5	4	5	0	6	4	5	3	102	5	0	102	6	31	8	33	5	68
10	1,1"-06	5	0	6	3	3	0	4	3	3	0	76	9	7	*07	5	30	12	37	6	81
10	1,1"-09	3	0	4	3	5	3	6	5	5	1	78	4	3	77	5	30	22	47	5	68
10	1,1"-14	4	0	5	2	4	3	5	4	4	4	101	5	4	101	49	*04	49	*04	5	68
11	-	4	0	5	3	4	0	5	3	4	0	77	4	0	77	7	32	9	34	6	81
12	-	3	0	4	2	3	0	4	2	3	0	52	3	0	52	5	30	4	29	4	54
13	-	5	0	6	4	5	0	6	4	5	0	102	5	0	102	32	57	19	44	7	95
14	-	4	0	5	2	4	0	5	2	4	0	53	4	0	53	5	30	7	32	5	68

RESULTS FOR ORDER 35

FN	C	SNOLEQJ(S)				SNOLEQJ				SNOLEQ(S)			SNOLEQ			SCU2S		U2S		BW	
		MS	MSV	MF	MJ	MS	MSV	MF	MJ	MS	MSV	MF	MS	MSV	MF	MS	MF	MS	MF	MS	MF
1	-	3	1	*11	3	41	40	*04	41	5	5	181	6	5	181	2	*11	2	*11	9	171
2	1"1	5	1	*11	4	3	1	*11	3	44	40	*14	42	40	*04	2	*11	2	*11	2	*11
3	-	10	8	*03	10	4	2	*11	4	7	4	*11	6	4	*11	3	*11	3	*11	2	38
4	1"1	20	0	63	18	29	20	56	29	20	0	693	50	40	*04	33	*04	33	*04	9	171
4	1"4	48	40	*04	48	48	40	*04	48	20	13	689	48	40	*04	33	*04	33	*04	18	*04
4	1"7	31	22	35	29	33	23	51	31	45	40	*07	20	10	*07	33	*04	33	*04	18	*04
5	-	3	0	4	2	3	0	4	2	3	0	74	3	0	74	4	40	5	41	4	76
6	-	6	0	7	3	6	0	7	3	6	0	112	6	0	112	10	46	13	49	6	114
7	1"1	6	0	7	4	6	0	7	4	6	0	147	6	0	147	17	53	17	53	6	114
7	1"4	11	0	12	9	11	0	12	9	11	0	327	11	0	327	14	50	22	58	12	228
8	-	4	0	5	3	4	0	5	3	4	0	110	4	0	110	6	42	6	42	4	76
9	-	5	0	6	2	5	0	6	2	5	0	76	5	0	76	6	42	6	42	4	76
10	1"+00,1	4	0	5	2	4	0	5	2	4	0	75	4	0	75	6	42	6	42	5	95
10	1"-03,1	4	0	5	2	4	0	5	2	4	0	75	4	0	75	6	42	6	42	5	95
10	1"-06,1	4	0	5	2	4	0	5	2	4	0	75	4	0	75	6	42	6	42	5	95
10	1"-09,1	4	0	5	2	4	0	5	2	4	0	75	4	0	75	6	42	6	42	5	95
10	1"-14,1	4	0	5	2	4	3	5	4	4	0	75	4	0	75	6	42	6	42	5	95
10	1,1"-03	4	0	5	4	4	0	5	4	4	0	145	4	0	145	9	45	11	47	18	*04
10	1,1"-06	3	0	4	3	3	0	4	3	5	0	146	5	0	146	6	42	15	51	13	247
10	1,1"-09	3	0	4	3	4	2	5	4	4	0	110	4	0	110	6	42	19	55	8	152
10	1,1"-14	4	0	5	3	4	3	5	4	3	3	109	3	3	109	33	*04	33	*04	7	133
11	-	5	0	6	2	5	0	6	2	5	0	76	5	0	76	7	43	9	45	6	114
12	-	3	0	4	2	3	0	4	2	3	0	74	3	0	74	5	41	5	41	4	76
13	-	51	40	*04	51	51	40	*04	51	27	14	*07	27	14	*07	7	*11	33	*04	18	*04
14	-	5	0	6	5	5	0	6	5	5	0	181	5	0	181	8	44	10	46	7	133

RESULTS FOR ORDER 46

FN	C	SNOLEQJ(S)				SNOLEQJ				SNOLEQ(S)			SNOLEQ			SCU2S		U2S		BW	
		MS	MSV	MF	MJ	MS	MSV	MF	MJ	MS	MSV	MF	MS	MSV	MF	MS	MF	MS	MF	MS	MF
1	-	41	40	*04	41	41	40	*04	41	5	5	236	6	5	236	2	*11	2	*11	10	245
2	1"1	4	1	*11	3	3	1	*11	3	5	1	*11	9	6	*07	2	*11	1	*11	2	*11
3	-	4	2	*11	4	4	2	*11	4	10	7	*11	13	10	*11	3	*11	3	*11	2	49
4	1"1	9	0	10	8	9	0	10	8	9	0	378	9	0	378	25	*04	25	*04	9	221
4	1"4	17	10	26	16	17	10	26	16	13	6	526	17	10	716	25	*04	25	*04	14	*04
4	1"7	18	11	19	16	21	12	39	18	10	5	*07	15	6	*07	25	*04	25	*04	14	*04
5	-	3	0	4	2	3	0	4	2	3	0	96	3	0	96	4	51	4	51	4	98
6	-	6	0	7	3	6	0	7	3	6	0	145	6	0	145	10	57	13	60	6	147
7	1"1	6	0	7	4	6	0	7	4	6	0	191	6	0	191	17	64	17	64	6	147
7	1"4	11	0	12	9	11	0	12	9	11	0	426	11	0	426	14	61	22	69	12	294
8	-	4	0	5	3	4	0	5	3	4	0	143	4	0	143	6	53	6	53	4	98
9	-	5	0	6	2	5	0	6	2	5	0	98	5	0	98	7	54	6	53	4	98
10	1"+00,1	4	0	5	3	4	0	5	3	4	0	143	4	0	143	6	53	7	54	6	147
10	1"-03,1	4	0	5	3	4	0	5	3	4	0	143	4	0	143	6	53	7	54	6	147
10	1"-06,1	4	0	5	3	4	0	5	3	4	0	143	4	3	143	6	53	7	54	6	147
10	1"-09,1	4	0	5	3	4	0	5	3	4	0	143	4	3	143	6	53	7	54	6	147
10	1"-14,1	4	0	5	3	4	3	5	4	4	0	143	4	3	143	6	53	7	54	6	147
10	1,1"-03	7	5	11	7	7	5	11	7	9	7	380	9	7	380	12	59	11	58	5	122
10	1,1"-06	6	0	7	5	5	3	6	5	9	3	*07	8	7	*07	8	55	22	69	7	171
10	1,1"-09	4	0	5	4	4	2	5	4	5	3	190	4	3	143	6	53	25	*04	14	*04
10	1,1"-14	4	0	5	3	4	3	5	4	4	4	189	5	4	189	5	52	25	*04	5	122
11	-	5	0	6	3	6	0	7	3	5	0	144	6	0	145	7	54	10	57	6	147
12	-	3	0	4	2	3	0	4	2	3	0	96	3	0	96	5	52	5	52	4	98
13	-	6	0	10	6	6	0	10	6	6	0	286	6	0	286	17	64	18	65	9	221
14	-	4	0	5	3	5	0	6	2	4	0	143	5	0	98	5	52	5	52	5	122

STELLINGEN

BIJ HET PROEFSCHRIFT

NUMERICAL SOLUTION OF SYSTEMS OF NONLINEAR EQUATIONS

VAN

J.C.P. BUS

16 APRIL 1980

I

De resultaten 3.1, 3.2 en 3.3 in [1] kunnen worden verscherpt door in de standaardvoorwaarden (3.6) I en II te vervangen door:

I. Let $F \in C^1(D)$, with the path-connected component of x_0 in $G_0(A)$ contained in D for some $A \in A_M$. Let $J(x)$ denote the Jacobian matrix.

II. Let $J(x)$ be nonsingular on the path-connected component of x_0 in $G_0(A)$.

[1] P. DEUFLHARD (1974), *A modified Newton method for the solution of ill-conditioned systems of nonlinear equations with application to multiple shooting*, Numer. Math. 22, p. 289-315.

II

Zij M een Newton-achtige methode. Stel dat voor alle $F \in F$, $M(F) = (\Psi_1, \Psi_2)$ en $B_F(x,H) \in L(\mathbb{R}^n)$ voldoen aan

$$\Psi_2(x,H) = (B_F(x,H))^+, \qquad (x,H) \in D_\Psi,$$
$$B_{TF}(x,H) = TB_F(x,H) \ ,$$

voor alle niet-singuliere $T \in L(\mathbb{R}^n)$. Stel verder dat er een $F \in F$ en $(x,H) \in D_\Psi$ bestaat zodat $B_F(x,H)$ singulier is. Dan is M niet affien invariant. (Vgl. lemma 4.29 van bijbehorend proefschrift.).

III

Stelling 1.1 in [2] is onjuist. De stelling geldt indien de "levelset" $G_k(A)$ wordt gedefinieerd zoals in [1], of indien als extra eis wordt gesteld dat D een open verzameling is in $\mathbb{R}^n$.

[2] P. DEUFLHARD (1974), *A relaxation strategy for the modified Newton method*. In: R. Bulirsch, W Oettli and J. Stoer (eds) Conference on Optimization and optimal control, Oberwolfach, Springer.

IV

Zij $F : \mathbb{R}^n \to \mathbb{R}$ een tweemaal differentieerbare functie die naar beneden is begrensd. Stel dat, voor zekere positieve constanten m,N en L $(0 < m \leq M)$ en alle x en y in $\mathbb{R}^n$, de hessiaan $G(x)$ voldoet aan

$$m\|u\|^2 \leq u^T G(x) u \leq M\|u\|^2, \quad \text{voor alle } u \neq 0 \text{ in } \mathbb{R}^n,$$

en

$$\|G(x) - G(y)\| \le L\|x-y\|.$$

Zij $A(U)$ een variabele-metriek-methode zoals gedefinieerd in [3] voor minimalisering van F. Stel dat $A(U)$ zo is dat de staplengtefactor α_k gelijk aan 1 wordt gekozen indien deze keuze aan de voorwaarden in de definitie van $A(U)$ voldoet. Stel dat $A(U)$, met $r \le m/(2M)$ en $c \le 0.5$, een rij punten $\{x_k\}$ genereert voor gegeven startpunt x_0 en symmetrische matrix H_0. Dan convergeert $\{x_k\}$ naar een punt $x^* \in \mathbb{R}^n$ waarvoor F minimaal is en convergentie is superlineair als

$$\lim_{k\to\infty} \frac{\|(H(x_k)-H_k)F'(x_k)\|}{\|F'(x_k)\|} = 0,$$

met $H(x_k) = (G(x_k))^{-1}$ en H_k de variabele-metriek-benadering van $H(x_k)$.

[3] J.C.P.BUS (1975), *On the convergence of a class of variable metric algorithms,* Mathematisch Centrum, NW 16/75, Amsterdam.

V

Aan het criterium van reproduceerbaarheid van numerieke resultaten, zoals geformuleerd in [4], kan niet altijd worden voldaan. Dit is een gevolg van het feit dat de met behulp van een computer verkregen resultaten van een mathematisch programmeringsprobleem in het algemeen afhankelijk zijn van de gevoeligheid van het probleem met betrekking tot zijn parameters.

[4] H.P. CROWDER, R.S. DEMBO & J.M. MULVEY (1978), *Guidelines for reporting computational experiments in mathematical programming.* In: H.J. Greenberg (ed.), Design and implementation of optimization software, Sijthoff and Noordhoff.

VI

Ondanks enkele uitgebreide studies ter evaluatie van programmatuur voor niet-lineaire programmering ([5], [6], [7]), bestaat nog steeds onduidelijkheid over de bruikbaarheid van deze programmatuur bij gebruikers. De reden hiervoor moet voor een groot deel worden gezocht in het feit dat

- het veelal onduidelijk is in welke mate de experimenteel verkregen resultaten afhangen van de algoritme die ten grondslag ligt aan de programmatuur en in welke mate van de specifieke implementatie die wordt getest;
- selectie van programmatuur in belangrijke mate afhangt van subjectieve criteria van de gebruiker van die programmatuur.

[4] A.R. COLVILLE (1968), *A comparative study of nonlinear programming codes*, Report 320-2949, IBM New York.

[5] E. SANDGREN (1978), *The utility of nonlinear programming algorithms*, Thesis, Purdue University.

[6] K. SCHITTKOWSKI (1979), *A numerical comparison of optimization programs using randomly generated test problems*. In: L.D. Fosdick (ed.), Performance evaluation of numerical software, North Holland.

VII

Bij de samenstelling van een verzameling testproblemen voor evaluatie van bepaalde programmatuur dient kennis van de structuur en de moeilijkheidsgraad van de te kiezen problemen voorop te staan. Noch de suggestie (zie bijv. [5]) dat een probleem uit de dagelijkse praktijk altijd een goed testprobleem is, noch het idee (zie bijv. [6]) dat uitbreiding van de verzameling testproblemen altijd tot een betere testverzameling leidt, is juist.

VIII

De waardering voor een computerprogramma wordt in belangrijke mate bepaald door de kwaliteit van zijn documentatie (zie [7]).

[7] H.P. CROWDER & P.B. SAUNDERS, *Results of a survey on MP performance indicators*, Committee on Algorithms Newsletter, jan. 1980, Mathematical Programming Society.

IX

Als een niet-publiekrechtelijk orgaan bij het verlenen van subsidie aan een stichting met ten minste 100 werknemers, voorwaarden stelt ten aanzien van het aanstellings-, ontslag- of bevorderingsbeleid van die stichting, dan vormt dit een belemmering voor de goede uitvoering van de wet op de ondernemingsraden.

X

De ontwikkelingen in Nederland ten aanzien van het aanstellingsbeleid van wetenschappelijke onderzoekers maken het afsluiten van arbeidsplaatsen-overeenkomsten (APO's) in deze sector dringend noodzakelijk.