Konrad-Zuse-Zentrum für Informationstechnik Berlin

U. Nowak L. Weimann

A Family of Newton Codes for Systems of Highly Nonlinear Equations

Technical Report TR-91-10 (December 1991)

A Family of Newton Codes for Systems of Highly Nonlinear Equations

U. Nowak L. Weimann

Abstract

This reports presents new codes for the numerical solutiuon of highly nonlinear systems. They realize the most recent variants of affine invariant Newton Techniques due to Deuflhard. The standard method is implemented in the code NLEQ1, whereas the code NLEQ2 contains a rank reduction device additionally. The code NLEQ1S is the sparse version of NLEQ1, i.e. the arising linear systems are solved with sparse matrix techniques. Within the new implementations a common design of the software in view of user interface and internal modularization is realized. Numerical experiments for some rather challenging examples illustrate robustness and efficiency of algorithm and software.

Contents

0. Introduction

The efficient and robust numerical solution of a system of nonlinear equations can be a rather challenging problem — especially in cases where only little a priori information on the solution is available. A quite outstanding method for solving nonlinear equations is the Newton method, an iterative scheme which is known to converge quadratically near the solution, but only, if the initial guess is sufficiently close to the solution. Now, in order to extend the convergence domain of Newton's method, some globalizations are in common use, e.g. damped Newton methods, steepest descend methods and Levenberg-Marquardt methods. Based on the latter techniques, some state-of-the-art software has been developed, e.g. the codes from IMSL, NAG and MINPACK [23]. In contrast to this, the codes presented in this paper are based on the affine invariant damped Newton techniques due to Deuflhard [5, 6, 7]. Within these algorithms the usual local Newton techniques are combined with a special damping strategy in order to create globally convergent Newton schemes. One essential property of these algorithms — as well as of the underlying theory — is their affine invariance. As far as possible, this property of the algorithms is preserved by the codes. Furthermore, the damping strategy and the convergence checks are implemented in a scaling invariant form — a feature which is extremely useful especially for real life applications.

The codes presented here are revised and extended versions of former research codes due to Deuflhard. Within the new codes the most recent theoretical results of [7] are regarded and, furthermore, some new algorithmic variants are realized. The main new features of the implementation are the internal workspace management, an option setting concept and the modular programming. The new interface allows on one hand an easy-to-use application and on the other hand an easy selection and control of special features of the codes. The codes are still considered to be research codes, in the sense, that a variety of algorithmic options may be set by the user and that they are not optimized with respect to performance.

The family of Newton codes which is presented in this paper consists of three different codes. The code NLEQ1 represents the basic version of the global affine invariant Newton algorithm. The code NLEQ2 contains, in addition to the damping strategy of NLEQ1, a special rank reduction device. Finally there is the code NLEQ1S, which is the sparse version of NLEQ1, i.e. sparse matrix techniques are used for the solution of the internally arising linear systems. All these codes belong to the numerical software library CodeLib of the Konrad Zuse Zentrum Berlin, thus they are available for interested users.

The paper is organized as follows. Chapter 1 contains a detailed description of the algorithmic characteristics of the global affine invariant Newton codes. Within Section 1.1 the main features of the basic algorithm are outlined, and the whole algorithm is summarized in Section 1.2. Section 1.3 deals with details and variants of the basic algorithm, e.g. internal scaling, rank reduction and modifications of the damping strategy. Chapter 2 describes the implementation of the basic algorithm and its variants. First, a general overview is given in Section 2.1. Section 2.2 deals with the interfaces of the codes and Section 2.3 describes how to use and adapt the special facilities of the codes. Typical numerical experiments are presented in Chapter 3. First, in Section 3.1 a set of test problems is established. The numerical results of solving a certain basic test set are given in Section 3.2. In order to rate these results some comparisons with another state-of-the-art code (HYBRJ1 from MINPACK [23]) are presented. The results of some special experiments are reported in Section 3.3. Finally, some concluding remarks are made.

1. Global Affine Invariant Newton Techniques

The codes presented in this paper are designed to solve a system of n nonlinear equations in n unknowns:

$$\begin{aligned} f_1(x_1,\ldots,x_n) &= 0 \\ \vdots \qquad & \quad \vdots \\ f_n(x_1,\ldots,x_n) &= 0 \end{aligned} \tag{1.1}$$

or, in short notation:

$$F(x) = 0 \tag{1.2}$$

where

$$\begin{aligned} F : D &\longrightarrow \mathrm{I\!R}^n \\ x = (x_1,\ldots,x_n)^T &\in D \subset \mathrm{I\!R}^n \end{aligned}$$

Usually, some a priori information about the *solution point* x^* of (1.2) is available in form of an *initial guess* x_0. Besides the nonlinearity of F and the dimension n of the system, the quality of this information will strongly affect the computational complexity of solving (1.2) numerically. Taking this into account, the complete problem formulation may be written as:

$$\begin{aligned} &a) \quad F(x) = 0,\ x \in \mathrm{I\!R}^n \\ &b) \quad x_0 \text{ given initial guess.} \end{aligned} \tag{1.3}$$

1.1 Outline of algorithm

Within this Section the basic ideas and considerations, which led to the development of the damped affine invariant Newton methods due to Deuflhard [5, 6, 7], are shortly described. A proper and detailed mathematical formulation of that topic can be found in the recent monograph of Deuflhard [7]. Note that the theoretical advantages of these algorithms — as well as their limitations – show up also in the codes presented in this paper.

1.1.1 Newton methods

First, consider the ordinary Newton method for problem (1.3). Let

$$J(x) := F'(x) = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \dfrac{\partial f_n}{\partial x_1} & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{1.4}$$

denote the Jacobian (n, n)-matrix, assumed to be nonsingular for all $x \in D$. Then, for a given starting point $x_0 \in D$, the *ordinary Newton iteration* reads

$$\begin{aligned} & k = 0, 1, 2, \ldots \\ & a) \quad J(x_k)\Delta x_k = -F(x_k) \\ & b) \quad x_{k+1} = x_k + \Delta x_k \\ & \Delta x_k \ : \ \text{ordinary Newton correction.} \end{aligned} \tag{1.5}$$

This method is known to be *quadratically convergent* near the solution x^*, i.e. only a few iteration steps are necessary to generate a highly accurate numerical solution for (1.3). However, the scheme (1.5) is only *locally convergent*, i.e. the initial guess x_0 must be "close enough" to the solution x^*. To achieve convergence also for "bad" initial guesses x_0, one may globalize (1.5) by introducing a damping parameter λ. With that, (1.5) is extended to a *damped Newton iteration*

$$\begin{aligned} & k = 0, 1, 2, \ldots \\ & a) \quad J(x_k)\Delta x_k = -F(x_k) \\ & b) \quad x_{k+1} = x_k + \lambda_k \Delta x_k \\ & \lambda_k \ : \ \text{damping factor} \quad (0 < \lambda_k \leq 1) \end{aligned} \tag{1.6}$$

Indeed, in order to create an efficient and robust solution method for problem (1.3), this iteration must be combined with an adaptive steplength control. Within such a procedure the damping factors λ_k should be chosen in such a way, that the iterates x_k approach successively the solution x^* — possibly fast. As the "true" convergence criterion

$$\|x_{k+1} - x^*\| < \|x_k - x^*\| \, , \ x_k \neq x^* \tag{1.7}$$

and the associated stopping criterion

$$\|x_{k+1} - x^*\| \leq tol \tag{1.8}$$

are computationally not available, *substitute approach criteria* must be introduced. Usually, such criteria are based on the definition of a so-called *level function* (test function). A widely used level function is given by

$$T(x) := \frac{1}{2}\|F(x)\|_2^2 = \frac{1}{2}F(x)^T F(x) \tag{1.9}$$

and (1.7),(1.8) may be substituted by

$$T(x_{k+1}) < T(x_k) \tag{1.10}$$

$$\sqrt{T(x_{k+1})} \leq tol\ . \tag{1.11}$$

The main objection to this type of criteria is, that the checks of (1.10) and (1.11) are made in the "wrong" space, namely in the space of the residuals. Instead of this, the approach and the quality of the iterates x_k should be checked in the space of the (unknown) solution x^*. This requirement is also a consequence of the general *affine invariance principle*, which is the leading theoretical concept of the Newton techniques due to Deuflhard.

1.1.2 Affine invariance

Let A denote an arbitrary nonsingular (n,n)-matrix, i.e. let $A \in GL(n)$. Then problem (1.2) is equivalent to any problem of the type

$$A \cdot F(x) = 0\,. \tag{1.12}$$

In other words, problem (1.2) is *invariant* under the *affine transformation*

$$F \longrightarrow G := AF\,,\ A \in GL(n)\,. \tag{1.13}$$

Equivalently, problem (1.2) is said to be *affine invariant.* This property is shared by the ordinary Newton method. Application of method (1.5) to the transformed problem (1.13) yields corrections

$$\Delta x_k^G = -G'(x_k)^{-1}G(x_k) = -F'(x_k)^{-1}A^{-1}AF(x_k) = -J(x_k)^{-1}F(x_k) = \Delta x_k^F\ .$$

Hence, starting with the same initial guess $x_0^G = x_0^F$, an identical iteration sequence $\{x_k^G\}_{k=0,1,2,\ldots} = \{x_k^F\}_{k=0,1,2,\ldots}$ is generated. Consequently, for a damped Newton method of type (1.6) the affine invariance property of problem (1.3) should carry over. Thus, an adaptive steplength control algorithm for (1.6) must be formulated in terms of affine invariant quantities — including affine invariant substitute criteria for (1.7),(1.8).

1.1.3 Natural monotonicity test

In order to create a computationally available affine invariant convergence (monotonicity) criterion one may define *generalized level functions*

$$T(x|B) := \frac{1}{2}\|B \cdot F(x)\|_2^2 = \frac{1}{2}F^TB^TBF\,,\ B \in GL(n) \tag{1.14}$$

which have the property

$$\begin{aligned} T(x|B) > 0 &\Leftrightarrow x \neq x^*\,, \\ T(x|B) = 0 &\Leftrightarrow x = x^*\,. \end{aligned}$$

With an accepted iterate x_k and a *trial iterate* x_{k+1} (e.g. from (1.6)) at hand, the associated *generalized monotonicity criterion* reads

$$T(x_{k+1}|B) < T(x_k|B)\,. \tag{1.15}$$

A quite outstanding choice for B turns out to be (see (1.25) below)

$$B := J(x_k)^{-1}\,. \tag{1.16}$$

This choice ensures affine invariance of (1.14) and (1.15). This can be seen easily by applying (1.14), with the choice (1.16), to the original problem (1.2) and to the transformed problem (1.12) respectively. With the notation $F_k = F(x_k)$, $J_k = J(x_k) = F'(x_k)$, $G_k = G(x_k)$, $G'_k = G'(x_k)$, one has

$$T_F^k := T(x_k|J_k^{-1}) = \frac{1}{2} F_k^T J_k^{-T} J_k^{-1} F_k$$

and

$$T_G^k := T(x_k|{G'}_k^{-1}) = \frac{1}{2} G_k^T {G'}_k^{-T} {G'}_k^{-1} G_k\,.$$

Using $G = AF, G' = AJ$ one immediately obtains:

$$\begin{aligned} T_G^k &= \tfrac{1}{2}(AF_k)^T (AJ_k)^{-T} (AJ_k)^{-1} (AF_k) \\ &= \tfrac{1}{2} F_k^T A^T A^{-T} J_k^{-T} J_k^{-1} A^{-1} A F_k \\ &= \tfrac{1}{2} F_k^T J_k^{-T} J_k^{-1} F_k \\ &= T_F^k \end{aligned}$$

For the left hand side of (1.15) one has

$$\overline{T}_F^k := T(x_{k+1}|J_k^{-1}) = \frac{1}{2} F_{k+1} J_k^{-T} J_k^{-1} F_{k+1}$$

and obviously

$$\begin{aligned} \overline{T}_G^k &:= T(x_{k+1}|{G'}_k^{-1}) \\ &= \overline{T}_F^k \end{aligned}$$

holds. Thus, the approach to the solution x^* of a trial iterate x_{k+1} is tested by the affine invariant *natural monotonicity check*

$$T(x_{k+1}|J_k^{-1}) < T(x_k|J_k^{-1})\,. \tag{1.17}$$

In contrast to the above mentioned convergence check (1.10) — which is, however, not affine invariant — the evaluation of (1.17) requires some additional computational work. First, note that the evaluation of

$$T(x_k|J_k^{-1}) = \frac{1}{2}\|J_k^{-1} F\|_2^2 = \frac{1}{2}\|\Delta x_k\|_2^2$$

is (essentially) for free, as the ordinary Newton correction

$$\Delta x_k = -J_k^{-1} F_k \tag{1.18}$$

has been already computed in order to generate the trial iterate $x_{k+1} = x_k + \lambda \Delta x_k$. But, in order to evaluate

$$T(x_{k+1}|J_k^{-1}) = \frac{1}{2}\|J_k^{-1}F_{k+1}\|_2^2$$

the so-called *simplified Newton correction*

$$\overline{\Delta x}_{k+1} := -J_k^{-1} F_{k+1} \tag{1.19}$$

has to be computed additionally. However, this additional amount of work is usually small — compared to the overall costs for one Newton step (evaluation of F_k, J_k, solution of (1.18)). Recall that the computation of the ordinary Newton correction is usually done by solving the linear system (1.6.a) with *direct linear solvers* (LU-decomposition of J_k, followed by a forward/backward substitution). Now, in order to compute $\overline{\Delta x}_{k+1}$ just another forward/backward substitution is necessary. Things may change for large scale systems, where the linear systems must be solved with *iterative methods.* Affine invariant Newton techniques for this case have been studied in [8]. One result of the investigations in [8, 24] is, that the additional costs for solving (1.19) are again acceptable.

The combination of a damped Newton iteration (1.6) with the natural monotonicity criterion (1.17) turns out to be the essential step in order to have a globally convergent Newton method. In principle, any (affine invariant) damping strategy can be added to (1.6),(1.17). As an example take a steplength strategy of Armijo type [1] selecting damping factors

$$\begin{aligned}
&\lambda_k \in \{1, \tfrac{1}{2}, \tfrac{1}{4}, \ldots, \lambda_{min}\} \text{ subject to} \\
&T(x_k + \lambda_k \Delta x_k | J_k^{-1}) \le (1 - \frac{\lambda_k}{2}) T(x_k|J_k^{-1}) \\
&T(x_k + \lambda_k \Delta x_k | J_k^{-1}) = \min_{\lambda} T(x_k + \lambda \Delta x_k | J_k^{-1})\,.
\end{aligned}$$

Concerning a substitute stopping criterion for (1.8), the condition $\|\Delta x_{k+1}\| < tol$ is a quite natural choice, as the Newton iteration converges quadratically near the solution x^*. Moreover, since $\|\Delta x_{k+1}\| \approx \|\overline{\Delta x}_{k+1}\|$ holds near x^* (i.e. $\lambda = 1$), one may apply a criterion in terms of the simplified Newton correction $\overline{\Delta x}_{k+1}$,

$$\|\overline{\Delta x}_{k+1}\| \le tol\,. \tag{1.20}$$

In this case, the current iterate x_{k+1} may be improved by adding $\overline{\Delta x}_{k+1}$, thus saving the computation of the next ordinary Newton correction Δx_{k+1}. In order to overcome pathological situations, the above criterion (1.20) is only accepted for termination, if additionally the condition

$$\begin{aligned} &a) \quad \|\Delta x_k\| \leq \sqrt{tol \cdot 10} \\ &b) \quad \lambda_k = 1 \end{aligned} \tag{1.21}$$

holds.

Finally, a quite interesting feature of the natural level function should be mentioned. The steepest descent direction $\widetilde{\Delta x}_k$ for $T(x_k|B)$ in x_k is given by

$$\widetilde{\Delta x}_k := -\mathrm{grad}T(x_k|B) = -J_k^T B^T B F_k \,.$$

Therefore, the specification $B := J_k^{-1}$ yields

$$\widetilde{\Delta x}_k = -J_k^T J_k^{-T} J_k^{-1} F_k = \Delta x_k \,.$$

In other words: for a given x_k, the steepest descent direction for the natural level function $T(x_k|J_k^{-1})$ and the ordinary Newton correction coincide.

1.1.4 Damping strategy

The above mentioned outstanding choice $B := J_k^{-1}$, as well as the selection of an associated *optimal damping factor* λ_k for (1.6) is based on a substantial theoretical result (see [7, 5] for details and proper formulation). In order to characterize the nonlinearity of the problem, one may introduce an affine invariant Jacobian Lipschitz constant ω. Assume that a global constant ω exists such that

$$\begin{aligned} &\|J(y)^{-1}[J(y) - J(x)]\| \leq \omega \|y - x\| \\ &x, y \in D \,,\ \omega < \infty \end{aligned} \tag{1.22}$$

Then, with the convenient notation

$$h_k := \|\Delta x_k\| \cdot \omega \,,\ \ \overline{h}_k := h_k \mathrm{cond}(BJ(x_k)) \tag{1.23}$$

the following inequality holds

$$\begin{aligned} &a) \quad T(x_k + \lambda \Delta x_k|B) \leq t_k^2(\lambda|B) T(x_k|B) \\ &\text{where} \\ &b) \quad t_k(\lambda|B) = 1 - \lambda + \tfrac{1}{2}\lambda^2 \overline{h}_k \end{aligned} \tag{1.24}$$

The result (1.24) implies, that maximizing the descent means minimizing $t_k(\lambda|B)$. Straightforward calculations leads to the optimal choice

$$\begin{aligned} &a) \quad \lambda_k^{opt}(B) = \min\left\{1, \frac{1}{\overline{h}_k}\right\} \\ &b) \quad B_k^{opt} = J(x_k)^{-1} \end{aligned} \tag{1.25}$$

with the extremal properties ($B \in GL(n)$)

$$\begin{aligned} &a) \quad t_k(\lambda|J_k^{-1}) \le t_k(\lambda|B) \\ &b) \quad \lambda_k^{opt}(J_k^{-1}) \ge \lambda_k^{opt}(B)\,. \end{aligned} \tag{1.26}$$

In order to have a computational available estimate λ_k for $\lambda_k^{opt}(J_k^{-1})$ due to (1.25) and (1.23) an estimate for ω is required. From (1.22) a local estimate $[\omega_k]$ for the local Lipschitz constant $\omega_k \le \omega$ can be derived. Estimating

$$\begin{aligned} \|\overline{\Delta x}_k - \Delta x_k\| &= \|J_{k-1}^{-1}F_k - J_k^{-1}F_k\| \\ &= \|J_k^{-1}J_kJ_{k-1}^{-1}F_k - J_k^{-1}J_{k-1}J_{k-1}^{-1}F_k\| \\ &= \|J_k^{-1}(J_k - J_{k-1})\overline{\Delta x}_k\| \\ &\le \|J_k^{-1}[J(x_{k-1} + \lambda_{k-1}\Delta x_{k-1}) - J(x_{k-1})]\| \cdot \|\overline{\Delta x}_k\| \\ &\le \omega\lambda_{k-1}\|\Delta x_{k-1}\| \cdot \|\overline{\Delta x}_k\|\,, \end{aligned}$$

an affine invariant local estimate

$$[\omega_k] := \frac{\|\overline{\Delta x}_k - \Delta x_k\|}{\lambda_{k-1}\|\Delta x_{k-1}\| \cdot \|\overline{\Delta x}_k\|} \le \omega_k \le \omega \tag{1.27}$$

is inspired. Insertion into (1.23) and (1.25) respectively yields an *a priori estimate* for the damping factor λ_k^{opt}:

$$\begin{aligned} &a) \quad \lambda_k^{(0)} := \min\left\{1, \frac{1}{[h_k^{(0)}]}\right\} \\ &b) \quad [h_k^{(0)}] := \frac{\|\overline{\Delta x}_k - \Delta x_k\|\|\Delta x_k\|}{\|\Delta x_{k-1}\|\|\overline{\Delta x}_k\|} \cdot \lambda_{k-1}\,. \end{aligned} \tag{1.28}$$

Note that this so-called *prediction strategy* requires information from the previous Newton iteration (accepted damping factor λ_{k-1}, corrections $\overline{\Delta x}_k$, Δx_{k-1}). Thus, for the very first iteration of (1.6) an (user given) initial estimate $\lambda_0^{(0)}$ is required. However, if this value (or the a priori factor $\lambda_k^{(0)}$, $k > 0$) leads to a failure of the natural monotonicity test, a so called *reduction strategy* can be invoked. Recall that at this stage of the computation a first trial value $x_{k+1} = x_k + \lambda_k^{(0)}\Delta x_k$ and the associated simplified Newton correction $\overline{\Delta x}_{k+1}^{(0)} = -J_k^{-1}F(x_{k+1}^{(0)})$ are at hand. Based on this information one is able (c.f. Bock [2]) to compute (recursively) *a posteriori estimates* $[h_k^{(j)}], j = 1, 2, \ldots$ and to establish a reduction strategy

$$\begin{aligned} &a) \quad \lambda_k^{(j)} := \min\left\{1, \frac{\lambda_k^{(j-1)}}{2}, \frac{1}{[h_k^{(j)}]}\right\} \\ &b) \quad [h_k^{(j)}] := \frac{2}{\lambda_k^{(j-1)}} \frac{\|\overline{\Delta x}_{k+1}^{(j-1)} - (1 - \lambda_k^{(j-1)})\Delta x_k\|}{\|\Delta x_k\|}\,, \quad j = 1, 2, \ldots \end{aligned} \tag{1.29}$$

Usually, this a posteriori loop is quite rarely activated. Nevertheless, in order to avoid an infinite loop, the estimates $\lambda_k^{(j)}$, $j = 1, 2, \ldots$ — as well as the a priori estimate $\lambda_k^{(0)}$ — must satisfy the condition $\lambda_k^{(j)} \geq \lambda_{min}$, $j = 0, 1, 2, \ldots$, where λ_{min} denotes a minimal permitted damping factor. Otherwise, the damped Newton iteration (1.6) is stopped.

1.1.5 Newton path

There is a quite interesting theoretical consideration which gives insight to the behavior of the damped Newton algorithm derived above. Recall the definition of the generalized level function (1.14) and the associated generalized monotonicity criterion (1.15). As long as B is not yet specified, a generalized, but quite natural, requirement for an iterative method is, that the next iterate x_{k+1} descends — but now for all possible choices $B \in GL(n)$. This requirement can be formulated by introducing the generalized level sets

$$G(x_k|B) := \{x \in \mathrm{IR}^n | T(x|B) \leq T(x_k|B)\} \tag{1.30}$$

and rewriting the monotonicity criterion (1.15) to

$$x_{k+1} \in G(x_k|B)\,.$$

With that, the above quoted generalized requirement reads

$$\begin{aligned} x_{k+1} &\in \overline{G}(x_k) \\ \overline{G}(x_k) &:= \bigcap_{B \in GL(n)} G(x_k|B)\,. \end{aligned}$$

Under certain assumptions, the intersection $\overline{G}(x_k)$ exists and turns out to be a topological path $\overline{x} : [0, 2] \longrightarrow \mathrm{IR}^n$, the so called Newton path, which satisfies

$$\begin{aligned} a)\quad & F(\overline{x}(s)) = (1-s)F(x_k) \\ & T(\overline{x}(s)|B) = (1-s)^2 T(x_k|B) \\ b)\quad & \frac{d\overline{x}}{ds} = -J(\overline{x})^{-1}F(x_k) \\ & \overline{x}(0) = x_k\,,\ \ \overline{x}(1) = x^* \\ c)\quad & \left.\frac{d\overline{x}}{ds}\right|_{s=0} = -J(x_k)^{-1}F(x_k) \equiv \Delta x_k\,. \end{aligned} \tag{1.31}$$

This result (see e.g. [7]) deserves some contemplation. The constructed Newton path $\overline{x}$ is outstanding in the respect that *all level functions* $T(x|B)$ *decrease along* $\overline{x}$ — this is the result (1.31.a). Therefore, a rather natural approach would be to just follow that path computationally, say, by numerical integration of the initial value problem(1.31.b). But, as — in contrast to the original problem (1.3) — the problem (1.31.b) can be extremely ill-posed, this approach is no real alternative. Furthermore, even following the path with an appropriate piece of numerical software, solving (1.31.b) is quite costly and shows no benefit — compared to a solution with the damped Newton codes presented here. However, the local information about the tangent direction at x_k should be used — which is just the Newton direction (result (1.31.c)). This means that even "far away" from the solution point x^*, the Newton direction

$$\frac{\Delta x_k}{\|\Delta x_k\|}$$

is an outstanding direction, only the length $\|\Delta x_k\|$ will be too large for highly nonlinear problems. Geometrically speaking, the (damped) Newton step in x_k continues along the tangent of the Newton path $\overline{G}(x_k)$ with an appropriate length and at the next iterate x_{k+1}, the next Newton path $\overline{G}(x_{k+1})$ will be chosen for orientation towards x^*. Likewise, these considerations reveal the limit of the affine invariant Newton techniques. If the Newton path (from x_0 to x^*) does not exist, the damped Newton iteration will, in general, fail to converge. This situation occurs, if the solution point x^* and the initial guess x_0 are separated by a manifold with singular Jacobian. Typically, this case may arise in problems, where multiple solution points exist. In practical applications, however, different solution points x^* usually have a distinct physical interpretation. Thus, using a solution method, which "connects" the (meaningful) starting point x_0 with the "associated" solution x^* may be a great advantage. This feature of the algorithm shows up also in the codes and is illustrated in Figure 1.1. Herein, the Newton path $\overline{G}(x_0)$ connecting the

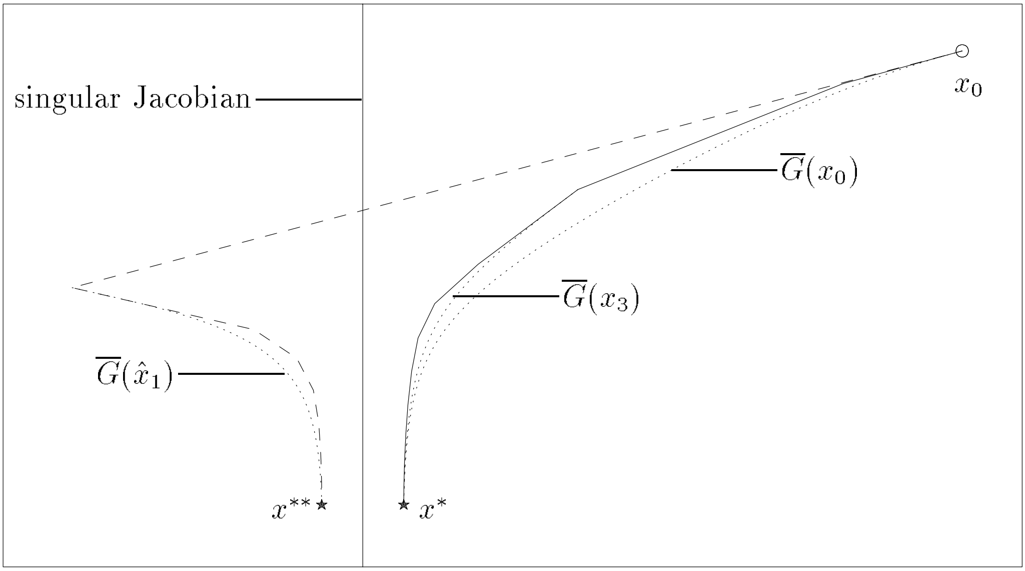

Figure 1.1 Newton path and Newton iterates

(positive) starting point x_0 and the (positive) solution x^* is nicely "followed" by the damped Newton iterates (x_k, connected by solid line), whereas the undamped Newton iteration ($\|\Delta x_0\|$ too large) crosses the critical line $x_{(1)} = 0$, where the Jacobian is singular. Consequently, these iterates ($\hat{x}_k$, dashed line) converge to the symmetric (negative) solution — following the Newton path $\overline{G}(\hat{x}_1)$. In order not to overload the Figure 1.1 just three Newton pathes ($\overline{G}(x_0)$, $\overline{G}(x_3)$, $\overline{G}(\hat{x}_1)$, dotted lines), which have been computed with the Linearly IMplicit EXtrapolation integrator LIMEX [9, 10], are plotted in the Figure.

1.2 Basic algorithmic scheme

The following informal algorithm shows the basic structure of a damped affine invariant Newton iteration (including steplength strategy) due to Deuflhard. Essentially, this scheme consists of the outer Newton iteration loop and an inner steplength reduction loop. This a posteriori loop is part of the outer loop and may be performed repeatedly. The Newton step comprises control of convergence and check for termination, as well as an a priori estimate for the damping factor λ. Within the steplength reduction loop just a refined (a posteriori) damping factor is selected and the convergence of the associated refined Newton iterate is checked again.

Global affine invariant Newton scheme (Algorithm B)

Input:

x_0	initial guess for the solution
tol	required accuracy for the solution
λ_0	initial damping factor
λ_{min}	minimal permitted damping factor
$itmax$	maximum permitted number of iterations

user routine to evaluate the nonlinear system function $F(x)$
user routine to evaluate the Jacobian of the system $J(x) := \frac{\partial F}{\partial x}$
(may be dummy as internal numerical differentiation procedures may be used)
standard routines for direct solution of linear equations

Start:

$$k := 0$$

evaluate system

$$F_k := F(x_k)$$

Newton step:

evaluate Jacobian

$$J_k := J(x_k)$$

compute ordinary Newton correction

$$\Delta x_k := -J_k^{-1} F_k$$

compute a priori damping factor

$$\text{if } (k > 0) \quad \lambda_k^{(0)} := \min\left\{1, \frac{1}{[h_k^{(0)}]}\right\}$$

$$\text{where } \; [h_k^{(0)}] := \frac{\|\overline{\Delta x}_k - \Delta x_k\| \|\Delta x_k\|}{\|\Delta x_{k-1}\| \|\overline{\Delta x}_k\|} \cdot \lambda_{k-1}$$

$$\text{else} \qquad \lambda_k^{(0)} := \lambda_0$$

$$j := 0$$

$$\lambda := \max\{\lambda_k^{(0)}, \lambda_{min}\}$$

a posteriori loop

compute trial iterate

$$x_{k+1}^{(j)} := x_k + \lambda \Delta x_k$$

evaluate system

$$F_{k+1}^{(j)} := F(x_{k+1}^{(j)})$$

compute simplified Newton correction

$$\overline{\Delta x}_{k+1}^{(j)} := -J_k^{-1} F_{k+1}^{(j)}$$

termination check

$$exit = (\|\overline{\Delta x}_{k+1}^{(0)}\| \leq tol \wedge \|\Delta x_k\| \leq \sqrt{tol \cdot 10} \wedge \lambda = 1)$$

$$\text{if } exit: \quad x_{out} := x_{k+1}^{(0)} + \overline{\Delta x}_{k+1}^{(0)}$$

solution exit

compute a posteriori damping factor

$$\lambda_k^{(j+1)} := \min\left\{1, \frac{1}{[h_k^{(j+1)}]}\right\}$$

$$\text{where } \; [h_k^{(j+1)}] := \frac{2}{\lambda} \frac{\|\overline{\Delta x}_{k+1}^{(j)} - (1-\lambda)\Delta x_k\|}{\|\Delta x_k\|}$$

```
monotonicity check
        $konv := \|\overline{\Delta x}^{(j)}_{k+1}\| \leq \|\Delta x_k\|$

    if $konv$  :  $\overline{\Delta x}_{k+1} := \overline{\Delta x}^{(j)}_{k+1}$
                  $x_{k+1} := x^{(j)}_{k+1}$
                  $F_{k+1} := F^{(j)}_{k+1}$
                  $\lambda_k := \lambda$
                  $k := k + 1$
                  if ( $k > itmax$ ) : fail exit
                  proceed at *Newton step*
    else:         $j := j + 1$
                  if ( $\lambda = \lambda_{min}$ ) : fail exit
                  $\lambda := \min\{\lambda^{(j)}_k, \frac{\lambda}{2}\}$
                  $\lambda = \max\{\lambda, \lambda_{min}\}$
                  proceed at *a posteriori loop*
```

For the above scheme, the minimal user input consists of x_0, $F(x)$ and *tol*, as internal standard values (routines) for λ_0, λ_{min}, *itmax*, $J(x)$ and linear system solution are available within the codes. In order to perform one Newton step with this scheme, the essential computational work turns out to be: one evaluation of $J(x)$, one evaluation of $F(x)$ and the solution of two linear systems — as long as no a posteriori steplength reduction is necessary. In such a case, each reduction step requires additionally one evaluation of F and one linear system solution, but, this device is activated quite rarely.

1.3 Details of algorithmic realization

In order to describe the underlying algorithms of the codes NLEQ1, -1S, -2 some details of the algorithmic realization are worth mentioning. In practical applications the robustness and efficiency of the algorithm presented in the preceding Sections may e.g. strongly depend on the selected norm, the reasonable choice of termination criterion and the chance to select special variants or modifications of the basic scheme.

1.3.1 Choice of norm

Generally speaking, to control the performance of the damped Newton algorithm *smooth* norms (such as the Euclidean norm $\|\cdot\|_2$) are recommended. Non-smooth norms (such as the max-norm $\|\cdot\|_\infty$) may lead to some nonsmooth performance, e.g. alternating between competing components of the iterates. For the termination criterion of the Newton iteration, however, the max-norm may be used. Within the current realization of NLEQ the so called *root mean square* norm is used. This norm and the underlying scalar product is defined by

$$\|v\|_{rms} := \sqrt{\frac{1}{n}\sum_{i=1}^{n} v_i^2} \qquad v \in \mathrm{IR}^n \tag{1.32}$$

Note that for large scale systems this norm may significantly differ from the usual $\|\cdot\|_2$ or $\|\cdot\|_\infty$ norms, which are defined as follows:

$$\|v\|_2 \;:=\; \sqrt{\sum_{i=1}^{n} v_i^2} \tag{1.33}$$

$$v \in \mathrm{IR}^n$$

$$\|v\|_\infty \;:=\; \max\{|v_i|\} \tag{1.34}$$

The choice of (1.32) is motivated by the following consideration. Assume that the problem (1.3) represents a discretized PDE. In order to check the quality of the discretization, one may solve the underlying continuous problem on grids with different levels of fineness, i.e. varying dimension n of the discretized problem. For adequate discretizations one may expect the same behavior of the Newton scheme — (almost) independent of n. To achieve this aim, the algorithm must use quantities which are independent of the dimension of the problem — like $\|\cdot\|_{rms}$ or $\|\cdot\|_\infty$. Note that for special classes of applications, the use of (1.32) within NLEQ is certainly not the best choice, but for an algorithm which is designed to solve general problems of the form (1.3) this choice turns out to be quite reasonable. Observe that the Newton scheme, as presented here, is exclusively controlled by norms to be evaluated in the space of the iterates (and not in the space of the residuals) — a necessary condition for an affine invariant method. Furthermore, in order to control the algorithmic performance ratios of norms are used, whereas the absolute value of a norm is just used for the termination criterion. These facts are of essential importance for the reliability of the algorithm.

1.3.2 Scaling and weighting

A proper *internal scaling* plays an important role for the efficiency and robustness of an algorithm. A desirable property of an algorithm is the so called *scaling invariance*. This means, e.g. regauging of some or all components of the vector of unknowns x (say, from Å to km) should not effect the algorithmic performance — although the problem formulation may change. In order to discuss this essential point consider a *scaling transformation* defined by:

$$\begin{array}{l} a) \quad x \mapsto y := S^{-1}x \\ \text{with a diagonal transformation matrix} \\ b) \quad S := \mathrm{diag}(s_1, \ldots, s_n) \end{array} \tag{1.35}$$

Insertion into the original problem (1.3) leads to a transformed problem

$$H(y) := F(Sy) = F(x) = 0 \tag{1.36}$$

where the associated solution y^* and Jacobian matrix H_y are given by

$$\begin{aligned} y^* &= S^{-1}x^* \\ H_y(y) &= F_x(x) \cdot S = J(x) \cdot S \end{aligned} \tag{1.37}$$

The problem (1.3) is said to be *covariant* under the scaling transformation (1.35) — a property which is shared by the ordinary Newton method, as for $k = 0, 1, \ldots$

$$\begin{aligned} &a) \quad \Delta y_k = -H_y^{-1}(y_k)H(y_k) = -S^{-1}J^{-1}(x_k)F(x_k) = S^{-1}\Delta x_k \\ &b) \quad y_{k+1} = y_k + \Delta y_k = S^{-1}x_k + S^{-1}\Delta x_k = S^{-1}x_{k+1} \end{aligned} \tag{1.38}$$

holds. Note that the theoretical covariance property $y_k = S^{-1}x_k$ may be disturbed in real computations due to roundoff, except for special realizations like symbolic computations. As long as the Newton update is done via (1.38b) the simplified Newton correction is covariant also:

$$\overline{\Delta y}_{k+1} = -H_y^{-1}(y_k)H(y_{k+1}) = -S^{-1}J^{-1}(x_k)F(x_{k+1}) = S^{-1}\overline{\Delta x}_{k+1} \quad (1.36.c)$$

But, if norms (in the space of the iterates) enter into the algorithm, e.g. to perform a damping strategy or an error estimation, the covariance property of the algorithm is lost. As, in general

$$\|\Delta y_k\| = \|S^{-1}\Delta x_k\| \neq \|\Delta x_k\| \tag{1.39}$$

holds, the control and update procedures within the algorithm will generate a different algorithmic performance if they are applied to problem (1.36) instead of (1.3). To overcome this difficulty one may internally replace the usual norm (e.g. (1.32)) by an associated *scaled* or *weighted norm*:

$$\|v\| \longrightarrow \|D^{-1}v\| \tag{1.40}$$

where D is a diagonal matrix to be chosen. Consider now the first Newton step of algorithm (B). Assume, a choice

$$D := \text{diag}(x_1^0, \ldots, x_n^0) \quad , \qquad x^0 \text{ initial guess for } x^* \tag{1.41}$$

is possible ($x_i^0 \neq 0, i = 1, \ldots, n$). Inserting (1.41) into (1.40) yields for system (1.3):

$$\|\Delta x_0\| \longrightarrow \|D^{-1}\Delta x_0\| \tag{1.42}$$

Applied to the transformed system (1.36) one has

$$\overline{D} := \text{diag}(y_1^0, \ldots, y_n^0) \quad , \qquad y^0 \text{ initial guess for } y^*$$

and due to

$$y^0 = S^{-1}x^0$$

one has

$$\overline{D}^{-1} = D^{-1}S$$

thus:

$$\|\Delta y_0\| \longrightarrow \|\overline{D}^{-1}\Delta y_0\| = \|D^{-1}SS^{-1}\Delta x_0\| = \|D^{-1}\Delta x_0\| \,. \tag{1.43}$$

In contrast to the case of unscaled norms (c.f. (1.39)) for the scaled norms (1.42) and (1.43) the norms of the first Newton corrections coincide. The same holds for the norms of the first simplified Newton correction. From this follows, that the first monotonicity test (1.17) will lead to the same algorithmic consequences, independent of an eventual a priori transformation of type (1.35). Even all subsequent decisions of the algorithm will be invariant. With that, the fixed choice of (1.40) for the internal scaling matrix D will yield invariance for all Newton steps. But concerning the termination criterion of the Newton scheme an *adaptive* choice is indispensable.

Consider the natural stopping criterion for a Newton method in its unscaled form:

$$\begin{aligned} &err := \|\Delta x_k\| \\ &\text{stop, if} \quad err \leq tol \\ &tol\ :\ \text{prescribed (required) tolerance (accuracy)} \end{aligned} \tag{1.44}$$

In this unscaled form, *err* is a measure for the *absolute* error of the numerical solution x_k. Using a scaled norm

$$err := \|D_*^{-1}\Delta x_k\| \tag{1.45}$$

with

$$D^* := \mathrm{diag}(x_1^*, \ldots, x_n^*) \tag{1.46}$$

err is a measure of the *relative error* of x_k. Note that $\|D_*^{-1}(x^* - x_k)\|$ is the true relative error of x_k (still depending on the selected norm), whereas $\|D_*^{-1}\Delta x_k\|$ is just an estimate of it, but a quite reasonable one, as Newton's method converges quadratically near the solution x^*. Again, similar to (1.41), $x_i^* \neq 0,\ i = 1, \ldots, n$ is required, but, in any case, x^* is usually not available. To avoid the difficulties coming from zero components and to connect the natural scaling matrices D_0, D_* ((1.41), (1.46)) within the course of the Newton iteration the following scaling strategy is applied. An internal *weighting vector* xw is used to define local scaling matrices D_k by

$$D_k := \mathrm{diag}(xw_1, \ldots, xw_n) \tag{1.47}$$

and xw may be locally defined by:

$$xw_i := \max\{|x_i^k|, thresh\} \tag{1.48}$$

where

$$thresh > 0 : \text{ threshold value for scaling.}$$

This scaling procedure yields reasonable values for the scaled norms used in the codes. Note that the actual value of *thresh* determines a componentwise switch from a pure relative norm to a modified absolute norm. As long as $x_i^k > thresh$ holds, this component contributes with

$$\frac{\Delta x_i^k}{|x_i^k|}$$

to the norm, whereas for $x_i^k \leq thresh$ this component contributes with

$$\frac{\Delta x_i^k}{thresh}$$

to the total value of the norm. In order to allow a componentwise selection of *thresh* and to take into account that the damped Newton algorithm uses information from two successive iterates the following extension of (1.47) and (1.48) is used in the codes.

Scaling update procedure

Input:

$xw^u :=$ user given weighting vector

Initial check:

a) if $(|xw_i^u| = 0)$

$$xw_i^u := \begin{cases} tol & \text{if problem is highly nonlinear} \\ 1 & \text{if problem is mildly nonlinear} \\ & \text{(see Table 1.1 for problem type)} \end{cases} \qquad (1.49)$$

Initial update:

b) $xw_i^0 := \max\{|xw_i^u|, |x_i^0|\}$

Iteration update:

c) $xw_i^k := \max\left\{|xw_i^u|, \frac{1}{2}(|x_i^{k-1}| + |x_i^k|)\right\}$

Thus, the scaling matrix and norm (*weighted root mean square*) used in the codes are given by:

$$\begin{aligned} &a) \quad D_k := \operatorname{diag}(xw_1, \ldots, xw_n) \\ &b) \quad \|v\| := \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{v_i}{xw_i}\right)^2} \;. \end{aligned} \qquad (1.50)$$

Remarks:

(1) The final realization of scaling within the algorithm must be done carefully to achieve properly scaled norms for terms which include values from different iterates.

(2) In order to allow a scaling totally under user control, the updates (1.49.b,c) can be inhibited optionally.

(3) In order to have scaling invariance also for the linear system solution, the arising linear systems are internally scaled:

$$J_k \Delta x_k = -F_k \longrightarrow (J_k D_k)(D_k^{-1} \Delta x_k) = -F_k \qquad (1.51)$$

Thus, a user rescaling via (1.35) does not change the performance of the linear solver. For a further discussion of linear system solution see Section (1.3.4).

1.3.3 Achieved accuracy

First, recall that all norms used for the algorithmic control are evaluated in the space of the iterates x_k and not in the space of the residuals. Concerning the termination criterion this means, that the associated error estimate yields a direct measure for the error of the associated solution vector. As scaled norms are used (*relative* error criterion) an interpretation in terms of correct leading decimal digits is possible. Assume, a termination criterion of the form

$$err_{rel} \approx \|\Delta x_k\| \leq tol \qquad (1.52)$$

holds, where $\|\cdot\|$ is the weighted root mean square norm (1.50). Then, one has roughly

$$cld := -\log_{10}(tol)$$

correct decimal leading digits in the mantissa of each component x_i^k, independent of the actual exponent — except $x_i^k \ll xw_i^k$. In such a case the number of correct digits in the mantissa is approximately

$$cld := -(\log_{10}(tol) - (\log_{10}(|x_i^k|) - \log_{10}(xw_i^k))) \; .$$

In other words, the componentwise *absolute* error is, for both cases, approximately given by

$$err_{abs}^i \approx tol \cdot xw_i^k \; .$$

In contrast to this, an unscaled termination criterion in the space of the residuals

$$\|F(x_k)\| \leq tol$$

neither controls the error in the computed solution nor shows any invariance property. A simple reformulation of the original problem (1.3) of the form

$$F \longrightarrow \hat{S}F =: \hat{F} \ , \quad \hat{S} = \mathrm{diag}(tol^{-1}, \ldots, tol^{-1})$$

will lead to

$$\|\hat{F}(x_k)\| \approx 1$$

whereas a stopping criterion like (1.52) is not affected. In order to realize invariance against such a rescaling one may again use a scaled check, e.g.

$$\|\hat{D}^{-1}F\| \leq tol$$

where

$$\hat{D} := \mathrm{diag}(F_1(x_0), \ldots, F_n(x_0)) \ .$$

However, there is some arbitrariness in the choice of $\hat{D}$ and it is not clear how to develop an adaptive selection of further scaling matrices $\hat{D}_k$. In any case, the disadvantage of checking in the wrong space is still remaining.

Remark: Assume that the problem (1.3) is well scaled, i.e. unscaled norms yield meaningful numbers. If in such a situation

$$\|\Delta x_k\| \leq tol \quad \text{with} \quad \|F(x_k)\| \text{ "large"}$$

holds, the underlying problem is said to be ill-conditioned. That means, $\|F(x)\|$ "large" may occur even for $x := float(x^*)$ — just because x^* can't be represented exactly due to the finite length of the mantissa. For a badly scaled problem a check for the condition of the problem must use scaled norms, i.e.

$$\|D^{-1}\Delta x_k\| \leq tol \text{ with } \|\hat{D}^{-1}F(x_k)\| \text{ "large"} \tag{1.53}$$

indicates an ill-conditioned problem, provided that $D, \hat{D}$ are properly chosen. Note that within the codes an optional printout of $\|F(x_k)\|$ is possible — but in unscaled form. Thus, situations like (1.53) may be pretended to users, but due to $\hat{D} = I$ the problem is well-conditioned but ill-scaled.

Furthermore, recall that the implemented termination criterion for the Newton iteration in NLEQ is a modification of (1.52). But the considerations made above hold also for the implemented stopping criterion (1.20), (1.21).

Finally, note that for the Newton iteration no heuristic divergence criterion is needed. Clearly, a maximum number of iterations may be prescribed, but usually an internal fail exit condition

$$\lambda_k^{(j)} < \lambda_{min} \tag{1.54}$$

will stop a divergent iteration before.

1.3.4 Solution of linear systems

All codes presented in this paper use direct methods to solve the arising linear systems of type

$$A \cdot x = b\,. \tag{1.55}$$

Recall, that within the course of the damped Newton iteration (Algorithm B) systems with varying right hand side b but fixed matrix A have to be solved. Thus, a split into a factorization (decomposition) phase and a forward/backward substitution (solution) phase is a quite natural requirement. As, in general, application of Gaussian elimination is the most efficient way to solve (1.55), the corresponding standard software from LINPACK [12] is used within NLEQ1 in order to perform LU-factorization (DGEFA) and solution (DGESL). For a rather wide class of problems this software works quite efficient and robust. If, however, the dimension of the problem is "not small", the above mentioned standard *full mode* software may be no longer the method of choice. For large n, however, the Jacobian matrices often have a special structure. As a typical example take discretized PDE problems (in one space dimension). Usually, the associated Jacobian shows band structure. Thus, within NLEQ1 a special *band mode* LU-factorization solution software, again from LINPACK, is available.

Whenever the Jacobian shows no special structure but turns out to be sparse (number of nonzero elements $\approx c \cdot n, c \ll n$) *sparse mode* elimination techniques may be successfully applied up to a considerable size of n. Within NLEQ1S, the well known MA28-package due to Duff [13, 14] is used to solve the linear problems (1.55). In order to call the MA28 codes as efficient as possible, the adaptation techniques presented by Duff/Nowak [15] for an application of MA28 within a stiff extrapolation integrator, can be essentially applied.

A quite different elimination technique is used in the code NLEQ2. Therein, a special QR-decomposition [4] of A is performed. During this decomposition the so-called *sub-condition* number sc(A) (see Deuflhard/Sautter [11]) is monitored. If this estimate for the condition of the matrix A becomes "too large", say

$$\mathrm{sc}(A) > \frac{1}{epmach}\,, \qquad \textit{epmach}\text{: relative machine precision} \tag{1.56}$$

instead of (formally) generating the inverse A^{-1}, a rank-deficient Moore-Penrose *pseudo inverse* A^{+} is computed. In other words, instead of

$$\overline{x} := A^{-1}b \tag{1.57}$$

a unique, but rank deficient, solution

$$\tilde{x} := A^{+}b \tag{1.58}$$

of the associated linear least squares problem $\|Ax - b\|_2^2 = \min$ is used as the numerical approximation to the true solution x of (1.55). Roughly speaking, the reason for this rank reduction is the finite length of the mantissa. The quality of the numerical solution of a linear system (1.55) may strongly depend on the condition of A (cond(A)). If A is ill-conditioned — but still regular in the sense that non-zero pivots can be found — the computed solution $\overline{x}$ (via (1.57)) may be totally (or partially) wrong, as small errors due to roundoff may be amplified, (in the worst case) by the factor cond(A) to errors in $\overline{x}$.

In order to avoid this, in NLEQ2 the QR-decomposition/solution codes DECCON/SOLCON are used to compute a solution of the arising linear systems — either via (1.57) if A is well conditioned or via (1.58) if A is extremely ill-conditioned. The rank reduction of A can be interpreted as a replacement

$$A \longrightarrow \tilde{A}^{(q)}\,, \quad q := \mathrm{rank}(\tilde{A}^{(q)}) < n$$

such that

$$\mathrm{sc}(\tilde{A}^{(q)}) \leq \frac{1}{epmach}\,, \quad \mathrm{sc}(\tilde{A}^{(q+1)}) > \frac{1}{epmach}$$

holds. The general disadvantage of solving (1.55) with QR-decomposition and associated rank reduction option should be mentioned. Usually, a QR-decomposition is more expensive (roughly twice) than a LU-decomposition. Furthermore, no efficient extension (especially because of the sub-condition estimate) for large systems with special structure (banded, sparse) is available.

Finally, a proper scaling of the system (1.55) plays an important role. As pointed out above, the linear systems, to be solved with the above mentioned decomposition techniques, are already (column) scaled — c.f. (1.51). Thus, the linear system solution is invariant under rescaling of type (1.35). But, an affine transformation of type (1.13) may effect the performance of the linear solver. In order to avoid this, a special row scaling — again internally done within the course of the Newton iteration — turns out to be quite helpful, but does not guarantee affine invariance of the linear system solution. In general, however, small perturbation in solving (1.55) will not destroy the affine invariance of the Newton scheme. The total internal scaling is as follows. Recall that systems of the type

$$J_k \Delta x_k = -F_k$$

have to be solved. Then, systems to be identified with (1.55) read

$$(\overline{D}_k^{-1} J_k D_k)(D_k^{-1} \Delta x_k) = -(\overline{D}_k^{-1} F_k)\,. \tag{1.59}$$

Herein, D_k is given by (1.50.a) and $\overline{D}_k$ is another diagonal matrix

$$\overline{D}_k := \mathrm{diag}(\overline{d}_1, \ldots, \overline{d}_n)\,.$$

Now, let $a_{i,j}$ denote the elements of the column scaled Jacobian $J_k \cdot D_k$, then $\overline{d}_i$ is chosen according to

$$\overline{d}_i := \max_{1\leq j\leq n} |a_{i,j}| \; , \quad i = 1, \ldots, n \, .$$

1.3.5 Rank reduction

As pointed out above, within the QR-decomposition routine of NLEQ2 the rank of a Jacobian J_k may be automatically reduced by checking the corresponding sub-condition number $\mathrm{sc}(J_k)$. Beyond that, there is still another case where a Jacobian rank reduction may be helpful. Recall that for the standard scheme (Algorithm B) the iteration stops, say at x_k, if $\lambda_k = \lambda_{min}$ and $\|\overline{\Delta x}_{k+1}\| \geq \|\Delta x_k\|$. In such a situation — often indicating an iteration towards an (attractive) point $\hat{x}$ where the Jacobian $J(\hat{x})$ is singular — a deliberate rank reduction of J_k may avoid this emergency stop. In order to do so, the ordinary Newton correction Δx_k is recomputed according to (1.58) — but now with a prescribed maximum allowed rank $q := n - 1$. With the new (trial) correction $\Delta x_k^{(q)}$ at hand, the current step is repeated, i.e. a new a priori damping factor $\lambda_k^{(0,q)}$, a new trial iterate $x_{k+1}^{(0,q)} := x_k + \lambda_k^{(0,q)} \Delta x_k^{(q)}$ and a new simplified correction $\overline{\Delta x}_{k+1}^{(0,q)} := -J_k^{(q)+} F_{k+1}^{(0,q)}$ are computed. If the monotonicity check is now successfully passed, the iteration proceeds as usual. Otherwise, the damping factor λ is recomputed using a posteriori estimates $\lambda_k^{(j,q)} (j = 1, 2, \ldots)$. If $\lambda_k^{(j,q)} < \lambda_{min}$ occurs, the maximum allowed rank is reduced again and the repetition of the current steps starts once more. This rank reduction procedure is carried out until natural monotonicity ($\|\overline{\Delta x}_{k+1}^{(j,q)}\| \leq \|\Delta x_k^{(q)}\|$) holds or $q < q_{min}$ $(0 < q_{min} < n)$ is reached.

It should be mentioned, that the application of a rank reduced Newton step means to perform an intermediate Gauß-Newton step. Although, in principle, both methods are algorithmically quite similar, there are some essential theoretical differences (see [7]). As a direct consequence of this fact, the usual estimates for the quantities $[h_k^{(j)}]$, $j = 0, 1, \ldots$ must be modified — but these details are omitted here and can be found in [7]. Rather, the rank reduction strategy will be described in the following informal algorithm. Note that an emergency rank reduction can occur in a step where the rank of J_k has been already reduced because of the sub-condition limit (1.56).

Global Newton scheme with rank strategy (Algorithm R)

Input:

x_0	initial guess for the solution
λ_0	initial damping factor

λ_{min} minimal permitted damping factor
$irank_{min}$ minimal permitted rank q_{min}
$cond_{max}$ maximal permitted sub-condition number $\mathrm{sc}(J_k^{(q)})$
tol, *itmax*, *FCN*, *JAC*, direct linear solver subroutines

Start:

```
        $k := 0$
        $F_k := F(x_k)$
(A)     $J_k := J(x_k)$
        $irank = n$
(B)     $\Delta x_k^{(q)} := -J_k^{(q)+} F_k$
            $q, J_k^{(q)}$ such that $q \le irank$, $\mathrm{sc}(J_k^{(q)}) \le cond_{max}$, $q$ maximal
        $\lambda_k^{(0,q)} := \begin{cases} \min\left\{1, \frac{1}{[h_k^{(0,q)}]}\right\}, & k > 0 \\ \lambda_0, & k = 0 \end{cases}$
        $j := 0, \ \lambda_k^{(-1,q)} := 1$
        $\lambda := \lambda_k^{(0,q)}$
(C)     if $((q = n \vee q = irank_{min}) \wedge \lambda < \lambda_{min} \wedge \lambda_k^{(j-1,q)} > \lambda_{min}) : \lambda = \lambda_{min}$
        if $(\lambda < \lambda_{min})$ then : $irank = irank - 1$
                                 if ($irank <$ irank$_{min}$) : fail exit
                                 proceed at (B)
        $x_{k+1}^{(j,q)} := x_k + \lambda \Delta x_k^{(q)}$
        $F_{k+1}^{(j,q)} := F(x_{k+1}^{(j,q)})$
        $\overline{\Delta x}_{k+1}^{(j,q)} := -J_k^{(q)+} F_{k+1}^{(j,q)}$      ($J_k^{(q)+}$ from (B))
        if $(\|\overline{\Delta x}_{k+1}^{(j,q)}\| \le tol \wedge \|\Delta x_k^{(q)}\| \le \sqrt{tol \cdot 10} \wedge \lambda = 1)$
          then : solution exit $(q = n)$
                 special exit $(q < n)$
        $\lambda_k^{(j+1,q)} := \min\left\{1, \frac{1}{[h_k^{(j+1,q)}]}\right\}$
        $konv := \|\overline{\Delta x}_{k+1}^{(j,q)}\| \le \|\Delta x_k^{(q)}\|$
        if $(konv)$ then : $x_{k+1} := x_{k+1}^{(j,q)}$
                           $F_{k+1} := F_{k+1}^{(j,q)}$
                           $k := k + 1$
                           if ( $k > itmax$ ) : fail exit
                           proceed at (A)
```

```
else:        $j := j + 1$
             if ( $\lambda = \lambda_{min}$ ) : fail exit
             $\lambda := \min\{\lambda_k^{(j,q)}, \frac{\lambda}{2}\}$
             $\lambda = \max\{\lambda, \lambda_{min}\}$
             proceed at (C)
```

Note that the first emergency rank reduction is only activated, if a step with $\lambda = \lambda_{min}$ (for $q = n$) has been already tried. So, in principle, the code NLEQ2 is a real extension of NLEQ1 in the sense, that the (emergency) rank strategy does not replace (even not partially) the usual damping strategy, but it is an additional device in order to extend the convergence domain of the method. However, as the solution of the arising linear systems is done with different algorithms (QR-decomposition with optional rank reduction / LU-decomposition with check for zero pivot), for a given problem at hand, the codes NLEQ1 and NLEQ2 respectively, may show a different algorithmic performance, even if the emergency rank reduction is never activated.

1.3.6 Rank-1 updates

Whenever the Jacobian evaluation dominates the computational work of a (damped) Newton step, then Jacobian savings may speedup the overall performance, even if some additional steps are needed. However, the overall iteration behavior should not be affected too much. Instead of just fixing the Jacobian — which would yield a *simplified Newton iteration* — *Jacobian rank-1 updates* due to Broyden [3], say, of the type

$$\hat{J}_{k+1} := J_k + (F_{k+1} - (1 - \lambda_k)F_k)\frac{\Delta x_k^T}{\lambda_k \|\Delta x_k\|_2^2}\,, \tag{1.60}$$

may be applied. Near the solution x^*, the associated *quasi-Newton iteration* is known to converge *superlinearly.* The decision, to apply (1.60) instead of the usual evaluation of J_{k+1}, is based on an estimate for the expected *Jacobian change* (c.f. (1.22), (1.23))

$$\|J_k^{-1}(J_{k+1} - J_k)\| \leq \lambda_k h_k\,. \tag{1.61}$$

If the condition $\lambda_k h_k \ll 1$ holds, then the new Jacobian J_{k+1} is not worth generating. In order to realize (1.61) (without knowledge of J_{k+1}) one may use the a posteriori estimate $[h_k^{post}]$ — c.f. (1.29.b). Then, if the substitute condition

$$\lambda_k^{prio}[h_k^{post}] \leq \frac{1}{\sigma} \tag{1.62}$$

holds, J_{k+1} may be computed according to (1.60). Empirical values for σ range in the interval $[3, 10]$ — a choice, which turns out to be not very critical. Numerical experience, however, suggests to permit quasi-Newton steps only if the undamped Newton iteration converges, which is indicated by $\lambda_{k-1} = 1\,, \lambda_k^{prio} = 1$. For cases where $\lambda_k^{prio} < 1$ and (1.62) holds, the current (trial) Newton step $k \longrightarrow k+1$ may be repeated with an appropriately increased damping factor λ_k^{post}. Thus, based on (1.62) two extensions of algorithm (B) are realized:

$$\begin{array}{ll} a) & \text{if } (\lambda_{k-1} = 1 \,\wedge\, \lambda_k^{prio} = 1 \,\wedge\, \lambda_k^{prio}[h_k^{post}] < \frac{1}{\sigma}) \text{ then} \\ & \qquad \text{evaluate } \; J_{k+1} \text{ by (1.60),} \\ & \qquad \text{try } \; \lambda_k^{prio} := 1 \text{ for all following steps } \; (\overline{k} > k)\,, \\ b) & \text{if } (\lambda_k^{prio}[h_k^{post}] < \frac{1}{\sigma_2}) : \\ & \qquad \text{repeat step } \; x_k \longrightarrow x_{k+1} \text{ with } \; \lambda_k^{post}\,, \end{array} \tag{1.63}$$

where

$$\sigma = 3\,, \quad \sigma_2 = \frac{10}{\lambda_{min}}\,.$$

Obviously, the update (1.60) is affine invariant. But, concerning scaling invariance, the situation is quite different. Recall the considerations on scaling invariance and scaling covariance made in Section (1.3.2). Straightforward calculation shows, that, in general, $\hat{J}_{k+1}$ is not scaling covariant, i.e. applying update (1.60) for problem (1.36) instead of (1.3), will, in general, not yield the required property $\hat{H}'_{k+1} = \hat{J}_{k+1} \cdot S$ (c.f. (1.37)). Thus, in order to generate scaling covariant quasi-Newton correction $\widehat{\Delta y}_{k+1} \equiv S^{-1}\widehat{\Delta x}_{k+1}$ the update (1.60) must be modified to

$$\hat{J}_{k+1} := J_k + (F_{k+1} - (1-\lambda_k)F_k)\frac{(D_{k+1}^{-2}\Delta x_k)^T}{\lambda_k \|D_{k+1}^{-1}\Delta x_k\|_2^2}\,, \tag{1.64}$$

where D_{k+1} is recursively defined by (1.49), (1.50.a).

In order to exploit the Jacobian rank-1 updates beyond just using (1.60) instead of a standard evaluation $J(x_{k+1})$, one may realize the quasi-Newton step(s) in a special iterative manner. Within such an iterative realization the usual decomposition of $\hat{J}_{k+1}\,,\ \hat{J}_{k+2}\,,\ \ldots$ is saved additionally. Based on the relations (presented for the case $\lambda = 1$)

$$\begin{gathered} \widehat{\Delta x}_{k+1} = -\hat{J}_{k+1}^{-1}F_{k+1} = \frac{\overline{\Delta x}_{k+1}}{1-\alpha_{k+1}} \\ \text{where} \\ \alpha_{k+1} := \frac{(\Delta x_k)^T(\Delta x_{k+1})}{\|\Delta x_k\|_2^2} \end{gathered} \tag{1.65}$$

and

$$\hat{J}_{k+1}^{-1} = (I + \frac{\widehat{\Delta x}_{k+1}(\Delta x_k)^T}{\|\Delta x_k\|_2^2})J_k^{-1} \tag{1.66}$$

all subsequent quasi Newton corrections $\widehat{\Delta x}_{k+l+1}$, $l = 0, 1, \ldots$ can be computed according to an iterative scheme as presented in [7]. The analogous scheme for the scaling covariant update (1.64) — taking also into account the linear system scaling (1.59) — reads as follows. Let Δx_k be the latest ordinary Newton correction, i.e. J_k is the last evaluated Jacobian. Now, assume that Δx_k (to be identified with $\widehat{\Delta x}_k$), all following quasi Newton corrections $\widehat{\Delta x}_{k+l}$, $l = 1, 2, \ldots$, the (diagonal) scaling matrices D_k , D_{k+1} , D_{k+l+1} , $l = 1, 2, \ldots$ and the row scaling matrix $\overline{D}_k$ have been saved (currently accepted iterate is x_{k+l+1}). Then the next quasi-Newton correction $\widehat{\Delta x}_{k+l+1}$ can be computed by

$$\begin{aligned}
a) \quad & \text{solve} \;\; (\overline{D}_k^{-1} J_k D_k) w = -\overline{D}_k^{-1} F_{k+l+1} \\
& v := D_k w \\
b) \quad & i = k+1, \ldots, k+l \\
& \beta_{i-1} := \frac{(D_i^{-1} \widehat{\Delta x}_{i-1})^T (D_i^{-1} v)}{\| D_i^{-1} \widehat{\Delta x}_{i-1} \|_2^2} \\
& v := v + \beta_{i-1} \widehat{\Delta x}_i \\
c) \quad & \alpha_{k+l+1} := \frac{(D_{k+l+1}^{-1} \widehat{\Delta x}_k)^T (D_{k+l+1}^{-1} v)}{\| D_{k+l+1}^{-1} \widehat{\Delta x}_{k+l} \|_2^2} \\
& \widehat{\Delta x}_{k+l+1} := \frac{v}{1 - \alpha_{k+l+1}} \, .
\end{aligned} \tag{1.67}$$

The computational costs for the above scheme are $O(l \cdot n)$. As long as l is of moderate size, the evaluation of $\widehat{\Delta x}_{k+l+1}$ with (1.67) will save computing time — compared to the evaluation by solving $\hat{J}_{k+l+1} \Delta x_{k+l+1} = -F_{k+l+1}$. Furthermore, this iterative update can be applied also for Jacobians with special structure (banded, sparse), whereas the update (1.64) would destroy this structure. However, (1.67) requires additional storage, around $2l$ vectors of length n. In principle, the storage of $D_{k+1}, \ldots, D_{k+l}$ can be avoided, as these values can be recomputed within the course of the iteration (1.67). But this would diminish the computational advantages of (1.67). In lieu of that, one may use the current scaling matrix D_{k+l+1} for a fixed scaling within (1.67). With that, (1.67) is a slight modification of the direct update technique (1.64), but a quite reasonable one, as, in general, D_{k+l+1} approaches the optimal scaling matrix D^*. Thus, one may expect even "better" updates $\hat{J}_i$ by using D_{k+l+1} instead of D_i. Numerical experiments reveal only petty changes due to the replacement $D_i \longrightarrow D_{k+l+1}$ in (1.67).

1.3.7 Refined damping strategies

In order to start the computation an initial damping factor λ_0 is needed. For mildly nonlinear problems, where an undamped Newton scheme converges, the choice $\lambda_0 = 1$ is optimal. Even for highly nonlinear problems $\lambda_0 = 1$ may be used, as the a posteriori damping strategy will correct a too optimistic choice of λ_0. The additional costs are usually low, but note that the a priori factor λ_0 enters via (1.29) into the a posteriori estimate. Furthermore, in critical applications the necessary evaluation of $F(x_0 + \lambda_0 \Delta x_0)$ may cause problems for $\lambda_0 = 1$ (e.g. overflow, invalid operand) as the first undamped trial iterate can be out of bounds. So, a "small" initial guess λ_0 is recommended. As pointed out above, a further parameter of the damping strategy, the minimal permitted damping factor λ_{min}, has to be selected. Besides this, for extremely sensitive problems a recently developed modification (see [7]) of the damping strategy may be selected. Within this *restricted* damping strategy the estimates $[h_k]$ are replaced by $[h_k]/2$. Note that this restricted strategy shows some nice theoretical properties. Furthermore, intended for problems where the function values vary in an extremely large range, a *bounded* λ-update turns out to be quite helpful. Within this additional device a bounding factor f_b limits the decrease and the increase of the damping factors (either between two successive iterations or within the a posteriori loop of Algorithm (B)). Thus, the following condition is claimed for a new damping factor λ^{new}:

$$\lambda^{old}/f_b \leq \lambda^{new} \leq \lambda^{old} \cdot f_b \; , \quad f_b > 1 \tag{1.68}$$

and $f_b = 10$ may be used as a standard factor. This bounded λ-update helps in cases, where a too bad trial value leads to nonsensical estimates $[h_k^{prio}]$, $[h_k^{post}]$. Finally, another additional device is realized in the codes. If, for the current trial value, say $x_k^{(j)}$, the nonlinear function $F(x_k^{(j)})$ can't be evaluated (e.g. too large argument for exp, negative argument for sqrt) this case can be indicated (see Section (2.2.2) below) and the damping factor will be reduced (repeatedly if necessary) according to

$$\lambda^{(j+1)} := \lambda^{(j)} \cdot f_u \; , \quad f_u = \frac{1}{2} \, . \tag{1.69}$$

In order to simplify the choice of λ_0 , λ_{min} and the type of the strategy the following (Table 1.1) specification of problem classes and associated internal selection of parameters is made.

If no user classification of the problem is available the problem is assumed to be "highly nonlinear". Observe that for the classification of a problem not only the nonlinear function F is relevant but also the quality of x_0. The case "linear" is realized in the NLEQ-codes to allow the solution of a linearized problem with the same piece of software as for the original nonlinear problem. Because a specification "mildly nonlinear" at least forces the computation of the first simplified Newton correction, a problem specification "linear" includes the computation of the first ordinary Newton correction with the

problem class	λ_0	λ_{min}	λ-strategy	λ-update
linear	1	—	—	—
mildly nonlinear	1	10^{-4}	standard	standard
highly nonlinear	10^{-2}	10^{-4}	standard	standard
extremely nonlinear	10^{-4}	10^{-8}	restricted	bounded

Table 1.1 Definition of problem classes

update $x_1 := x_0 + \lambda_0 \Delta x_0$ only. Thus, the computational overhead for the solution of a linear problem with the codes is small. Note that in such a case the required tolerance *tol* is ignored, whereas a user selection $\lambda_0 < 1$ is observed.

2. Implementation

2.1 Overview

The algorithms presented in Chapter 1 are implemented in three concurrent pieces of numerical software, the codes NLEQ1, NLEQ1S and NLEQ2 respectively. They belong to the numerical software library CodeLib of the Konrad Zuse Zentrum Berlin, thus they are available for interested users. All codes are written in ANSI standard FORTRAN 77 (double precision) and are realized as subroutines which must be called from a user written driver program. All communication between the codes and the calling program is done via the arguments of the calling sequence of the subroutines. Some (optional) data- and monitor output is written to special FORTRAN units. Besides the interface to the calling program each of the codes has two further interfaces. There are calls from the NLEQ subroutines to a subroutine named FCN which has to evaluate the nonlinear function $F(x)$ of problem (1.3) and there is a call to a subroutine named JAC which must return the system Jacobian $F'(x)$.

The following sections give a short introduction in how to use the NLEQ codes. A detailed documentation including all technical details is part of the code and is not reproduced here. Rather, the relation of the software to the underlying algorithms of Chapter 1 is pointed out and some general comments are made. The codes are implemented with respect to the general design guidelines stated for CodeLib programs and incorporate many common features as calling modes, output structure and others. Internally, the routines use some arrays for working purposes and the user is only required to pass two workspace arrays - one of type integer and one of type real to the subroutine. The routine called by the user makes the division of the user supplied workspace into the appropriate smaller pieces and passes them to the kernel subroutine which realizes the algorithm.

Several options common to the three routines may be passed through an option integer array. Additionally, some internal parameters of the algorithm may be changed by setting certain positions of the workspace arrays to the appropriate nonzero values.

All three programs may be used in a standard mode, i.e., called with a given starting point and prescribed precision, all necessary Newton steps to compute a solution satisfying the required precision are done within this one call. Alternatively, they may be used in a so-called one-step mode. This means, that the program returns control to the calling routine after each performed Newton step. By examining a return value, the calling program can obtain information if there are additional Newton steps needed to get an approximation of the solution fitting the required precision, and can occasionally initiate another successive call of the program — again having the choice between a standard mode and a one step mode call.

Except for the sparse solver, which always needs a user supplied routine for computation of the Jacobian, the Jacobians computation may optionally be done by an internal numerical differentiation routine referencing only the user function which implements the nonlinear system.

Output is written to three different FORTRAN units — the choice of the unit depends on the type of output. One unit, named the error unit, receives all error and warning messages issued by the program. Another unit, named the monitor unit, receives protocol information about the settings of input parameters, some values characterizing the performance of the Newton iteration ($\|\Delta x_k\|$, $\|\overline{\Delta x}_{k+1}\|$, λ_k, $\ldots$), and a final summary information. The iteration vector and characteristic values may be received for each Newton step by the solution unit — with some additional control information, suitable to be processed as input by a graphics program.

Substitution of the linear solvers delivered with the programs only needs the adaptation of two easy to survey subroutines (three for the sparse solver), which establish the interface to the linear solver. Furthermore, the scaling update procedure (1.49) and the used norm for the stopping criterion can be easily modified by just replacing the associated internal routines.

Finally, in addition to the standard routines mentioned above, so-called easy to use interfaces are available, which may facilitate a first use of the codes.

2.2 Interfaces

2.2.1 Easy to use interface

In order to obtain a solution of the nonlinear problem with a minimum of programming effort the user may decide to use the additionally provided "easy to use calls" of the codes NLEQ1 or NLEQ2, namely the subroutines NLEQ1E or NLEQ2E. These subroutines call the corresponding code with the algorithmic standard option settings, and some monitor output will be written to FORTRAN unit 6. The easy to use interface subroutines read as follows:

$$\text{SUBROUTINE}\ \begin{matrix}\text{NLEQ1E}\\ \text{NLEQ2E}\end{matrix}\ (\text{N, X, RTOL, IERR})\ ,$$

where the arguments must be supplied as described below:

N integer, input :
dimension of the nonlinear system to be solved, N $\leq$ 50

X real array(N), in-out :
in : initial guess x^0
out : final (current) approximation x_{out} (x_k) of the solution x^*

RTOL real, in-out :
required (in) / achieved (out) relative accuracy of x_{out}

IERR integer, output :
error flag (IERR > 0 signals an error or warning condition)

In order to use this interface, the nonlinear problem function must be named exactly FCN and its required interface reads:

SUBROUTINE FCN(N, X, F, IFAIL)

where:

N integer, input : see above

X real array(N), input:
the argument X for which F(X) is to evaluate

F real array(N), output:
Must get the vector F(X)

IFAIL integer, in-out
error flag, =0 on input
on output: =0 indicates successful evaluation of F(X)

Remember that the codes NLEQ provide several options. In order to solve highly critical nonlinear problems it may be necessary to use nonstandard option settings. All the available options can be modified using the standard interfaces of the codes, which are described in the subsequent sections.

2.2.2 Standard interfaces

The actual implemented standard user interface subroutines (to the calling program) read:

```
           NLEQ1
SUBROUTINE NLEQ2 ( N, FCN, JAC, X, XSCAL, RTOL, IOPT, IERR,
                   LIWK, IWK, LRWK, RWK)
```

```
SUBROUTINE NLEQ1S ( N, NFMAX, FCN, JAC, X, XSCAL, RTOL,
                    IOPT, IERR, LIWK, IWK, LI2WK, I2WK,
                    LRWK, RWK)
```

Within these calling sequences one may distinguish two groups of arguments: one group for the problem definition and the associated solution requirements and another which refers to the algorithm. The arguments of the first group are:

N integer, input :
  dimension of the nonlinear system to be solved

FCN external subroutine, input :
  evaluation of the nonlinear function $F(x)$

JAC external subroutine, input :
  evaluation of the Jacobian matrix $J = F'(x)$

X real array(N), in-out :
  in : initial guess x^0
  out : final (current) approximation x_{out} (x_k) of the solution x^*

XSCAL real array(N), in-out :
  in : initial user scaling vector xw^u
  out : current internal scaling vector xw^k

RTOL real, in-out :
  required (in) / achieved (out) relative accuracy of x_{out}

Arguments, which apply to NLEQ1S only:

NFMAX integer, input :
  Maximum number of nonzero elements in the Jacobian matrix

In order to make a proper initial setting for these arguments the strong internal coupling of the arguments XSCAL, RTOL, X and even N should be observed. Recall that the internal scaling procedure (1.49) connects X-input, XSCAL and RTOL. Due to (1.50) the current scaling vector xw^k influences the whole algorithmic performance, especially the termination criterion (c.f. (1.20)). Furthermore, an interpretation of the achieved accuracy can only be made in connection with xw^k and x^k and with regard of the internally used norm (c.f. (1.50)) — where N enters. A zero initiation of XSCAL is possible but may lead to an unpleasant behavior of the algorithm — especially together with $x^0 = 0$.

The second group of arguments is:

IOPT integer array(50), in-out :
  selection of options

IERR integer, output :
  error flag (IERR > 0 signals an error or warning condition)

LIWK integer, input :
declared length of the integer work array IWK
LIWK = N+50 for NLEQ1, N+52 for NLEQ2, 8*N+57 for NLEQ1S

IWK integer array(LIWK) - in-out :
first 50 elements: selection of special internal parameters (e.g. limit on iteration count) (in), statistics of the algorithmic performance (out)
elements up from the 51 th : integer workspace for the linear solver

LRWK integer, input :
declared length of the real array RWK
LRWK = (N+NBROY+13)*N+61 for NLEQ1 with full storage mode Jacobian, (2*ML+MU+NBROY+14)*N+61 for NLEQ1 with banded storage mode Jacobian — where ML denotes the lower, MU the upper bandwidth and NBROY the maximum number of possible consecutive rank-1 update steps;
(N+NBROY+15)*N+61 for NLEQ2, 3*NFMAX+(11+NBROY)*N+62 for NLEQ1S.

RWK real array(LRWK), in-out :
first 50 elements: selection of special internal parameters of NLEQ (e.g. starting and minimum damping factor) and possibly of the linear solver (in), actual values of some internal parameters (out)
elements up from the 51 th : real workspace for NLEQ and the linear solver.

Arguments, which apply to NLEQ1S only:

LI2WK integer, input (NLEQ1S only) :
declared length of the "short integer" work array I2WK
LI2WK = Int(7.5*NFMAX)+5*N

I2WK integer array(LI2WK):
"Short integer" workspace. In the current implementation the same integer type as this of IWK, e.g. INTEGER*4.

Besides providing workspace the arguments IOPT, RWK and IWK can be used to control and monitor the performance of NLEQ. This can be done by assigning special values to (a part of) the first fifty elements of these arrays. Note that a zero initiation forces an internal assignment with the default values.

Furthermore, some of these elements will hold helpful information after return — e.g. the minimum needed length of IWK and RWK to solve the given nonlinear problem (LIWK_{min} = IWK(18) , LRWK_{min} = IWK(19)). Observe that the internal default values are chosen according to the suggestions made in Chapter 1. The features of the codes which can be influenced by this type of option selection are described in Section 2.3 below.

Concerning the external subroutines FCN and JAC, from NLEQ the following requirements are made:

Self-evident, FCN/JAC must provide the function value $F(x)$ / $F'(x)$ for that vector x which is input to FCN/JAC. Note that all components of $F(x)$ have to be set by FCN (even constant values) as F (on input) contains no information from a previous call. The same requirement holds for the argument DFDX of JAC. The error flag IFAIL can be used to invoke the heuristic damping device (1.69).

SUBROUTINE FCN(N,X,F,IFAIL)

N see above

X real array(N), input
current (trial) iterate $x_k^{(i)}$

F real array(N), output
function values $F(x_k^{(i)})$

IFAIL integer, in-out
error flag, =0 on input
on output: if < 0, NLEQ terminates
if > 0, invokes heuristic damping device

for NLEQ1 and NLEQ2:

SUBROUTINE JAC (N,LDJAC,X,DFDX,IFAIL)

N see above

LDJAC integer, input
leading dimension of the array DFDX

X see above

DFDX real array(LDJAC,N), output
Jacobian matrix

IFAIL integer, in-out
error flag, =0 on input
on output: if < 0, NLEQ terminates

for NLEQ1S:

SUBROUTINE JAC (N,X,DFDX,IROW,ICOL,NFILL,IFAIL)

N see above

X see above

DFDX real array(NFILL), output
real values of the Jacobian

IROW integer array(NFILL)
row indices of the Jacobian

ICOL integer array(NFILL)
column indices of the Jacobian

NFILL integer, in-out
in: dimension of DFDX, IROW and ICOL
out: Number of nonzeros stored in DFDX, IROW and ICOL

IFAIL see above

2.3 Options

Though the underlying algorithm of the codes is self-adaptive in the sense that the damping factor λ is automatically adapted to the problem at hand, there are still some algorithmic parameters and variants open for an adjustment by the user. In general, the influence on the overall performance is not dramatic but in special applications a skillful matching may increase efficiency and robustness drastically. Besides these algorithmic options some other useful options, e.g. output generation, are available for the user. As pointed out in the preceding section the adaptation can be easily performed by assigning special values to specific positions of the arrays IOPT, IWK and RWK. As far as possible, the input is checked for correctness.

Jacobian generation and storage mode

Within the codes NLEQ1 and NLEQ2 the Jacobian is either evaluated via the user subroutine JAC or, alternatively, approximated by an internal subroutine using numerical differentiation. This selection may be done by setting IOPT(3). The choice IOPT(3)=1 means that the user subroutine JAC will be called to get the Jacobian. A value IOPT(3)=2 selects the numerical approximation of the Jacobian by an internal subroutine. Setting IOPT(3)=3 causes the use of another internal numerical differentiation routine which additionally uses a feedback strategy to adapt the finite difference disturbance.

The decision which storage mode for the Jacobian has to be used can be made by setting IOPT(4). If IOPT(4)=1 is selected, NLEQ1 awaits that the Jacobian is supplied in band storage mode and calls the appropriate linear solver from LINPACK (DGBFA/DGBSL). Using IOPT(4)=0 implies that the Jacobian is supplied in full storage mode, thus the corresponding linear solver subroutines DGEFA/DGESL from LINPACK will be called for solving the arising linear systems. Note that in case of selecting numerical approximation of the Jacobian by IOPT(3)$\neq$1, the mode of the numerical differentiation (full or banded) is selected according to the value of IOPT(4).

Scaling

The internal update of the user scaling (weighting) vector XSCAL according to (1.49) can be switched off by setting IOPT(9)=1. Note that the initial check (1.49a) is done in any case. The problem classification which may influence the performance of (1.49a) is done via IOPT(31) — see below. Furthermore, recall that the arising linear systems are also row-scaled (c.f. (1.59)). This scaling may be switched off by setting IOPT(35)=1 — by default it is switched on.

Output generation

The amount of output produced by the codes is internally controlled by the actual values of some output flags and directed to associated FORTRAN units. In order to monitor the algorithmic performance of the code, the internal flag MPRMON can be modified by the user by setting the associated element of the IOPT array (MPRMON corresponds to IOPT(13) and the associated output unit LUMON to IOPT(14)). An user assignment IOPT(13)=0 produces no monitor print output, whereas a setting IOPT(13)=3 will generate a detailed iteration monitor, e.g. the unscaled norm (1.32) $\|F(x_k)\|_{rms}$ (where F denotes the problem function introduced in (1.3) and x_k the Newton iterate), the scaled norms (1.50.b) of the current Newton corrections $\|\Delta x_k\|$, $\|\overline{\Delta x}_{k+1}\|$ and the current damping factor λ_k are written to FORTRAN unit IOPT(14). Similar flag/unit pairs are available for error/warning printout, data output and time monitor output — c.f. Table 2.1.

Option	Selection	Range	Default	Unit
error/warning messages	IOPT(11)	0-3	0	IOPT(12)
Newton iteration monitor	IOPT(13)	0-6	0	IOPT(14)
Solution output	IOPT(15)	0-2	0	IOPT(16)
Time monitor	IOPT(19)	0-1	0	IOPT(20)

Table 2.1 Options for output generation

Modification of the damping strategy

The damping strategy of the Newton scheme can be partially modified by the user. First, a general problem classification can be made by the user by setting the parameter NONLIN (c.f. Table 1.1) to the desired value (1 – linear problem, 2 – mildly nonlinear, 3 – highly nonlinear, 4 – extremely nonlinear). But besides this, the values of some special internal parameters can be adapted separately. Thus, the initial and minimum damping factors λ_0, λ_{min}, as well as the bounding factor f_b of (1.68), can be set individually. Furthermore, the bounded λ-update may be switched on or off separately, whereas the general type of the damping strategy (standard or restricted) still depends on NONLIN. Recall that λ_{min} appears in the emergency stopping criterion (1.54). Thus, the emergency exit of all codes as well as the emergency rank reduction device of NLEQ2 are directly effected by a modification of λ_{min}. Finally, the decision whether a (successful) step is repeated because the a posteriori damping factor is greater than the a priori estimate can be influenced by specifying σ_2 — c.f. (1.6.b). Note that the default value for σ_2 inhibits a step repetition. An overview on the options related to the damping strategy of the Newton iteration is given in Table 2.2.

Option	Selection	Range	Default
problem classification	IOPT(31)	0-4	3
bounded damping strategy	IOPT(38)	0-2	see Table 1.1
Newton iteration limit	IWK(31)	≥ 1	50
initial damping factor	RWK(21)	≤ 1	see Table 1.1
minimum damping factor	RWK(22)	≤ 1	see Table 1.1
bounding factor	RWK(20)	> 1	10.0
step repetition	RWK(24)	≥ 1	$10/\lambda_{min}$

Table 2.2 Options for the damped Newton iteration

Rank-1 updates

The ordinary Newton corrections computed within each iteration step may be replaced (under certain conditions, c.f. Section 1.3.6) by quasi Newton corrections. Recall that these corrections are computed without needing a Jacobian evaluation or LU-decomposition — thus the activation of this option (IOPT(32)=1) may save computing time. However, as within consecutive iterative Broyden steps all previous quasi Newton corrections are needed for computing the next one (c.f. (1.67)), storage must be reserved in order to keep them. The maximum number of consecutive Broyden steps may be changed by setting IWK(36) to the desired value — the default is chosen such that there will be in general no more storage needed as for one Jacobian, except that a minimal value 10 is allowed (i.e. IWK(36) = $\max(n, 10)$ if the Jacobian is stored in full mode). The decision criterion (1.63.a) whether Broyden steps are done, depends on the parameter σ (default is 3), which may be changed by setting RWK(23) to the requested value.

Controlling the rank strategy of NLEQ2

Recall that the activation of the reduction strategy is mainly determined by the minimal permitted damping factor λ_{min} and the maximum allowed subcondition number $cond_{max}$ — c.f. algorithm (R) of Section 1.3.5. As mentioned above, the value of λ_{min} is influenced by IOPT(31) or can be set directly by specifying RWK(22). The subcondition limit can be changed by setting RWK(25) to a preferred value (default is $cond_{max} = 1/epmach \approx 10^{16}$, where *epmach* denotes the relative machine precision). Furthermore, in each Newton step the Jacobians rank is initially assumed to be full. But, for special cases, the maximum initial rank of the Jacobian at the starting point x_0 may be limited to some smaller value by setting IWK(32) $< n$.

Sparse linear algebra options of NLEQ1S

As the efficient generation of a sparse Jacobian matrix by numerical differentiation is an independent, nontrivial task, no internal differentiation routine is available within the current version of NLEQ1S. Thus, values and patterns of the Jacobian must be provided in the user subroutine JAC, however, eventually generated by a user written approximation scheme. Note that the efficiency of the sparse linear system solution clearly depends on whether the sparse pattern of the Jacobian changes within the course of the Newton iteration. To make this clear, the general strategy for the sparse linear system solution is now shortly summarized. The very first Jacobian matrix is factorized with the so-called *Analyze-Factorize* routine MA28A — which means that a conditional pivoting is performed in order to minimize the fill-in but still ensure numerical stability. This optimization process — which can be rather time consuming — is controlled by a special threshold value (*thresh1*). Numerical experience suggests the choice *thresh1* = 0.01, which represents a good compromise between fill-in minimization and stability preservation. Now, the matrices arising in the subsequent Newton steps may be decomposed with the fast *Factorize* routine MA28B — which works on the pattern provided by MA28A. Hence, these matrices must have the same pattern as the first Jacobian matrix, but the numerical values may have changed. The numerical stability of such factorizations can be monitored by checking the arising pivot ratios. If such a ratio is beyond a certain threshold value (*thresh2*) the *Analyze-Factorize* routine is called again.

Now, to indicate an always fixed pattern to NLEQ1S, IOPT(37)=1 must be set. Otherwise, (and that is the standard option) in order to check whether a MA28B call is possible, the pattern of a new Jacobian is internally compared to that pattern which was used for the latest MA28A call. Thus, if the sparse structure of all Jacobians provided by JAC is known to be the same, additional storage and computing time can be saved by setting IOPT(37)=1.

In principle, the performance of the sparse linear solver can be influenced by changing certain parameters of the common blocks MA28ED, MA28FD — for more information refer to [18]. Some of these values (e.g. *thresh1* $= 10^{-2}$,*thresh2* $= 10^{-6}$) are already adapted in the internal subroutine NISLVI of NLEQ1S.

On return from NLEQ1S, the positions IWK(46), IWK(47) contain information about minimal integer workspace needed by the sparse linear solver, and IWK(48), IWK(49) are holding the maximum count of some compression operations done by the solver on the integer workspace — thus giving a measure for the efficiency of the linear solver. If this count seems to be too large, the integer workspace should be increased. Finally, IWK(50) contains an estimate of the latest Jacobian rank.

For each LU-decomposition, the current values of the informal integers mentioned above and, additionally, the number of nonzeros and the percentage of nonzeros in the Jacobian may be obtained from a special output stream — the linear solver output. This stream is activated by setting IOPT(17)=2, and the FORTRAN unit where to send the output is chosen by setting IOPT(18) to the desired number.

Convergence order monitor

If the Newton iterates x_k approach the solution x^* and $J(x^*)$ is not singular, the iteration converges quadratically, or, for quasi Newton iteration, at least superlinearly. However, due to roundoff errors, the Newton corrections $\Delta x_k = -J(x_k)^{-1}F(x_k)$ are only computed up to a certain precision. In praxis, this fact may lead, at least, to a slowdown of the convergence, or, even worse, convergence may be lost if the required accuracy *rtol* is chosen too stringent. In such a case, if no further improvement of the accuracy of x_k is possible, it is desirable that the Newton iteration stops (with an error indicator returned). Therefore, an optional convergence order estimate and associated optional stop criteria are available.

As soon as $\lambda_k = 1$ holds, the convergence rate L_k and the convergence order α_k of the current step i.e.

$$\|\Delta x_{k+1}\| \doteq L_k \|\Delta x_k\|^{\alpha_k}$$

are tried to estimate. Omitting details, the convergence is defined to be

$$\begin{array}{ll} \textit{superlinear} \text{ if } & \tilde{\alpha}_k \geq 1.2 \\ \textit{quadratic} \quad \text{ if } & \tilde{\alpha}_k \geq 1.8\,, \end{array}$$

where $\tilde{\alpha}_k$ is an estimate for α_k. Now, a *convergence slowdown* is said to appear in the k-th Newton step (with $\lambda_k = 1$), if superlinear or quadratic convergence has been already achieved in a previous Newton step, but now

$$\tilde{\alpha}_k < 0.9$$

holds.

The action taken as consequence of a convergence slowdown depends on the value of the *convergence order monitor option* IOPT(39). If this value is set to

2 or 0, the *weak stop* option is chosen. This means that the Newton iteration is terminated and an error indicator (IERR=4) is returned, if in some Newton step k_0 convergence slowdown has been detected and if in a (possibly later) step k_1 the monotonicity test fails to be satisfied — while all intermediate damping factors $\lambda_{k_0}, \ldots, \lambda_{k_1}$ have taken a value 1. If IOPT(39)=3 is set, the *hard stop* option is selected, which means that a convergence slowdown leads to an immediate termination of the iteration and return with the error indicator as above. An error (warning) indicator IERR=5 is returned, if the usual convergence criterion is fulfilled, but no quadratic or superlinear convergence has been detected so far. This proceeding takes into account, that the error estimate (1.52) may be rather poor for such cases. Finally, the choice IOPT(39)=1 totally switches off the convergence order monitor.

One step mode

Within special applications it may be useful to perform the Newton iteration step by step, i.e. the program control is passed back to the calling program after one Newton step. This mode can be selected by setting the mode flag IOPT(2)=1. In order to distinguish the first call — certain initializations and checks are made by NLEQ — and successive calls an associated flag IOPT(1) is used. IOPT(1)=0 indicates that this is the first call whereas IOPT(1)=1 indicates a successive call. Note that the codes internally set IOPT(1)=1 on return if it was called in stepwise mode. Furthermore, the error flag IERR is set to -1 as long as the stopping criterion (1.20)+(1.21) does not yet hold. As an example for an application of this option, just this internal stopping criterion can be substituted by a user defined one as the continuation of the iteration is under user control.

Time monitor

In order to get more detailed information concerning the performance of the codes (including the user supplied problem subroutines), a time monitor package, which is designed for time measurements of multiple program parts, is added to the NLEQ packages and may easily be used. Before the first use of the monitor package for time measurements, because of the machine dependency of that stuff, the user has to adapt the subroutine SECOND in such a way that on output the only argument of this routine contains a "time stamp", measured in seconds. As distributed, SECOND is a dummy routine which always returns zero to this argument.

The monitor may be turned on by setting IOPT(19)=1. Its output will be written to the FORTRAN unit IOPT(20). The printout includes the total time used for solving the nonlinear problem and detailed statistics about the following listed program sections: factorization of the Jacobian, solution of the linear system (forward/backward substitution), subroutines FCN and JAC and NLEQ monitor- and data output. Statistics about the remaining code pieces are summarized as the item "NLEQ1" respectively "NLEQ1S", "NLEQ2". For each section and for the remaining code the following statistics are printed out before the NLEQ subroutine exits: number of calls of the code section, time used for all calls, average time used for one call, percentage of time related to the total time used and related to the summarized time of the specific parts measured.

Of course, the time monitor package distributed with the NLEQ codes may be used as a separate piece of software within any program to print out statistics as described above for program sections which may be arbitrary defined by the user. These sections may even be nested — as applied within the CodeLib software package GIANT. The use of the monitor package is described in detail within its code documentation.

Machine dependencies

One aspect of the modular structure of the codes is the fact that the machine dependencies of the whole program are concentrated in two FORTRAN modules: the timing subroutine SECOND and the machine constants double precision function D1MACH. The routine SECOND is only called when the time monitor is switched on and therefore described within the context of this monitor. The function D1MACH returns, dependent on its input argument, typical machine dependent double precision constants, such as the machine precision, the smallest positive and the largest real number. It consists of comments which contain sets of FORTRAN data-statements with the appropriate constants for several different machine types. Before using the NLEQ code on a machine different from SUN, the user should comment out the data-statements related to these machines and uncomment the set of data-statements which is appropriate for the target machine.

3. Numerical Experiments

In this Chapter some computations, illustrating the behavior of the damped affine invariant Newton techniques described in Chapter 1, are presented. The discussion will focus on the performance of the codes NLEQ1, NLEQ2 and NLEQ1S. But, in order to allow a comparison with other available software, the results of some experiments with the well-known and widely used codes HYBRJ1/HYBRD1 [23] from MINPACK are given additionally. Note that the corresponding nonlinear equation software in NAG and IMSL is derived from these codes, which are of Levenberg/Marquardt type. Although the comparison shows some interesting results, this Chapter should not be misunderstood as a general software evaluation or a proper comparison test.

3.1 Test problems

In order to create a certain "basic test set" some quite different test problems from literature are put together. The first part of the test set consists of 14 examples, which have been used already in [22] as a test set for nonlinear equation solvers. One may criticize some of these problems as they are artificially generated test problems with properties, which are not typical for real life applications. Nevertheless, all these examples are part of the basic test set. However, in contrast to the tests made in [22, 19], each problem is used only once, i.e. for all test problems with a variable dimension, $n = 10$ is set (except the Chebyquad problem, where $n = 9$ is chosen) and only the standard starting point is used.

Next, the three chemical equilibrium problems from [19] are added. As no initial guess x^0 is given in [19], they are used with the initial values $x_i^0 = 1\,,\ i = 1, \ldots, n$. For the parameter R in the third example a choice $R = 10$ is made.

Three nice application problems (distillation column test problem) have been presented by R. Fletcher [17] in [21]. Two of them (Hydrocarbon-6, Methanol-8) are used in the basic test set, whereas the larger one (Hydrocarbon-20) is used for testing NLEQ1S — see Section 3.3.

In order to demonstrate the band mode facility of NLEQ1, a discretized 1D-PDE problem is used. This example (SST Pollution) is the stationary version of example F from [25]. For discretization, a simple finite difference scheme on an equidistant grid of size $n_x = 101$ is used. With that, a nonlinear system of dimension $n = 404$ arises. The results of solving this problem can be found in Section 3.3. However, the 0-dimensional problem ($n = 4$) of finding the chemical equilibrium neglecting diffusion is part of the basic test set. Here, the parameter SST1 is set to 3250. As an initial guess the quite poor values

$x_1^0 = 10^9$, $x_2^0 = 10^9$, $x_3^0 = 10^{13}$, $x_4^0 = 10^7$ are used.

A quite hard problem is selected from [20]. In this paper, the boundary conditions for a 2D semiconductor device simulation turn out to be

$$\begin{aligned}
f_1 &= \exp(\alpha(x_3 - x_1)) - \exp(\alpha(x_1 - x_2)) - D/n_i &&= 0\\
f_2 &= x_2 &&= 0\\
f_3 &= x_3 &&= 0\\
f_4 &= \exp(\alpha(x_6 - x_4)) - \exp(\alpha(x_4 - x_5)) + D/n_i &&= 0\\
f_5 &= x_5 - V &&= 0\\
f_6 &= x_6 - V &&= 0
\end{aligned} \tag{3.1}$$

where

$$\begin{aligned}
\alpha &= 38.683\\
n_i &= 1.22 \cdot 10^{10}\\
V &= 100\\
D &= 10^{17}
\end{aligned}$$

In combination with the starting point

$$x_i^0 = 1 \text{ for } \ i = 1, \ldots, 6$$

one has an extremely nonlinear and sensitive problem.

Finally, another artificial test problem is added to the basic test set. The equations read

$$\begin{aligned}
f_1 &= \exp(x_1^2 + x_2^2) - 3 &&= 0\\
f_2 &= x_1 + x_2 - \sin(3(x_1 + x_2)) &&= 0
\end{aligned} \tag{3.2}$$

and the initial value is given by

$$x_1^0 = 0.81 \, , \; x_2^0 = 0.82$$

The usual name, an identifying abbreviation, the dimension n and the reference to get a full documentation are given in Table 3.1 for all problems of the basic test set. Herein, a mark "†" in the column Ref. means a reference to this paper. It should be mentioned, that all test functions are implemented in such a way, that "bad" input values x (e.g. invalid argument for log, sqrt) cause a return to the calling routine with the error indicating return code IFAIL=1 for the codes NLEQ1 and NLEQ2 and IFAIL=-1 for the code HYBRJ1.

No	Name	Abbrev.	n	Ref.
1	Rosenbrock function	Rosenbr	2	[22]
2	Powell singular function	Powsing	4	[22]
3	Powell badly scaled function	Powbad	2	[22]
4	Wood function	Wood	4	[22]
5	Helical valley function	Helval	3	[22]
6	Watson function	Watson	10	[22]
7	Chebyquad function	Cheby9	9	[22]
8	Brown almost-linear function	Brallin	10	[22]
9	Discrete boundary value function	Discbv	10	[22]
10	Discrete integral equation function	Discint	10	[22]
11	Trigonometric function	Trigo	10	[22]
12	Variably dimensioned function	Vardim	10	[22]
13	Broyden tridiagonal function	Broytri	10	[22]
14	Broyden banded function	Broybnd	10	[22]
15	Chemical equilibrium 1	Chemeq1	2	[19], †
16	Chemical equilibrium 2	Chemeq2	6	[19], †
17	Chemical equilibrium 3	Chemeq3	10	[19], †
18	SST pollution, 0 dimensional	SST0D	4	[25], †
19	Distillation column - Hydrocarbon 6	Dchyd6	29	[21]
20	Distillation column - Methanol 8	Dcmeth8	31	[21]
21	Semiconductor boundary condition	Semicon	6	[20], †
22	Exponential/sine function	Expsin	2	†

Table 3.1 Problems of the basic test set

3.2 Numerical results for the basic test set

All experiments of Chapter 3 have been carried out on a Macintosh II personal computer equipped with a MC68881 coprocessor using the Language Systems FORTRAN 2.0 compiler with the options "-mc68020" and "-mc68881" under MPW 3.0 (System 6.0.3). For the experiments of this Section the optimization level "-opt=1" has been specified (due to buggy compilation when trying higher optimization levels), while all codes for the special experiments of Section 3.3 have been compiled using "-opt=2".

If not otherwise stated, the default options and parameters of the codes NLEQ1, NLEQ2 are active, which means, e.g. that all problems are specified as being highly nonlinear (i.e. $\lambda_0 = 10^{-2}$, Broyden update switched off, $cond_{max} = 1/epmach \approx 9 \cdot 10^{-17}, \ldots$). For the required relative tolerance of the solution a value of RTOL $= 10^{-10}$ is prescribed. Recall that this value has to be interpreted in connection with the associated scaling vector *xscale*. The internal scaling update is done according to (1.49) and an initial value of XSCALE $\equiv 10^{-6}$ is used for all problems — thus, a change of RTOL does not affect the internal scaling. Except for this special choice and the fact, that the analytical Jacobian is used by the codes, the call of the codes corresponds to a call of the easy to use interfaces NLEQ1E/NLEQ2E mentioned in Section 2.2.1. For that reason, the easy to use MINPACK code HYBRJ1 (HYBRD1 in case of numerical differentiation) is selected in order to have a rough comparison to another state-of-the-art code. The results of solving the basic test set with the "standard" codes NLEQ1/NLEQ2/HYBRJ1 (N1/N2/HY) are summerized in Table 3.2.

example		#F			#J			CPU (s)		
no	name	N1	N2	HY	N1	N2	HY	N1	N2	HY
1	Rosenbr	6	6	16	5	5	3	0.63	0.72	0.17
2	Powsing	54	54	*f4	53	53	–	1.23	1.45	2.33
3	Powbad	16	16	170	15	15	6	0.40	0.42	1.97
4	Wood	19	19	87	16	16	2	0.52	0.60	1.83
5	Helval	12	12	19	11	11	3	0.38	0.42	0.28
6	Watson	21	21	78	19	19	6	8.52	9.07	12.87
7	Cheby9	9	9	25	8	8	2	0.77	0.93	1.67
8	Brallin	*f3	67	16	–	34	3	0.28	5.03	1.22
9	Discbv	5	5	7	4	4	1	0.43	0.58	0.50
10	Discint	5	5	7	4	4	1	0.60	0.72	0.58
11	Trigo	*f3	16	*f4	–	14	–	0.40	1.55	4.95
12	Vardim	16	16	22	15	15	1	1.22	1.68	1.58
13	Broytri	7	7	13	6	6	1	0.53	0.77	0.92
14	Broybnd	8	8	23	7	7	1	0.62	0.90	1.67
15	Chemeq1	8	8	*f2	7	7	–	0.28	0.32	3.38
16	Chemeq2	20	20	37	19	19	3	0.75	0.93	1.23
17	Chemeq3	12	12	40	11	11	2	0.90	1.27	2.92
18	SST0D	22	22	*f4	21	21	–	0.62	0.72	0.27
19	Dchyd6	7	7	16	6	6	2	3.07	7.37	10.08
20	Dcmeth8	6	6	16	5	5	1	2.92	7.20	9.83
21	Semicon	*f3	*f3	*fp	–	–	–	0.25	0.27	0.05
22	Expsin	13	13	*fp	11	11	–	0.35	0.37	0.02

Table 3.2 Results of NLEQ1/NLEQ2/HYBRJ1 solving the basic test set

Usually, the number of function evaluations and Jacobian calls needed to solve a problem are listed in such tables in order to indicate the efficiency of the codes. But, it must be pointed out, that the total amount of work may be dominated by internal matrix operations (e.g. LU- or QR-decomposition, Broyden updates), thus the overall computing times are presented additionally. A comparison of efficiency by just counting J and F calls — especially for different methods, where the distribution of the computational work is quite different — may lead to totally wrong conclusions. In any case, the most important measure for the quality of a code is its robustness — mainly in the sense of "a code may fail but must not lie" (B. Parlett), but also "better early return because of 'lack of progress' than wasting computing time until maximum iteration limit is reached" [19]. In this respect, all codes show a quite high degree of robustness. To examine this, a "true" solution $\tilde{x}^*$ is computed by an a posteriori call to NLEQ2, with RTOL $= 10^{-12}$, XSCALE $\equiv 10^{-10}$, and the approximate solution from the test run (x^{num}) is checked by computing the actually achieved accuracy

$$acc := \max_{1\le i\le n} \left| \frac{x_i^{num} - \tilde{x}_i^*}{\max\{10^{-6}, |\tilde{x}_i^*|\}} \right| . \qquad (3.3)$$

Note that neither NLEQ1/2 nor HYBRJ1 use this norm as the implemented stopping criterion. Nevertheless, no code claims "solution found" without really having a good approximation x^{num} for x^*. The worst case, $acc = 5 \cdot 10^2 \cdot rtol$ ($acc = 3.5 \cdot 10^1 \cdot rtol$), occurs for HYBRJ1 in example *Watson* (*Chemeq2*) while for all other numerical solutions $acc \approx rtol$ holds.

If one judges the quality of the codes by counting the fail runs, the Newton code with additional rank reduction device (NLEQ2) turns out to be the "best" (only 1 fail), followed by NLEQ1 (3 fails) and HYBRJ1 (6 (5) fails). While the NLEQ codes only stop due to f3 := "too small damping factor" , HYBRJ1 fails twice due to fp := "error return from user problem function (invalid x)", 3 times due to f4 := "bad iteration progress" and once due to f2 := "maximum iteration limit". It is worth mentioning, that for one f4-fail (*Powsing*) the final iterate of HYBRJ1 is already the solution (according to (3.3)). This behavior is a consequence of the fact that the stopping criterion of HYBRJ1 runs into problems if $x^* \equiv 0$ occurs. Furthermore, one fail of HYBRJ1 (example *SST0D*) can be removed by just restarting the code — a proceeding which is suggested by the f2 or f4 messages. Restarting NLEQ1 with $\lambda_{min} = 10^{-8}$ removes the failure for problem *Semicon*. Finally, note that the above ranking of the codes can be easily changed (reinforced) by adding e.g. example *Brallin* with dimensions $n = 30$, $n = 40$ — as in the test set used in [22] — (adding e.g. problem *SST-0D* with a parameter value SST1 = 360) to the basic test set.

Analyzing the performance and efficiency of the codes in more detail, the comparison of NLEQ1 with NLEQ2 nicely shows that NLEQ2 is an extension of NLEQ1 (less failures, same J and F count for solved problems). But this fact must be paid by more computing time (+35% in average, +150% for

Dcmeth8 ($n = 31$!)), because a QR-decomposition is more costly than a LU-decomposition. A comparison of NLEQ1 with HYBRJ1 (only solved problems) shows that HYBRJ1 needs (in average) twice as much F-evaluations as NLEQ1, but NLEQ1 needs 4 times more Jacobian calls. Concerning CPU-time, NLEQ1 is mostly faster than HYBRJ1 (1:2 in average, 1:5 for *Powbad*, but 4:1 for *Rosenbr*). Due to the low Jacobian count of HYBRJ1 one may expect advantages if the Jacobian must be generated by (internal) numerical differentiation. Rerunning the test set with this option (HYBRD1 instead of HYBRJ1) shows, that NLEQ1 is still faster (now 1:1.5) and the ratio of the F-counts (including all calls for the generation of the Jacobian) turns out to be 1.2 : 1. However, there are two additional fails for NLEQ1 with numerical differentiation (problems *Watson*, *Vardim*). But this is no contradiction to the fact, that the Newton scheme (B) is, in principle, insensitive to approximation errors in the Jacobian. Rather, the numerical differentiation procedures of NLEQ1 are realized in scaled form (XSCALE enters) and this feature, which helps in ill-scaled problems, invokes some trouble (at the very beginning of the iteration) in cases, where some or all components of x^0 are zero and XSCALE is small (10^{-6}). Apart from this exception, a very good accordance of both variants (with/without analytical Jacobian) can be stated for all methods (NLEQ1, NLEQ2, HYBRJ1/D1).

A doubtless interesting question is now, how far the theoretical affine invariance and scaling invariance property of the Newton algorithms carry over to the performance of the codes. Recall that there are some sources of trouble, e.g. the linear system solution, the scaling update thresholds and the finite length of the mantissa. The affine invariance property is now checked by solving the affine transformed test problems

$$G(x) := A \cdot F(x) = 0\,, \qquad x^0 \text{ as before}$$

where

$$\begin{aligned} A &:= \operatorname{diag}(a_1, \ldots, a_n) \\ a_{2i-1} &:= 8^{-(4-(i-1)\mathrm{mod}4)} && \text{for } 1 \le 2i-1 \le n \\ a_{2i} &:= 8^{4-(i-1)\mathrm{mod}4} && \text{for } 1 \le 2i \le n\,. \end{aligned} \tag{3.4}$$

The degree of practical scaling invariance is examined by solving transformed functions with regauged variables, i.e.

$$H(y) := F(S \cdot y) = 0\,, \qquad y^0 := S^{-1}x^0$$

where

$$\begin{aligned} S &:= \operatorname{diag}(s_1, \ldots, s_n) \\ s_{2i-1} &:= 10^{4-(i-1)\mathrm{mod}4} && \text{for } 1 \le 2i-1 \le n \\ s_{2i} &:= 10^{-(4-(i-1)\mathrm{mod}4)} && \text{for } 1 \le 2i \le n\,. \end{aligned} \tag{3.5}$$

Recall that the solution of these problems is given by $y^* = S^{-1}x^*$.

The performance changes of NLEQ2 and HYBRJ1, solving (3.4) and (3.5) instead of the original formulation (1.3), are summarized in Tables (3.3) and (3.4). As the performance changes of NLEQ1 are even less than the changes of NLEQ2 these results are omitted. Within the tables an unchanged performance is indicated by an entry "—" (or "=f*i*" in case of failure with the same return code *i* as for the original problem). Additional fail runs are marked by "+f*i*" (*i* indicates the reason for failure) and runs where the error message changes are marked by "f*i*→*j*". The result of this experiment clearly shows the advantages of using a code with invariance properties. Table 3.3 makes evident the total invariance of both the Newton codes with respect to the affine transformation (3.4), whereas Table 3.4 discloses the limit of practical scaling invariance.

example		ret. code		#F		#J	
no	name	N2	HY	N2	HY	N2	HY
1	Rosenbr	—	+f4	—	+49	—	+15
2	Powsing	—	=f4	—	−5	—	+3
3	Powbad	—	+f2	—	+130	—	−4
4	Wood	—	+f4	—	−13	—	+9
5	Helval	—	+f4	—	−8	—	−1
6	Watson	—	+f2	—	+1022	—	—
7	Cheby9	—	—	—	−1	—	—
8	Brallin	—	—	—	+2	—	—
9	Discbv	—	—	—	—	—	—
10	Discint	—	—	—	—	—	—
11	Trigo	—	=f4	—	+111	—	+15
12	Vardim	—	—	—	—	—	—
13	Broytri	—	—	—	—	—	—
14	Broybnd	—	—	—	—	—	—
15	Chemeq1	—	f2→4	—	−232	—	+14
16	Chemeq2	—	—	—	—	—	+2
17	Chemeq3	—	—	—	−11	—	+3
18	SST0D	—	=f4	—	+18	—	+5
19	Dchyd6	—	—	—	—	—	—
20	Dcmeth8	—	—	—	—	—	—
21	Semicon	=f3	=fp	—	—	—	—
22	Expsin	—	=fp	—	—	—	—

Table 3.3 Performance changes due to affine transformation of equations

Solving the rescaled problems (3.5) instead of (1.3), the code NLEQ2 shows one more deviation than the code NLEQ1 (additional fail for problem *Brallin*), which nicely reflects the fact that the rank strategy of NLEQ2 is affected by a rescaling of type (3.5). In contrast to the nearly invariant performance of the damped Newton codes, the performance of the hybrid code HYBRJ1 is strongly impaired by the transformations (3.4) and (3.5) respectively. Quite a large number of additional fail runs occur whereas for the solved problems the degree of the performance change is rather insignificant. Further experiments with other affine and scaling transformations confirm the observations mentioned above. Of course, the affine invariance property of the Newton codes can be destroyed by choosing the matrix A in (3.4) in such a way that the Jacobian $G'(x)$ is (nearly) numerically singular.

example		ret. code		#F		#J	
no	name	N2	HY	N2	HY	N2	HY
1	Rosenbr	—	—	—	+5	—	−1
2	Powsing	—	=f4	+13	−11	+13	−1
3	Powbad	—	—	—	−94	—	−4
4	Wood	—	—	—	−10	—	+2
5	Helval	—	+f4	−1	—	−1	—
6	Watson	—	—	+1	+20	+1	+8
7	Cheby9	—	+f4	—	+6	—	+3
8	Brallin	+f3	+f4	−30	−3	−20	−1
9	Discbv	—	—	—	—	—	—
10	Discint	—	—	—	—	—	—
11	Trigo	—	=f4	—	−49	—	−5
12	Vardim	—	—	—	+1	—	—
13	Broytri	—	—	—	—	—	—
14	Broybnd	—	—	—	—	—	—
15	Chemeq1	—	f2→4	—	−21	—	+5
16	Chemeq2	—	—	—	−5	—	—
17	Chemeq3	—	—	—	−12	—	+1
18	SST0D	—	=f4	—	—	—	—
19	Dchyd6	—	—	+1	+1	+1	—
20	Dcmeth8	—	—	—	—	—	—
21	Semicon	=f3	=fp	—	—	—	—
22	Expsin	—	=fp	—	—	—	—

Table 3.4 Performance changes due to rescaling of variables

Besides the experiments presented so far, a lot of further testing has been carried out, especially experiments with non-standard options (Broyden updates on, modified damping,...). The results can be summarized as follows. The Newton codes show an extreme robustness, but — for the basic test set — only minor performance changes due to (reasonably) modified options. The latter fact is not surprising, as the problems of the basic test set are not much large and, most of them, not really highly nonlinear in the sense that the Newton path from x^0 to x^* exists but can't be followed by the ordinary Newton method. Thus, on an average, sophisticated damping or setting special options does not pay off for these examples. Rather, the advantages of the algorithms and codes will show up especially for large and numerically sensitive problems.

3.3 Special experiments

SST Pollution

Finding the stationary state of a (discretized) system of nonlinear PDEs represents a quite interesting and challenging problem class for nonlinear equation software. For PDE problems in one space dimension the application of quite a lot discretization techniques (e.g. finite differences, finite elements, collocation) leads to a system of nonlinear equations where, typically, the corresponding Jacobian shows band structure. Although the dimension of the discretized system is mostly of still moderate size, this fact should be exploited in order to solve the arising linear systems efficiently. As an example take the problem *SST pollution* (see Section 3.1) where $n = 404$ and — due to the simple structure of the diffusion term — the lower and upper bandwidth of the Jacobian turn out to be $ml = mu = 4$. The results of some experiments with NLEQ1 are summarized in Table 3.5. The gain from switching on the band mode option is obvious but not surprising. Note that the internal band mode variant for the numerical differentiation (just 9 evaluations of F in order to generate the Jacobian) works quite efficient, as the overall computing time is just increased by 52% by switching on this option whereas, in the full mode case, the additional amount of work turns out to be 137%.

NLEQ1-variant	#F	#J	CPU (s)
full mode, num. diff.	23	22	2237.4
full mode, anal. Jac.	23	22	944.3
band mode, num. diff.	23	22	84.6
band mode, anal. Jac.	23	22	55.7

Table 3.5 NLEQ1 for problem *SST Pollution*: band mode vs. full mode

Hydrocarbon

Another interesting problem class are (moderate) large nonlinear systems where the non-zero elements of the Jacobians show a sparse but irregular pattern. The performance of the corresponding Newton code NLEQ1S is now illustrated by means of three variants of the distillation column test problem *Hydrocarbon* (see Section 3.1). The first variant is the *Hydrocarbon-6* example which has been already used in the basic test set ($n = 29$). Second, the *Hydrocarbon-20* problem is considered, which consists of a set of $n = 99$ coupled equations. The number of Jacobian non-zero elements turn out to be $nnz = 740$, thus the portion of (structural) non-zero elements in the Jacobian (7.6%) is still not really small. Finally, by modeling a 40-stage column (instead of 6 or 20 stages respectively) one gets an, at least medium sized, system with 199 unknowns (*Hydrocarbon-40*). Herein, 1520 of 39601 matrix elements (3.8%) are structurally non-zero. The results of applying NLEQ1, NLEQ1S and HYBRJ1 (analytical Jacobian, standard options otherwise) to these problems are summarized in Table 3.6. Obviously, the use of sparse matrix techniques pays off only for sufficient large n — but this limit not only depends on the dimension and sparseness of the problem, but also on the hardware configuration of the computer at hand (vector, parallel or sequential machine). Furthermore, increasing the number of stages within this model increases the complexity of the nonlinear problem as well. This clearly shows up in the *Hydrocarbon-40* example, where HYBRJ1 fails to converge and NLEQ1S needs 3 times more iterations as for the *Hydrocarbon-6* example.

Problem/Code	#F	#J	CPU(s)
Hydrocarbon-6			
NLEQ1	7	6	2.6
NLEQ1S	7	6	2.5
HYBRJ1	16	2	8.8
Hydrocarbon-20			
NLEQ1	13	12	68.0
NLEQ1S	13	12	15.5
HYBRJ1	24	4	266.0
Hydrocarbon-40			
NLEQ1	22	21	677.9
NLEQ1S	22	21	56.7
HYBRJ1	*f4(218)	(14)	(8689.7)

Table 3.6 Performance of standard methods for *Hydrocarbon* problems

A quite interesting effect shows up, if one repeats these runs on another machine (SUN-Sparc1+) where the relative machine precision is a bit worse ($10^{-19} \rightarrow 10^{-16}$). NLEQ1S needs 3 additional iterations in order to solve the *Hydrocarbon 40* problem whereas all other runs remain uneffected. However, analyzing both runs, the reason for this difference is obvious. Up to the 20 th iterate there is a perfect coincidence, but then, very close to x^* where $\Delta x \rightarrow 0$, the sparse linear system solution is not accurate enough to guarantee quadratic (or at least superlinear) convergence for the Newton method. Hence, NLEQ1S needs 3 more (linearly convergent) iterations in order to meet the required accuracy of 10^{-10}. It is quite satisfactory, that the internal convergence monitor of the code realizes this behavior and issues a corresponding warning message. As the accuracy for the sparse linear system solution can be improved by reducing the chance of conditional pivoting one may solve the problem with modified linear algebra options. First, all matrices are factorized with the *Analyze-Factorize* routine MA28A and, second, for all these factorizations the chance of conditional pivoting is switched off by setting *thresh1*:= 1. The results are given in Table 3.7. The disturbance of the Newton convergence is successively removed, but the average CPU time for one LU-decomposition is increased. For this specific example the latter effect is quite small, but, in general, a decrease of the required tolerance is a better response to the above mentioned warning message of the Newton code.

Hydrocarbon-40	#F	#J	ϕCPU for LU
standard options	25	24	0.07
only A/F, *thresh1*= 0.01	24	23	0.16
only A/F, *thresh1*= 1.	22	21	0.20

Table 3.7 Performance of NLEQ1S with modified linear algebra options (on SUN-Sparc1+)

Expsin

For a last experiment, the artificial test problem *Expsin* is now used to discuss again one of the structural advantages (and the associated limit) of the damped Newton scheme (B) presented in Section 1.2. For that purpose recall the reflections on the Newton path made in Section 1.1.5. Now, for this simple test problem the manifolds with singular Jacobian can be explicitly computed. Straightforward calculations show, that

$$\text{Det}(J(x)) = 0 \text{ occurs for } \quad (\ x \equiv (x,y)^T \ )$$

$$y = x$$

or

$$y = \pm\frac{1}{3}\arccos(\frac{1}{3}) \pm \frac{j \cdot 2\pi}{3}, \quad j = 0, 1, 2, \ldots \ .$$

These lines (plotted in Fig. 3.3) separate six different, but symmetric, solutions. The behavior of the code NLEQ1 (with standard options) is now checked by a special experiment. The domain $[-1.5, 1.5]^2$ is "discretized" by selecting initial values

$$\begin{array}{ll} x_i := -1.5 + i \cdot \Delta & \\ y_i := -1.5 + i \cdot \Delta & \qquad i = 0, \ldots, 50 \\ (\, \Delta := 0.06 \,) & \end{array}$$

and NLEQ1 is started from all these "initial guesses". The result is illustrated in Figure 3.1. Herein, the different markers indicate toward which of the six solution points the code converges whereas white space indicates failure. This information totally reveals the topological structure of the problem. Except for 4 cases (λ_0 too large) NLEQ1 converges from the given starting point to the "associated" solution — even from "far away", if $y \approx -x$ (e.g. $x = (5, -4)^T$). In cases, where x^0 and x^* are separated by a line with singular Jacobian, NLEQ1 fails to converge — even from "nearby", if $y \approx x$ (e.g. $x = (1, 0.9)^T$). The result of repeating the experiment with HYBRJ1 is shown in Fig. 3.2. This method shows more convergent runs, but the nice structure, induced by the problem, of Figure 3.1 is smeared. Furthermore, a quite large number of "lie points" occurs, i.e. HYBRJ1 claims "solution found" without returning it. All starting points at which this happens are marked with a black filled circle.

The above mentioned 4 exceptions remove if NLEQ1 is used with the "extremely nonlinear" option switched on. The reason for this is illustrated in Fig. 3.3. Herein, the neighborhood of one of the critical starting points is magnified. The extreme nonlinearity shows up in the rapid change of the Newton direction near x^0. Obviously, the standard choice for the initial damping factor ($\lambda_0 = 10^{-2}$) — a compromise between robustness and efficiency — is not small enough to prevent the iteration from crossing a line with singular Jacobian. As this first trial iterate passes the natural monotonicity check, the code converges — but to a solution which is not "connected with x^0". Starting with a smaller damping factor ($\lambda_0 = 10^{-4}$) the code gets sufficient local information on the nonlinearity near x^0 in order to properly adapt all following damping factors. With that, convergence to the connected solution is guaranteed.

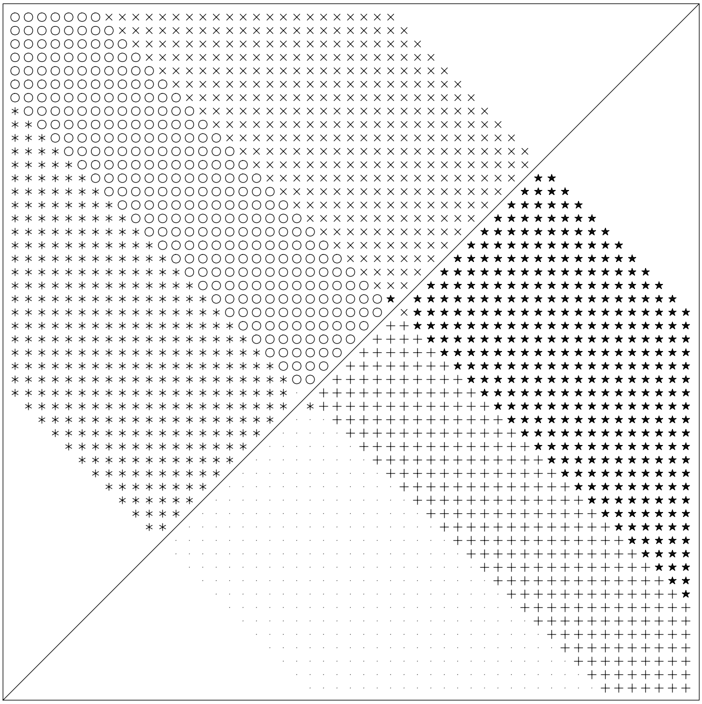

Figure 3.1 NLEQ1 — result of domain check for example *Expsin*

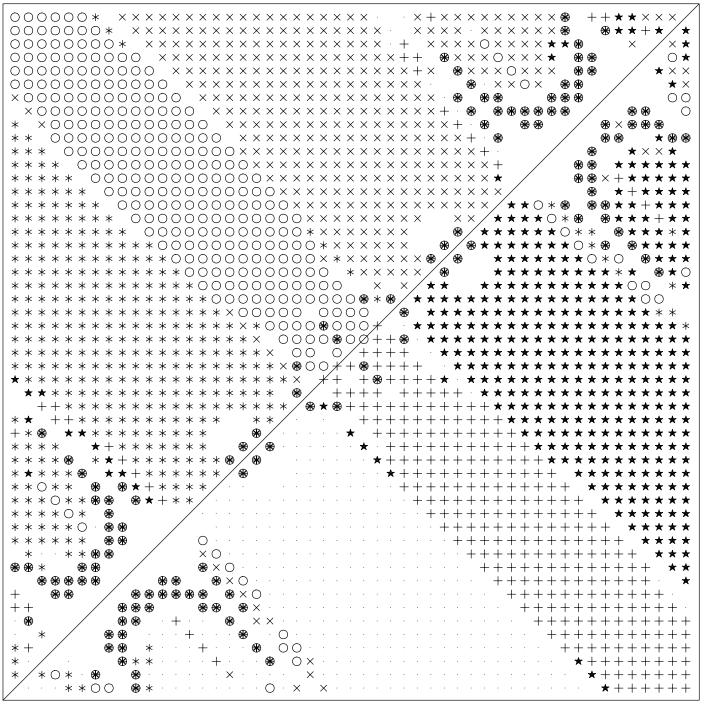

Figure 3.2 HYBRJ1 — result of domain check for example *Expsin*

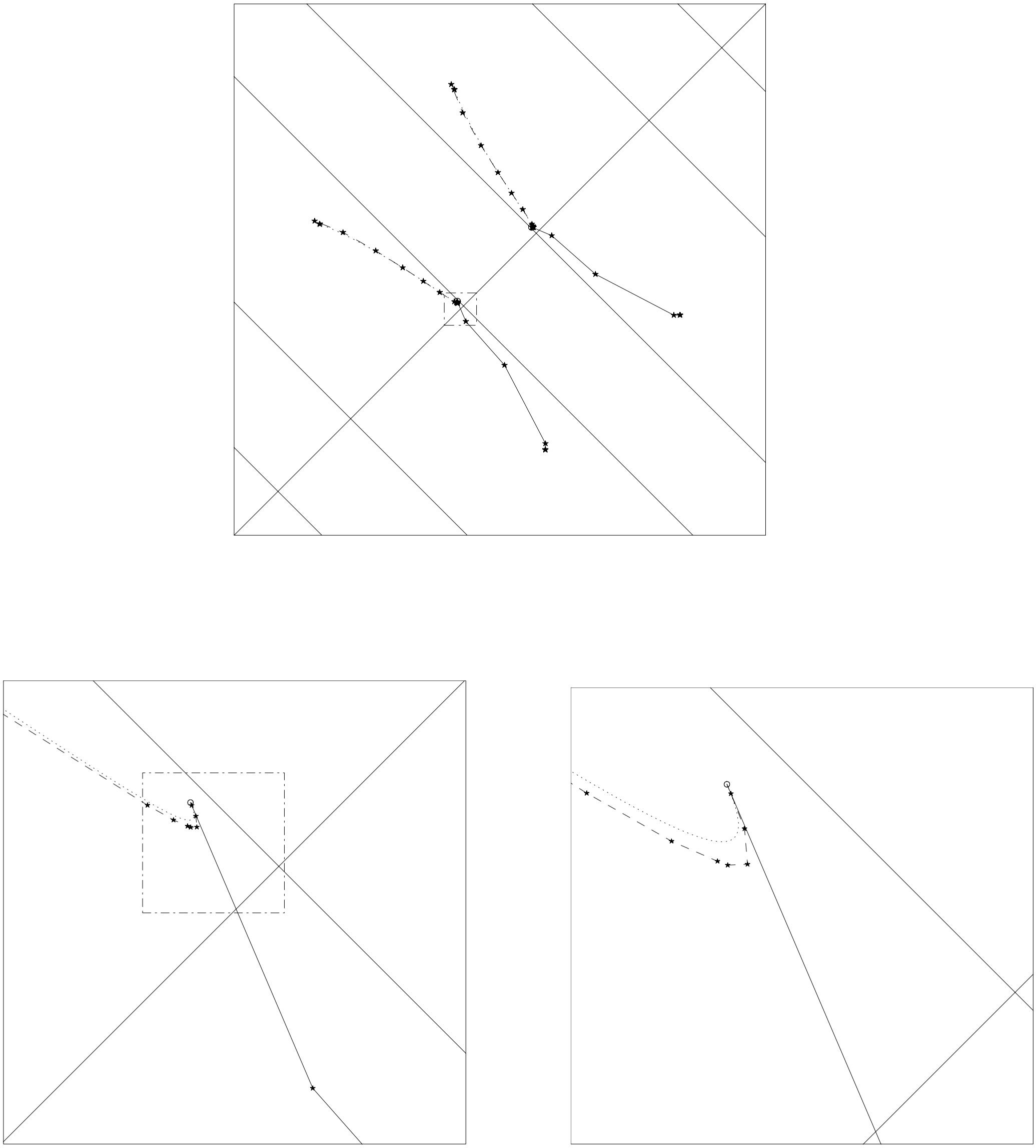

Figure 3.3 Neighborhood of a critical x^0. Original scale, first and second zoom in

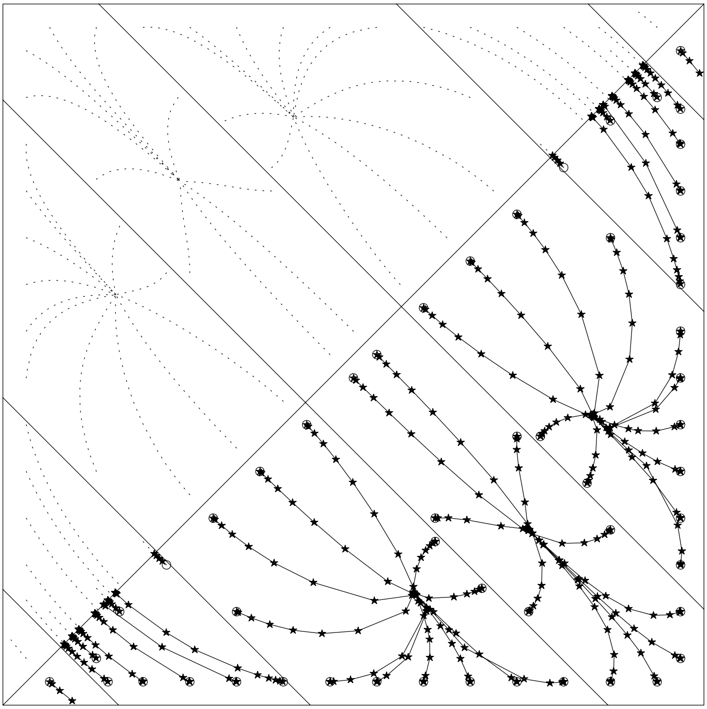

Figure 3.4 Damped Newton iterates and Newton pathes for example *Expsin*

Finally, Figure 3.4 makes again evident the extreme good accordance of theory, algorithm and code. For some selected starting points x^0 the associated Newton pathes $\overline{G}(x^0)$ (dotted lines) and the (extremely) damped Newton iterates (connected by solid lines) — from symmetric initial values $\overline{x}^0$ — are plotted.

Conclusion

Three new codes for the numerical solution of systems of highly nonlinear equations have been presented. For the first time, the affine invariant Newton techniques due to Deuflhard are available in the form of reliable and modern software. The essential features of the algorithms carry over to the codes. Systems up to a considerable size can be solved efficiently. The new codes turn out to be very robust.

References

[1] L. Armijo: *Minimization of functions having Lipschitz-continuous first partial derivatives.* Pacific J. Math. **16**, p. 1-3 (1966).

[2] H.G. Bock: *Numerical Treatement of Inverse Problems in Chemical Reaction Kinetics.* In: [16], p 102-125 (1981).

[3] C.G. Broyden: *A class of methods for solving nonlinear simultaneous equations.* Math. Comp. **19**, p. 577-583 (1965)

[4] P. Businger, G.H. Golub: *Linear least squares solutions by Householder transformations.* Num. Math. **7**, p. 269-276 (1965)

[5] P. Deuflhard: *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 (1974).

[6] P. Deuflhard: *A Relaxation Strategy for the Modified Newton Method.* In: Bulirsch/Oettli/Stoer (ed.): Optimization and Optimal Control. Springer Lecture Notes **477**, p. 59-73 (1975).

[7] P. Deuflhard: *Newton Techniques for Highly Nonlinear Problems - Theory and Algorithms.* Academic Press,Inc. (To be published)

[8] P. Deuflhard: *Global Inexact Newton Methods for Very Large Scale Nonlinear Problems.* Konrad-Zuse-Zentrum für Informationstechnik Berlin, Preprint SC 90-2 (1990).

[9] P. Deuflhard, E.Hairer, J. Zugck: *One-step and Extrapolation Methods for Differential-Algebraic Systems.* Num. Math., **51**, p. 501-516 (1987).

[10] P. Deuflhard, U. Nowak: *Extrapolation Integrators for Quasilinear Implicit ODE's.* In: P. Deuflhard, B. Engquist (eds.): Large Scale Scientific Computing. Progress in Scientific Computing Vol. **7**, p. 37-50, Birkhaeuser (1987).

[11] P. Deuflhard, W. Sautter: *On Rank-Deficient Pseudo-Inverses.* Lin. Alg. Appl. **29**, p. 91-111 (1980).

[12] J.J. Dongarra, C.B. Moler, J.R. Bunch, G.W. Stewart: *LINPACK.* SIAM, Philadelphia (1979).

[13] I.S. Duff: *MA28 — A Set of FORTRAN Subroutines for Sparse Unsymmetric Linear Equations.* AERE Report R. 8730; HMSO, London (1977).

[14] I.S. Duff: *Direct Methods for Solving Sparse Systems of Linear Equations.* SIAM J. Sci. Stat. Comput. **5**, p. 605-619 (1982).

[15] I.S. Duff, U. Nowak: *On Sparse Solvers in a Stiff Integrator of Extrapolation Type.* IMA Journal of Numerical Analysis **7**, p. 391-405 (1987).

[16] K.H. Ebert, P. Deuflhard, W. Jäger (ed): *Modelling of Chemical Reaction Systems.* Berlin-Heidelberg-New York: Springer Ser. Chem. Phys., vol. **18** (1981).

[17] R. Fletcher: *Practical Methods of Optimization.* Second Edition, John Wiley, 1987.

[18] HARWELL: *MA28 Subroutine library specification.* 28th May 1985.

[19] K.L. Hiebert: *An Evaluation of Mathematical Software that Solves Systems of Nonlinear Equations.* ACM Trans. Math. Software, Vol. 8, No. 1, p. 5-20 (1982).

[20] J. Molenaar, P.W. Hemker: *A multigrid approach for the solution of the 2D semiconductor equations.* IMPACT Comput. Sci. Eng. 2, No. 3, p. 219-243 (1990).

[21] J.J. Moré: *A Collection of Nonlinear Model Problems.* Preprint MCS-P60-0289, Mathematics and Computer Science Division, Argonne National Laboratory (1989)

[22] J.J. Moré, B.S. Garbow, K.E. Hillstrom: *Testing Unconstrained Optimization Software.* ACM Trans. Math. Software, Vol. 7, No. 1, p. 17-41 (1981).

[23] J.J. Moré, B.S. Garbow, K.E. Hillstrom: *User Guide for MINPACK-1.* Preprint ANL-80-74, Argonne National Laboratory (1980).

[24] U. Nowak, L. Weimann: *GIANT — A Software Package for the Numerical Solution of Very Large Systems of Highly Nonlinear Equations.* Konrad-Zuse-Zentrum für Informationstechnik Berlin, Technical Report TR 90-11 (1990).

[25] R. F. Sincovec, N. K. Madsen: *Software for Nonlinear Partial Differential Equations.* ACM Transactions on Mathematical Software Vol. 1, No. 3, September 1975, p. 232-260.

A. Program Structure Diagrams

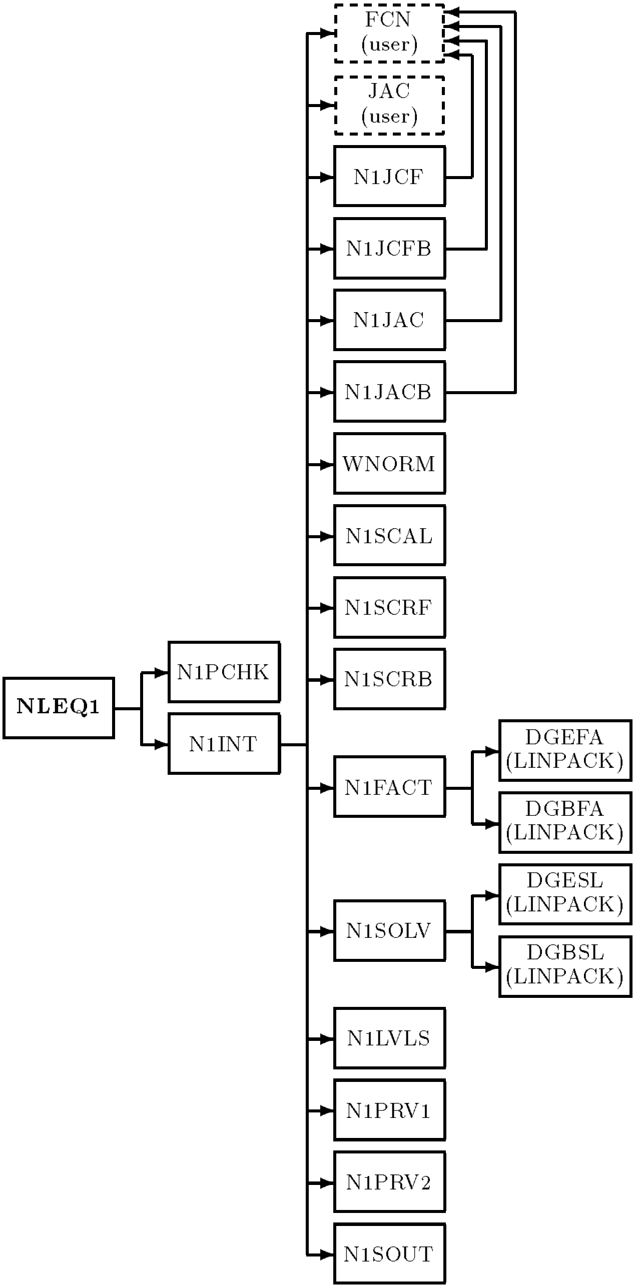

Figure A.1 NLEQ1: Program structure (subroutines)

routine	purpose
	Interface to the calling program
NLEQ1	Numerical solution of nonlinear equations — User interface and workspace distribution subroutine
N1PCHK	Checks, if input parameters and options have reasonable values
	Internal subroutines, realizing the algorithm (B)
N1INT	Main core subroutine of NLEQ1 — realizing the damped Newton scheme
N1LVLS	Descaling of the linear systems solution vector, computation of natural and standard level for the current (accepted or trial) iterate
N1FACT	Common interface routine to a matrix decomposition routine of a linear solver
N1SOLV	Common interface routine to a linear solver routine to be used together with the corresponding matrix decomposition routine called by N1FACT
WNORM	Computation of the norm used for the error estimate
	Jacobian approximation by numerical differentiation
N1JACB	Computation of a banded Jacobian
N1JCFB	Computation of a banded Jacobian, with steplength feedback control
N1JAC	Computation of a full mode storage Jacobian
N1JCF	Computation of a full mode storage Jacobian, with steplength feedback control
	Scaling subroutines
N1SCAL	Calculates the scaling vector for the damped Newton iteration
N1SCRF	Internal row scaling of the linear systems matrix A (full mode storage)
N1SCRB	Internal row scaling of the linear systems banded Matrix A
	Output subroutines
N1PRV1	Does print monitor output
N1PRV2	Does print monitor output (another format, different data as in N1PRV1)
N1SOUT	Output of the sequence of Newton iterates (or the solution only)
	Time monitor
MONON	Starts a specific time measurement part
MONOFF	Stops a specific time measurement part
MONINI	Initialization call of the time monitor package
MONDEF	Configuration of the time monitor — definition of one specific measurement part
MONSRT	Start of time monitor measurements
MONEND	Finishes time monitor measurements and prints table of time statistics
	LINPACK subroutines solving the linear systems
DGEFA	Matrix decomposition by Gauss algorithm (full mode storage)
DGBFA	Matrix decomposition by Gauss algorithm (band mode storage)
DGESL	Linear system solution routine to be used together with DGEFA
DGBSL	Linear system solution routine to be used together with DGBFA
	Machine dependent subroutines
D1MACH	Returns machine dependent double precision constants
SECOND	Returns a time stamp measured in seconds - used for the time monitor
	Routines to be supplied by the user of NLEQ1
FCN	The nonlinear problem function (required)
JAC	The Jacobian associated to the nonlinear problem function (optional)

Table A.1 purpose of NLEQ1 subroutines

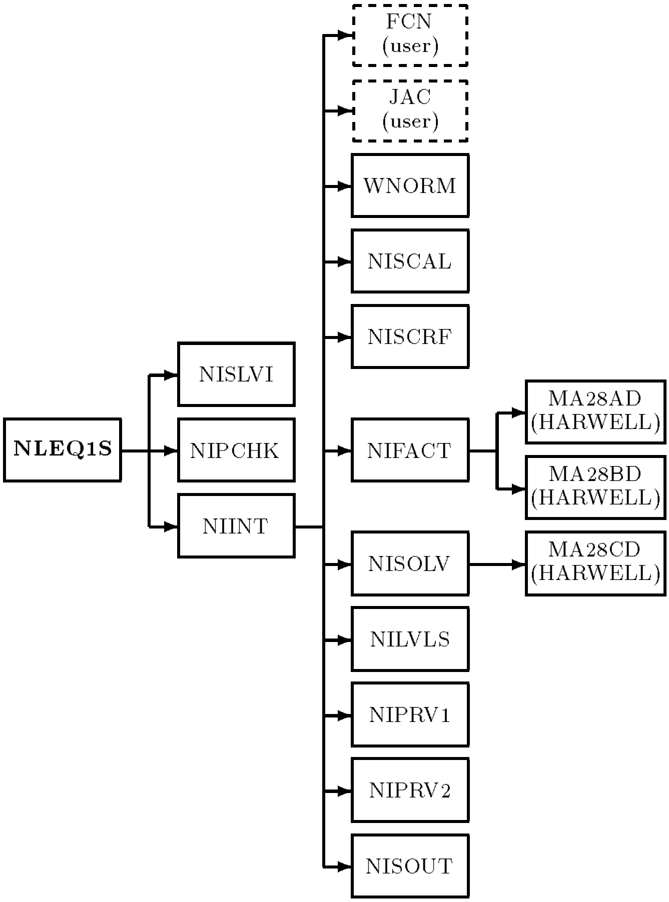

Figure A.2 NLEQ1S: Program structure (subroutines)

routine	purpose
Interface to the calling program and Initialization	
NLEQ1S	Numerical solution of nonlinear equations system with sparse Jacobian matrix — User interface and workspace distribution subroutine
NIPCHK	Checks, if input parameters and options have reasonable values
NISLVI	Performs initial settings for the sparse linear solver
Internal subroutines, realizing the algorithm (B)	
NIINT	Main core subroutine of NLEQ1S - realizing the damped Newton scheme
NILVLS	Descaling of the linear systems solution vector, computation of natural and standard level for the current (accepted or trial) iterate
NIFACT	Common interface routine to a matrix decomposition routine of a sparse linear solver
NISOLV	Common interface routine to a sparse linear solver routine to be used together with the corresponding matrix decomposition routine called by NIFACT
WNORM	Computation of the norm used for the error estimate
Scaling subroutines	
NISCAL	Calculates the scaling vector for the damped Newton iteration
NISCRF	Internal row scaling of the linear systems matrix A (sparse storage mode)
Output subroutines	
NIPRV1	Does print monitor output
NIPRV2	Does print monitor output (another format, different data as in NIPRV1)
NISOUT	Output of the sequence of Newton iterates (or the solution only)
Time monitor	
MONON	Starts a specific time measurement part
MONOFF	Stops a specific time measurement part
MONINI	Initialization call of the time monitor package
MONDEF	Configuration of the time monitor - definition of one specific measurement part
MONSRT	Start of time monitor measurements
MONEND	Finishes time monitor measurements and prints table of time statistics
LINPACK subroutines solving the linear systems	
MA28AD	Analyze-factorize subroutine for sparse linear system solution: needs to be initially called for factorizing a sparse matrix (using conditional pivoting)
MA28BD	Factorizes a sparse matrix, which has the same nonzeros pattern as some other sparse matrix passed before to a MA28AD call
MA28CD	Solves a sparse linear system with a matrix factorized before by a call of MA28AD or MA28BD
Machine dependent subroutines	
D1MACH	Returns machine dependent double precision constants
SECOND	Returns a time stamp measured in seconds - used for the time monitor
Routines to be supplied by the user of NLEQ1S	
FCN	The nonlinear problem function (required)
JAC	The sparse Jacobian associated to the nonlinear problem function (required)

Table A.2 purpose of NLEQ1S subroutines

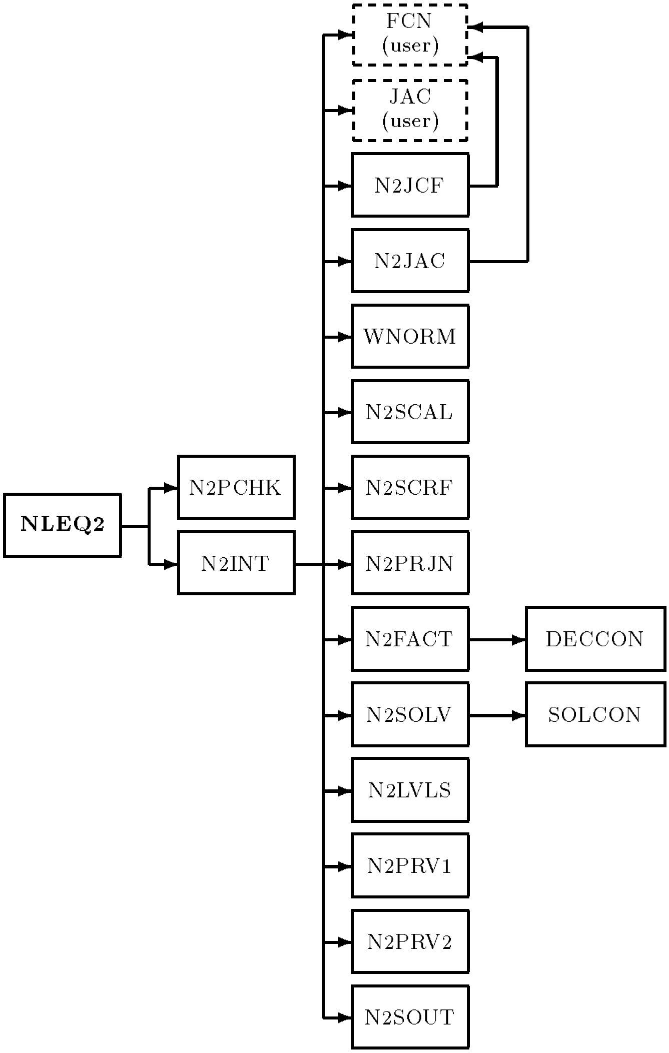

Figure A.3 NLEQ2: Program structure (subroutines)

routine	purpose
	Interface to the calling program
NLEQ2	Numerical solution of nonlinear equations — User interface and workspace distribution subroutine
N2PCHK	Checks, if input parameters and options have reasonable values
	Internal subroutines, realizing the algorithm (R)
N2INT	Main core subroutine of NLEQ2 — realizing the damped Newton scheme with rank-reduction option
N2LVLS	Descaling of the linear systems solution vector, computation of natural and standard level for the current (accepted or trial) iterate
N2PRJN	Provides the projection to the appropriate subspace in case of rank-reduction
N2FACT	Common interface routine to a matrix decomposition routine of a linear solver with determination of the matrix rank
N2SOLV	Common interface routine to a linear solver routine to be used together with the corresponding matrix decomposition routine called by N2FACT
WNORM	Computation of the norm used for the error estimate
	Jacobian approximation by numerical differentiation
N2JAC	Computation of a (full mode storage) Jacobian
N2JCF	Computation of a (full mode storage) Jacobian, with steplength feedback control
	Scaling subroutines
N2SCAL	Calculates the scaling vector for the damped Newton iteration
N2SCRF	Internal row scaling of the linear systems matrix A (full mode storage)
	Output subroutines
N2PRV1	Does print monitor output
N2PRV2	Does print monitor output (another format, different data as in N2PRV1)
N2SOUT	Output of the sequence of Newton iterates (or the solution only)
	Time monitor
MONON	Starts a specific time measurement part
MONOFF	Stops a specific time measurement part
MONINI	Initialization call of the time monitor package
MONDEF	Configuration of the time monitor — definition of one specific measurement part
MONSRT	Start of time monitor measurements
MONEND	Finishes time monitor measurements and prints table of time statistics
	LINPACK subroutines solving the linear systems
DECCON	QR-Matrix decomposition with rank determination (full mode storage)
SOLCON	Linear system solution routine to be used together with DECCON
	Machine dependent subroutines
D1MACH	Returns machine dependent double precision constants
SECOND	Returns a time stamp measured in seconds - used for the time monitor
	Routines to be supplied by the user of NLEQ2
FCN	The nonlinear problem function (required)
JAC	The Jacobian associated to the nonlinear problem function (optional)

Table A.3 purpose of NLEQ2 subroutines